Knowledge Discovery in Databases at a Conceptual Level

Dissertation

zur Erlangung des Grades eines

D o k t o r s d e r N a t u r w i s s e n s c h a f t e n

der Universität Dortmund
am Fachbereich Informatik

von

Timm Euler

Dortmund

2007

Tag der mündlichen Prüfung:	19.12.2007
Dekan:	Prof. Dr. Peter Buchholz
Gutachterin/Gutachter:	Prof. Dr. Katharina Morik Prof. Dr. Joachim Biskup

Danksagung

Zuerst danke ich Prof. Dr. Katharina Morik für die Betreuung dieser Arbeit und für ihre Bereitschaft, meinen etwas ungewöhnlichen Weg zu diesem Ergebnis zu unterstützen. Ebenso danke ich Prof. Dr. Joachim Biskup für die Übernahme der Begutachtung und seine Anregungen. Jean-François Boulicaut danke ich für einen Anstoß.
Allen Mitgliedern des LS 8 sei herzlich gedankt für die Schaffung einer angenehmen, respektvollen und unvergesslichen Arbeitsumgebung. Insbesondere Martin Scholz möchte ich für die gute Zusammenarbeit danken.
Den Herstellern der von mir evaluierten Software danke ich für die Überlassung der Lizenzen. Bei Andreas Greve möchte ich mich für seine Mühen bei der Installation der DB2-Datenbank bedanken.
Am dankbarsten aber bin ich für all die schöne Zeit, ob vergangen oder zukünftig, mit Verena.

Contents

List of Figures

List of Tables

1. Introduction

In recent decades, an ever-growing part of everyday processes, such as communication or trade processes, has substantially been transformed by using computers and networks for their administration. This trend is likely to continue in the 21st century, as the potential for digitalised services is still huge, be it in political, commercial, or health systems. It is, in many cases, only a secondary effect of this development that almost every computerised process leaves traces in the form of electronically stored *data*. This data may detail what has happened, when and where it has happened, who has been involved and so on. The amount of data that a typical organisation stores is growing fast and demands special tools to store and access it efficiently. Even where data collection has not been the end to which computerisation was the means, efficient data storage is becoming an urgent demand, for example for archiving, but also increasingly for legal and other reasons.

It has long been recognised that such data collections can provide added value to their owners, as they may reflect characteristics of the owners' business. Such characteristics are not likely to be easily seen by humans inspecting the data, as the sheer amount of data is usually far too high. This has led to the development of algorithms and tools that support data analysis in many different ways. *Data Mining* is an often-used general term for the discovery of hidden information in data. However, as was early recognised, there is a lot of work involved in a complete data mining project that does not strictly belong to the analysis step. In fact, a process of several distinguishable phases is needed. This process is generally referred to by the term *Knowledge Discovery in Databases* (KDD). A broadly accepted definition of KDD was given by Fayyad et al. (1996):

> The KDD process is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.

Here, a *pattern* is an expression in some language describing a subset of the data (or a model applicable to that subset). The term Data Mining has come to refer to a specific phase in this process, namely the phase where algorithms from the fields of Machine Learning or statistics are applied to a dataset in order to extract the patterns.

While a lot of research has been performed around this learning-based pattern extraction, fewer efforts have been put to the rest of the KDD process (an overview of this process is given in chapter 2). Most importantly, the collection and preparation of the relevant data turns out to be a complex and time-consuming endeavour in many projects. The main reason why data has to be prepared is that the algorithms in the mining phase cannot usually be directly applied to the "raw" data, in the form in which it has been collected. This part of the process, data preparation, is in the focus of this thesis, without neglecting its high interrelatedness with other phases. The overall aim of this work is to find out how both experienced analysts and beginners can be intensively supported by software during this phase, while allowing a smooth integration with the other phases.

Customer	Income	Gender	ProductGroup	Amount
10	30000	M	Notebooks	800
13	35000	F	Mobiles	140
13	35000	F	MP3-Players	50
13	35000	F	CDs	20
...	...	...	...	...

Figure 1.1.: Example for credit card transaction data, in a representation that is suitable for analysing typical transactions, for example by a clustering algorithm. Transactions are characterised by the type of product bought, the amount of money spent, and details about the person who made the transaction. Since one person can make several transactions, this introduces redundancy.

Good support for data preparation is highly needed, since both deciding how to prepare the data, and performing the preparation, involves solving complex problems. The following section discusses these problems in detail. Section 1.2 then outlines the approach taken in this work, and discusses how it solves or mitigates these problems. Section 1.3 provides an overview of the chapters of this thesis. Finally, section 1.4 lists this author's publications that have been used in this thesis.

1.1. Problem description

The starting point for data preparation is a given collection of structured data, together with some knowledge about what it represents. Collecting the data and knowledge about it is part of earlier phases in the overall KDD process, see chapter 2. For data preparation, it is crucial to understand that the data that one decides to use for knowledge discovery has in most cases been stored for other purposes, in particular supporting the operational demands of the data-owning institution. To support these purposes, database administrators employ techniques to organise their data such that efficient retrieval is possible, while at the same time redundant storage is avoided.

For data mining, data must usually be organised in a different way. Mining algorithms find patterns in a set of *examples* of a certain phenomenon, where each example must describe the phenomenon in question as detailed as necessary to find useful patterns. Most mining algorithms require all examples to be in a single table. Figure 1.1 shows a table with data about credit card transactions. Such a table might be used for analysing common features of typical transactions. However, the table exhibits some redundancy; it violates the usual design principles of relational databases (specifically, it violates the second normal form, see section 3.1.2). The institution that owns this data is unlikely to have stored it in this format. Data miners call this format *propositional*, since it is used by propositional learning algorithms (among others), which use some form of propositional logic to represent subgroups in the data. Thus it is necessary, for most mining approaches, to transform the data into this format, which is called *propositionalisation* (Knobbe, 2004). Propositionalisation often involves re-introducing redundancy that had been carefully removed by the database designers by splitting the data into separate

Table SalesInfo:

ShopID	ProductID	Week	Sales
12	430	1	16
12	430	2	15
12	430	3	18
...	...	...	...
13	5012	1	35
...	...	...	...

Table SeasonInfo:

Week	Christmas	SommerSale	...
1	0	0	...
...	...	...	...
30	0	1	...
...	...	...	...

Figure 1.2.: Input data for the drug store application. The shop and product IDs refer to additional tables. The "Week" attribute of the first table refers to the same attribute of the second table.

tables.

Data preparation involves much more than integrating data into one table, though. Before explaining the main preparation issues in detail, an example will serve to illustrate them.

1.1.1. Example KDD application

This example KDD project on sales data has been realised by Stefan Rüping (Rüping, 1999). While its data preparation part is not very complex, it serves well to illustrate the main issues. Note that even this less complex application took several months to develop. A larger KDD application is presented in chapter 5.

The input data for this project comes from a chain of drug stores. The stores sell a large range of products. For each product in each drug store, the given data contains the number of times it has been sold in a particular week, for a two-year period. The goal of this project was to predict the number of sales of certain products for the next week, given data from the last few weeks. Predicting this number is useful for reducing the amounts of a product that have to be kept in stock. Due to the requirements of the drug store chain, prediction had to be done separately for each particular product in every particular shop. Since there is a large number of products, and the drug store chain has 20 different stores, this amounts to a large number of individual applications of the mining algorithm. Therefore about 50 of the most interesting products have been selected. This means that the same learning task had to be solved for 1000 different selections of data of the same kind (i.e. having the same schema).

The input data is organised in a typical star schema: one relation (table) holds the information about the stores, another one the information about the products, while a central table keeps the sales information (number of times a product has been sold) for each product, each store and each week. For the data analysis, mainly the central table is needed in this application, but an additional table has been introduced by the data miner after a number of attempts to make useful predictions based on the central table alone had failed. For each week in which products were sold, this additional table specifies whether different seasonal events took place, like bank holidays or seasonal sales. Figure 1.2 shows the input data for this application.

Table MiningData_Shop12_Product430:

SalesWeek1	SalesWeek2	SalesWeek3	Christmas	SummerSale	SalesLabel
16	15	18	0	0	17
...	...	...	...	...	...
20	17	18	0	1	19
...	...	...	...	...	...

Figure 1.3.: Input data for the mining algorithm in the drug store application.

The learning/mining algorithm that has been used is the support vector machine (SVM), see section 7.2.5. For it to be applicable, the data must be represented as n-dimensional real vectors $\vec{x} \in \mathbb{R}^n$. The vectors represent the examples from which the patterns are to be found. Each vector is given a *label* that represents what is to be predicted. Training the SVM on such input will yield a function that can be used to predict the label of other vectors of the same kind, for which the label is not yet known.

In the drug store application, the label to be predicted is the sales information of the coming week, given some time point (the shop and product are fixed for each prediction task, as noted above). Several ways of setting up the example vectors can be imagined. One might try to use the complete sales data from before the given time point, or only parts of it. One might try to add information about the product or the shop to each vector. The representation that turned out to be successful, in terms of the achieved mining results, built the vectors by moving a window over the past sales data. Any time window of n subsequent weeks can produce one example (one combination of $\vec{x}$ and label), though in this application the time windows have been chosen so that they do not overlap. Further, for each window, the information about which seasonal event took place in the week whose sales are predicted, is added to the vector $\vec{x}$ (this information is available even for future weeks, since bank holidays etc. are fixed). Because the vectors must use real numbers (a technical input requirement by the SVM algorithm), whether or not an event takes place is indicated by the numbers 1 and 0.

Note, then, that some notion of *time* plays a particular role in this application. Yet, learning algorithms have no understanding of time. This is why time must be encoded in the representation, in this case using a fixed set of attributes for fixed-length time periods, the windows.

Figure 1.3 shows the data representation that is needed in this application for applying the SVM; the figure shows the input for only one of the 1000 learning tasks. In fact, the data in figure 1.3 is subject to another data preparation step before the SVM is applied: all sales values are scaled to the real interval [0..1], which is not a necessary technical requirement of the SVM, but often useful for training an SVM. This is not shown for better readability.

Figures 1.2 and 1.3 show the input and output of a particular preparation process. It can be seen that the data representation is changed and extended fundamentally between the two figures. The steps that are needed in this application example to create adequate input for the SVM are: selection of the shop; selection of the product; moving a window over the sales data for that product and that shop, and collecting the contents of all

windows in a new representation; adding the seasonal information that is relevant for each window; scaling the integer numbers to the new range between 0 and 1. These five steps have to be carried out in the same manner for the 1000 different combinations of shop and product.

This example illustrates the task of data preparation, which can be stated as follows for this work:

> Transform a given relational database to meet the technical input requirements of a chosen mining algorithm, such that the algorithm gives good results (finds valid, novel, and potentially useful patterns).

Note that this task description assumes that the decision which mining algorithm is to be used has already been made. Sometimes several algorithms are tried, then the task above has to be solved for each.

One can imagine that the first part of the task, meeting the technical requirements, can be solved by automatic approaches. A few attempts to do so have been made in the literature, see section 6.1.2. Essentially, these approaches are based on automatic planning. The planning goal is given by the technical input requirements of the chosen mining algorithm. However, for real-world KDD projects, such a planning goal is underspecified: meeting the technical requirements is possible in many ways, as the restrictions they impose on the input data format are not very strong. These restrictions are listed in section 2.1.3; they mainly involve a few data type constraints, based on an abstract notion of data type. Meeting these constraints is in no way sufficient to guarantee the success of the mining algorithm.

In other words, solving the second part of the task description above, namely choosing a representation that makes the algorithm successful, is much more complex (see also section 2.1.3). It requires human expertise, and cannot be automated currently. The space of possible input data representations is too big to be searched automatically, and no useful search heuristics are known. For humans, the best heuristic is a case-based approach, where the experience gained from earlier projects helps to guide the process in a new application. For example, the particular way of dealing with time in the above application can be useful in other projects, too. It is one goal of this work to support this case-based approach.

In a sense, data preparation, as a part of knowledge discovery, can be compared to software development. In software development, real-world requirements must be analysed first. They lead to a technical architecture for the software. The technical architecture determines the components of the software, and how they interact to achieve the main functionality. Specifying the components means to set up some technical requirements that the components must fulfil. But even realising the components, after they have been specified, involves human efforts and cannot be automated. Analogously, real-world goals for data analysis lead to the choice of a general method of analysis, in particular a mining algorithm, but also lead to some ideas for "information components" on which the analysis will be based. In the above example application, the information components are the time windows and the information about seasonal events. Both in software development and in data analysis, only human experts are capable of finding such components. But even after their specification, the way to realise them remains to be found by humans.

Decades of experience in software development have led to a number of heuristics, often called *design patterns* (Gamma et al., 1995), that can be used to guide programmers when realising the software components. To some extent, they may even provide guidelines for the overall architecture. The design patterns are abstractions of solutions that have been useful in the past, and describe their essence. The situation in knowledge discovery is not yet as advanced: not many useful design patterns have been found so far, despite many attempts to find correlations between data sets, real-world analysis goals, and suitable machine learning algorithms for their solution. These attempts are discussed in section 6.1.2. It turns out to be difficult to describe the essence of KDD solutions in general terms, so that they can be transferred to new KDD problems.

Therefore, this work identifies a suitable level of abstraction for the *modelling* of KDD solutions. This will provide the means to collect and describe KDD solutions in a detailed way, and to identify re-occurring patterns, which can be modelled in the same framework. In contrast to the software engineering design patterns, solutions and patterns modelled in this way will be directly *executable*, without the difficult and error-prone process of implementing an abstract software pattern in actual software. *Collections* of such patterns and successful solutions to KDD problems will make the experience that was gathered during their creation accessible to the public, and reusable by anyone. This will make data mining and the preparation of data for it much easier to perform, even for users without a strong background in the field, because they can rely on approved solutions which are modelled at an intuitive level and are ease to use. This re-usage framework thus addresses a larger audience than design patterns, for whose deployment expert knowledge is needed.

But why is it actually a problem that data preparation cannot be automated? The answer will be given in the following. The above application example helps to illustrate the main points.

1.1.2. Data preparation problems

It has been estimated (see (Pyle, 1999) and a 2003 KDnuggets poll[1]) that between 50 and 80 per cent of the time spent on a typical KDD project are dedicated to data preparation. The example above has illustrated the task, whose solution will be supported by the contributions of this thesis. Solving this task poses the following particular problems.

Exploration When developing a new KDD application, neither the mining algorithm nor the data representation that will give the best results are known beforehand. Several approaches usually have to be tried. Even in the example KDD project above, where the mining algorithm and the outline of its task were chosen early on, many options remain to be explored. As noted there, several ways of representing the examples for learning can be imagined. Even after deciding for the window-based approach, the number of weeks for which past data is used to predict the future (the number n above) is open. Which kinds of seasonal events should be included for prediction is also unclear. Prediction could be done for the following week given data for some weeks, but also for the week after that, depending on the requirements of the drug store chain. Note that any decision to change one of these

[1] `http://www.kdnuggets.com/polls/2003/data_preparation.htm`

options may involve changing several parts of the preparation process, not only the application of the mining algorithm.

Complexity Data preparation processes can be rather complex, involving dozens of single transformation steps (where the steps are in themselves not trivial, as will become apparent). Chapter 5 presents an example for a larger application. It has many steps, and each step produces a new, different (intermediate) representation of the input data. Keeping an overview of the many intermediate steps and their results is difficult: on the one hand, the data to be analysed must be known very well, on the other hand, a high-level overview of how it can be used in relation to the mining algorithm(s) must be kept.

Education It was already mentioned that the best heuristic for human KDD developers, when exploring a new application, is to rely on their experience about successful KDD projects from the past. But knowledge about past projects, and about what were the decisive factors that made them successful, is easily and quickly lost, and is difficult to transfer between humans. As noted above, suitable "design patterns", an analogue from software engineering, hardly exist yet. Such design patterns would describe the essence of a number of previously developed, successful solutions of KDD problems (for example how to deal with time).

Programming Early knowledge discovery projects had to rely on low-level programming to perform the required data transformations. Developing such programs is expensive in terms of human work efforts, and typically results in a poorly documented collection of programs that are difficult to maintain and difficult to reuse on similar problems. Even simple tasks, like the scaling of the values to [0..1] in the application above, become cumbersome by having to repeat them many times. The exploration of several preparation options (see above) is very tedious indeed with such programs. Note also that in the example from section 1.1.1, essentially the same task has to be solved 1000 times, but each time on different selections of the data. Organising this in a typical data querying language like SQL is not easy. These problems have led to the development of a number of commercial software tools that support various data transformations in a graphical way. However, as this thesis will show, these tools still suffer from a number of shortcomings; for example, they do not represent the data in an adequate way, they do not allow to represent many similar tasks in one model, and they typically offer only a few types of transformations that still leave the process more complex than it could be.

Large data sets Real-world KDD projects are usually fraught with the difficulties of processing very large amounts of data. The mining algorithms typically have super-linear runtime, and their implementations are therefore usually not capable of processing more data than fits into main memory of the system that runs them. Yet, assembling and preparing the data for this step, as well as the productive deployment of learned results, requires the handling of much larger amounts. This first of all means that efficient data storage is required, suggesting the use of information systems. Secondly, performing even simple transformations may consume a lot of

time, which prohibits a style of development in which every step in a complex transformation has to be executed immediately, before further steps can be specified. Unfortunately some data preparation tools enforce just this style of development. Similarly, low-level programming, error-prone as it tends to be, requires too many test cycles to be convenient on large data sets.

The present thesis provides analyses, and develops a framework with an implemented system, that help to solve or mitigate these problems. The following section explains the general approach.

1.2. Overview of the approach

The overall goal of this work is to *ease the work on data preparation in data mining for human users*. Therefore it examines how data preparation can be presented to users in intuitive terms that describe what is done using KDD-related vocabulary. For example, some notion of *time*, or the idea of a *label* (see above), should be made explicit. Rather than having to use general-purpose devices, like programming languages, data miners should be supported by software that directly employs this vocabulary. The term *conceptual level* is used for this description level. It is contrasted with a technical level which does not use KDD-specific concepts; for example, describing a KDD process in SQL would be located at the technical level. Section 2.2 discusses the two levels in more detail.

Any approach towards easing data preparation for humans must account for the explorative nature of data preparation, and should also address the other problems above. In particular, reusing approved solutions developed by others should be supported, to address the problems listed under "education" above. This work presents an environment, called *MiningMart*, that employs a conceptual level of KDD process descriptions. It can be used for the graphical modelling of KDD applications, their organisation into subparts, their immediate execution on relational databases, and their publication and reuse based on an open metamodel. Figure 1.4 shows a screenshot of the KDD project from section 1.1.1 above, as modelled in MiningMart. This application model represents the preparation and mining for all 1000 learning tasks involved in that project, which can be executed by a single mouse click.

MiningMart addresses the problems listed above by the following measures.

Providing a catalogue of transformation operators By providing a set of transformation operators that solve standard tasks, the development of complex data transformations can be reduced to combining such operators into directed acyclic graphs, in which each operator processes the output of the previous one(s). This avoids low-level programming completely and frees users from having to learn any formal languages. It also allows an intuitive graphical representation of the graphs. The operators are organised into groups according to the mining-related preparation problems they solve.

Providing a catalogue of preparation solutions Based on the above reference list of preparation operators, models of complete preparation processes can be created

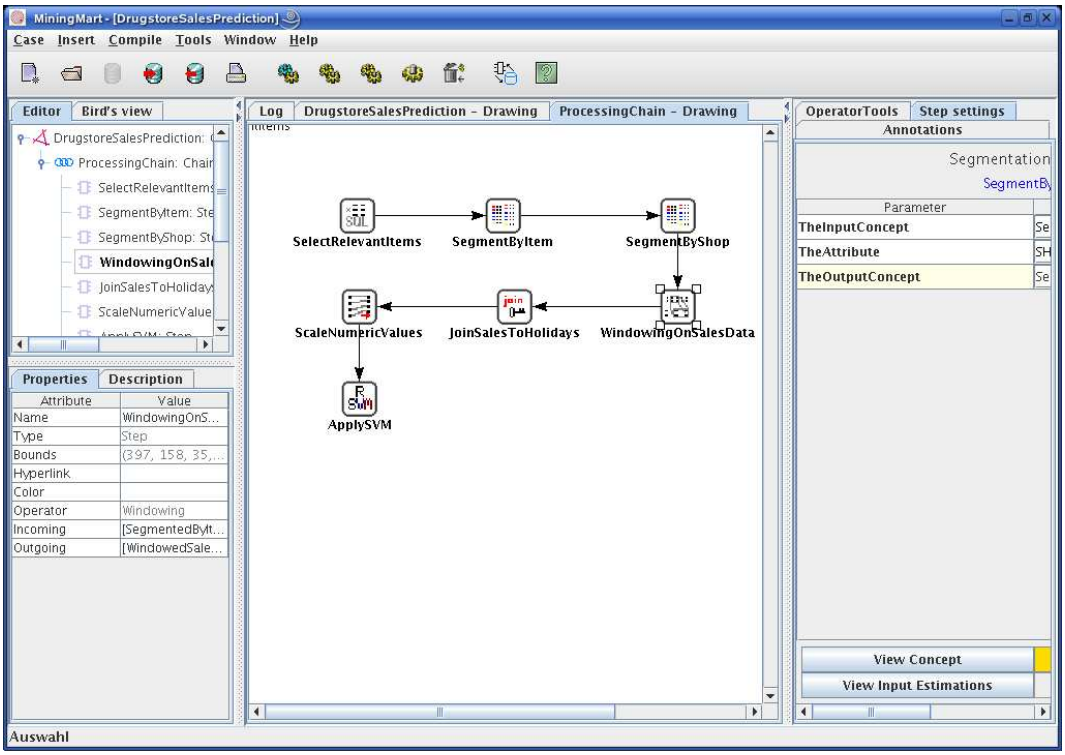

Figure 1.4.: The KDD application from section 1.1.1 in MiningMart.

and *published*. Then processes that have been successfully used in KDD applications can be shared among experts, can be directly re-used on different data sets, and can be used to educate new KDD analysts. A central web portal for publishing and downloading these processes has been set up. This web-based repository of KDD solutions can help to reduce the time needed for finding solutions to KDD problems, by providing examples of solutions that have worked previously. Functionality that supports re-using such solutions is included in MiningMart. Further, among the complete processes for whole applications, *subprocesses* can be identified that solve typical subproblems in data preparation, and that can be published separately as *templates* for those subproblems. These templates can be identified manually, but an automatic method of discovering them in a collection of complete solutions is also presented. The templates go beyond design patterns from software engineering (see above), since they are directly executable, and can be reused without a strong background in programming. All these aims, which are related to the "education" problem above, could not be achieved previously due to the lack of a common model for KDD applications.

Providing a suitable abstract data model Publishing preparation solutions is of little use if the data that they have been applied to is not also described. Publishing the data itself is undesirable, but publishing a model of the data *schema* is sufficient.

This work employs an abstract data model that is suitable for this purpose. Having such a model has other advantages. It allows to abstract from the given data and to use a view on it that is more oriented towards the tasks to be solved (although the abstraction should not be too high, as a data miner must know their data well). The specification of the preparation operators discussed above can be given in terms of this data model, making the operators applicable to any technical data source as soon as the latter is mapped into the abstract model. Thus a chain of operators becomes easily reusable by simply mapping the model of its input data to a new data source. The results of each application of a transformation operator are automatically documented. Further, using an abstract data model moves the many intermediate data sets that are created in a typical preparation process into the focus. These data sets are important artifacts of the KDD process, as they are excellent sources of information, or interfaces, for planning the further development of a preparation solution, or integrating additional tools or analyses (like data visualisation methods). One requirement that the abstract data model must fulfil therefore is the ability to structure these data sets, so that the user can keep a clear overview of them. This addresses the "complexity" problem above.

Speeding up development Developing a preparation solution consumes expensive human time, while performing the actual data processing consumes cheap computer time. MiningMart provides means to speed up development, and to reduce the number of test cycles during the development of a new KDD application. The latter mitigates the problem of handling large data sets. One of the ways to achieve the former is *pseudo-parallel* processing: a process is modelled once but can be executed on a number of identical tables. In the drug store sales analysis example above, this means that 1000 learning tasks are hidden behind one "conceptual" model of the KDD process they have in common; the system controls the many data sets involved, so that users can concentrate on modelling. This also serves to reduce the complexity of the task.

Processing data in an information system Information systems have been developed over decades towards powerful data storage systems. In most KDD applications the data to be analysed is initially stored inside a (often relational) database, anyway. MiningMart thus exploits the efficient data storage capabilities of such systems, avoiding the need introduced by many KDD tools to transfer data to other systems.

Easing documentation Representing the data transformation operators, as well as the data models they operate on, explicitly is in itself a much better documentation than can be provided by low-level programming code. The main reason is that these explicit elements are located at different levels of abstraction: the parameters of the operators, the operators themselves, the subprocesses, and the processes can be seen to form a hierarchy, whose explicit representation allows top-down browsing of a KDD application model. These levels are reflected in the KDD system, but also in the web repository. Additionally, all elements of these levels are documentable by free text annotations. Finally, publishing background information on

the purposes, goals, and achievements of each KDD application that is available in the web repository is supported.

The general MiningMart approach has been outlined in (Kietz et al., 2000; Kietz et al., 2001; Morik & Scholz, 2004) and (Euler, 2005d). MiningMart is based on a declarative metamodel, to be explained in section 6.3, which has first been documented in (Morik et al., 2001) and later in (Scholz & Euler, 2002). A sketch of the major components of the MiningMart framework is given in figure 6.1 on page 97.

The MiningMart system is compared to other KDD tools in this work, based on the following contribution.

Enabling objective comparisons of KDD tools Having specified a catalogue of preparation operators allows to compare different software packages that are designed for KDD applications with respect to the extent to which they support these operators. Detailed comparisons of such software tools are very useful for institutions that would like to start KDD projects, in order to find the product that matches their particular requirements best. However, for an in-depth comparison, the available operators are not the only criterion. Chapter 8 of this thesis develops not only further, more detailed criteria for KDD tools, but also presents a *methodology* by which these criteria can be derived and evaluated objectively. The methodology is applicable to other types of software products as well, and is adaptable to different evaluation purposes.

1.3. Overview of this thesis

This thesis first gives some background on the knowledge discovery process, following an informal, but widely accepted terminology standard, in chapter 2. The chapter introduces many notions and specific problems that are addressed in subsequent chapters. It argues that support for KDD, like for other application domains, is best given at a "conceptual" description level, which uses the concepts and ideas of KDD rather than general-purpose notions. The following chapters demonstrate how such a conceptual support can be achieved.

Chapter 3 chooses a conceptual data model for structured data that is tailored towards the specific needs of knowledge discovery. The chapter begins with a definition of physical, logical and semantic data models. The relational data model is identified as being still the most common technical-level model that represents input data for KDD. An entity-relationship model is found to be a suitable conceptual-level data model, by listing semantic notions (abstractions) that must be supported for KDD. The chapter ends by discussing the role of data types and data characteristics.

Chapter 4 then examines the data processing parts of a KDD process, in particular data preparation. Based on an analysis of the major preparation tasks that are needed for KDD, it explains the notion of a preparation *operator*. A comprehensive list of data preparation operators, together with their role for the preparation for mining, is given in appendix A. The operators are specified using the conceptual data model from chapter 3. The set of operators is divided into groups according to which mining-related preparation purpose they serve. The data model from chapter 3, and the process model presented in

this chapter, are established as two *dual* views on the preparation process. Control of the process can be executed from either view, but together they provide more information and flexibility to the user than each alone.

These theoretical chapters are followed by an illustration of their basic concepts using a model KDD application, in chapter 5. The model application is based on two real-world applications and involves rather complex data preparations. The chapter explains the application in terms of the two dual conceptual views; this level of description can be contrasted with the technical realisation of the model application, which is given in appendix D.

Chapter 6 introduces the MiningMart environment, which supports the two dual conceptual views on the KDD process, based on a metamodel for modelling KDD processes. A public repository of such process models is presented, which serves as a knowledge portal to KDD users, enabling the flow of knowledge between experts in the field and from experts to inexperienced users. The chapter discusses the central issues of reuse and adaptation of KDD process models. The chapter also introduces *templates* for solutions to typical, small data preparation problems; these templates are also published in the repository. Appendix B lists many templates developed for this work. They provide a public, modular collection of recipes for solving typical preparation tasks for KDD. A method for discovering such templates automatically is also presented in chapter 6.

Chapter 7 provides a more technical description of how certain parts of MiningMart have been implemented by the author. The implementations concern mainly the data model (section 7.1), but also some important operators (section 7.2). Besides these major system parts, further functionality has been added by the author, based on the analyses from previous chapters. In particular, the model application, the requirements for reusability from chapter 6, and a number of evaluation criteria from chapter 8 suggested certain additions that will be described. This includes measures that support the execution-independent development and the direct reuse of data and process models.

Chapter 8 uses the insights from the previous chapters, as well as practical experiences made implementing the model KDD application, to develop detailed criteria for the evaluation of KDD software tools, with a focus on data processing. The methodology for identifying these criteria is presented. It allows to tailor the evaluation towards different purposes or audiences. The criteria are used to evaluate a number of KDD tools, which exemplifies the practical applicability of the methodology.

Each of the above chapters contains a summary with the main arguments that are needed to follow the overall work.

Finally, chapter 9 summarises this thesis, outlines its contributions, and discusses open issues for future work.

1.4. Publications

Parts of this thesis have already been published in journals, conference proceedings and as technical reports. This previously published work is listed in the following.

A brief overview of the MiningMart approach (chapter 6) has been given in (Euler et al., 2003), which is joint work with Katharina Morik and Martin Scholz. The author of this thesis contributed 33% to this paper.

Work on the data model (chapter 3) was preceded by the paper (Euler & Scholz, 2004), which discusses using ontologies for MiningMart. This paper was joint work with Martin Scholz to which the author contributed 50%.

The model application from chapter 5 has been briefly presented in (Euler, 2005b), and also (Euler, 2005d). The latter paper mainly provides the presentation of the MiningMart web repository as a knowledge platform. Chapter 6 is based on it.

The preparation operators presented in chapter 4 and appendix A are a more detailed version of (Euler, 2005c). A slightly modified version of that paper has been published in a journal (Euler, 2006).

The software comparison methodology (chapter 8) has been published in (Euler, 2005a), including the evaluation of KDD software tools.

Using the support vector machine for feature selection, as discussed in section 7.2.5, has been documented in the technical report (Euler, 2002a).

The MiningMart meta model M4, discussed in section 6.3, is documented in the technical report (Scholz & Euler, 2002), which was joint work with Martin Scholz to which the author of this thesis contributed 50%.

Two other technical reports by this author which are related to the work in this thesis, though not used in the thesis as such, are (Euler, 2002b) and (Euler, 2002c).

2. Knowledge Discovery in Databases

This chapter sets the framework for the subsequent discussions by introducing the KDD process according to an informal, yet well-known standard. It has long been recognised that several phases of the process can usefully be distinguished. The early conceptions of these phases (for a brief but clear overview see (Gaul & Sauberlich, 1999)) were rather similar to each other and evolved quite naturally into the CRISP-DM standard (Chapman et al., 2000), which has established a common terminology. This standard is presented in section 2.1. Section 2.2 then presents the notion of describing a KDD process at two different levels, a technical and a conceptual, task-oriented one. The connection between the two levels is explored in subsequent chapters.

2.1. Overview of the KDD process

A complete KDD process consists of much more than just the application of learning algorithms to data. The various tasks around data analysis can be assigned to different phases of the process, which provides a good conceptual overview of KDD, though it does not imply that there are no interdependencies between the phases. The cross-industry standard CRISP-DM (see (Chapman et al., 2000)) is the most established conceptualisation of the KDD process that also provides a common terminology, and it will be used here to introduce basic concepts around KDD.

In CRISP-DM, a number of generic tasks is defined that need to be solved during most KDD projects. The generic tasks are intended to be general enough to cover all possible situations in the KDD process. They are categorised into six phases that make up the process; while there is a natural sequence for these phases, a typical project will experience backtracking and reviewing earlier phases in the light of intermediate results. The phases form the top level in this hierarchical process model; the generic tasks form the second level. At a third level, the generic tasks are specified and detailed according to the data mining context, that is, the given situation (for example the application domain, the type of mining problem etc.). Finally, the fourth level records the details of a concrete process instance.

The following subsections describe the six phases of the KDD process. The description draws on CRISP-DM (Chapman et al., 2000) and (Pyle, 1999). While a lot of details are omitted, what follows will provide an understanding of the context in which this work is placed. Other general introductions to KDD include (Witten & Frank, 2000).

2.1.1. Business understanding

This phase might more generally be called "Application understanding", as this process model is not restricted to business projects. In this phase, the most important task is to

identify the goals of the project in terms of the application domain or the end-users of the KDD results. In a health-related project, such a goal might be to understand the main factors affecting the success of a treatment of a particular disease. While the discussion in this phase should be non-technical, a clear understanding of what is to be achieved is needed. In particular, success criteria must be established. Further, a detailed project plan that lists the resources, requirements, risks, costs and possible benefits of the project should be made.

The purpose of this phase is to provide the data analysts with an understanding of the background of the KDD application. A common danger in data analysis is to find a pattern that is already known to domain experts, as such patterns often comprise coarse relationships in the data that show up easily (Morik et al., 2005). If this happens, time and effort are wasted unfruitfully. To avoid it is only possible with a good understanding of the domain and of the questions that the data owners would like to have answered. A simple example of directly addressing the needs of the data owners is to use mining to maximise the return on investment (ROI) in businesses, as modelled by a (heuristic) function (Ling et al., 2005; Singh et al., 2005).

Recently, Pechenizkiy et al. (2005) have attempted to support the identification of relevant issues in this phase by adapting frameworks from the field of Information Systems that relate technical systems to their organisational and external environments. Their work is preliminary and has provided only rather superficial models so far. But they suggest lines of further research, in particular to examine the key factors of successful use of data mining systems. Some experience-based ideas on these factors can be found in (Hermiz, 1999) and (Coppock, 2003). Both these authors point out that there exist many data analysis-related problems to which data mining is not the appropriate solution. But even if it is appropriate, there is a danger of gaining insights that are not actionable in the given organisation. In such cases, the solutions to the data mining tasks that were found cannot be translated, for some reason, into actions that help to achieve the overall goal of the institution (see also Piatetsky-Shapiro in Wu et al. (2003)). This highlights the need for involving the organisational environment in planning a KDD project. Coppock (2003) stresses that this is often a communication problem between technical and business experts; but similar problems can also occur between different groups of technical experts, as vividly described by (Freeman & Melli, 2005). One remedy suggested by Kohavi et al. (2004) is to present preliminary analysis results to the business experts in order to gain a common understanding (based on concrete material) of what could and should be done. Additional material about this phase can be found in (Pyle, 1999).

2.1.2. Data understanding

With a firm background about the application at hand, the available data collections should be examined. Because the data is often collected for other purposes than knowledge discovery (see the introduction to this chapter), simply accessing and assembling the data may be a nontrivial, time-consuming task, depending on the sizes of data sets, the way they are distributed in the organisation, and privacy or security issues. Usually, data from different computer systems, collected at different times in various formats, must be brought together. Often the data is copied to a central site, a *Data Warehouse* (Inmon, 1996; Meyer & Cannon, 1998), which provides a regularly updated, static snapshot of

the dynamic operational data. Such data warehouses are not only used for KDD, but for many different kinds of analysis. However, in an increasing number of institutions, data sets with the same or a similar structure exist at several locations, for example in the individual stores of a supermarket chain. In these cases it may make sense to mine the data sets locally and combine the results. This is referred to as *Distributed Data Mining*. A good introduction is (Park & Kargupta, 2002).

It is common to describe data sets as collections of *tables* which each have a number of *columns*. However, not all data sets easily allow this representation; for example, log files of web servers need extensive processing before (the relevant parts of) their contents can be represented as tables. A sub-discipline of KDD called Web Mining has emerged to deal with such data (see for example (Kosala & Blockeel, 2000)). But even when the data is available in tables, there is often a lot of further processing needed to allow automated analysis. This is the subject of the next phase in the KDD process, data preparation, and of the following chapters.

Data understanding involves more than the assembly of data, though. A description of the tables and their attributes is needed that includes

- the quantity of data (number of rows for each table);
- the meaning, or content description, of each of the attributes, whose names are often cryptic;
- the data type of each attribute (strings, numbers, dates, times, texts, media files and other types may occur);
- statistical information about each attribute, such as which different values it takes, how they are distributed, what the minimum, maximum, mean or median values are (if applicable), possible correlations with or dependencies on other attributes, and so on;
- information about the quality of the data, that is, whether the collection process was reliable, or whether "gaps" occur in the data (called *missing values*), and how they are represented;
- information about the integrity of the data, for example, whether different tables can be linked via key relationships;
- information about the completeness of the data, that is, to what extent all available data could be collected, and whether the available data may be considered representative of the population of entities that it describes;
- information about any known regularities or dependencies in the data, which is often based on prior (background) knowledge; for example, one can expect that data about vehicles would not contain an odd number of wheels, so that any such entries can be suspected to be errors;
- and any other information that describes the data as it is and serves to judge its relevance for the project, or to highlight "unreliable" or poorly understood parts of the data.

Specific attributes must not have missing or empty values
Specific attributes must be realised as real numbers
Only continuous or only discrete attributes are accepted
Discrete attributes must be realised as sets of boolean attributes
Not more than N values of discrete attributes are accepted
Only 2 classes (2 distinct values for the label attribute) can be used
Continuous attributes must be normalised to the same/a given range
Key attributes are not accepted as part of the representation
Only one data table is accepted

Table 2.1.: Possible input requirements of mining algorithms (after (Kietz et al., 2000)).

In other words, in this phase the data is made accessible for knowledge discovery, and basic information pertaining to it is collected and should be documented. Many of these issues have already been solved if the data-owning institution has created a data warehouse. In these cases data understanding becomes much easier. It can be further simplified by using graphical visualisations of various data properties.

The purpose of collecting the above information is to translate the business goals, which were stated in the previous phase in terms of the application, to technical goals. This is a very difficult task that requires much expertise in data mining, and much communication with the application experts (Kohavi et al., 2004). The goal is a first project plan that describes how to prepare the data and perform the analysis in the following phases. This plan would include the type of data mining problem that is going to be solved (see the mining phase, 2.1.4), and the choice of one or more supporting software tools; chapter 8 deals with tool selection criteria. Based on this plan, a first justification for the likelihood of success for the project can be given. However, such a plan must be considered tentative, as it is likely that insights from later phases will lead to some revisions.

2.1.3. Data preparation

The previous phases can be said to prepare the data analyst. The data itself, after assembly, is very likely to also need further preparation for a number of reasons:

- Technical requirements of mining algorithms: As noted in the introduction, data mining algorithms impose restrictions on the input data, such as accepting only continuous attributes (see section 3.3.1 about data types), or requiring the same scale for all continuous attributes. Table 2.1, adapted from (Kietz et al., 2000), lists possible input requirements of mining algorithms.

- Introducing background knowledge: Often, information that is not yet captured in the data can be added to ease the task for the mining algorithm (for mining, see the following phase). For example, a person's birthday can be used to compute their current age or even, more abstractly, their age group according to some background criteria.

- Removing background knowledge: Contrasting with the previous point, it may also make sense to remove patterns from the data that are already known and are likely to distract the mining algorithm from more subtle, new patterns. An example is to remove trends from time series data.

- Controlling the process: Some mining algorithms internally change the data. For example, some decision tree algorithms (e.g. (Quinlan, 1993)) internally group values of discrete attributes. It is usually preferable to control such changes by performing them explicitly beforehand; at least, this should increase the understandability of the mining result.

- Exposing information content: Mining can be significantly sped up if only relevant parts of the data are used. Some attributes may be redundant and can be removed, which is called feature selection; here, some automatic methods are available (Liu & Motoda, 1998). In contrast, feature *construction* attempts to construct new attributes based on the given ones, with the aim of making some "hidden" information directly available to the mining algorithm. This is discussed further below.

- Changing the data organisation: Except for multirelational learning algorithms (see e.g. (Wrobel, 1997; Muggleton, 1995; Knobbe, 2004)), most mining algorithms expect the data in a single table. This may require joining different data tables into one, a process called *propositionalisation* in the context of data mining (Knobbe et al., 2001); it is known to be an effective way of gathering information from more than one table into one, for mining purposes (Krogel et al., 2003). Many propositionalisation approaches automatically add columns with information from other tables to one central table which is then used for mining. However, such automatic approaches do not scale well to complex and large databases. A careful manual selection of columns to be added is required in such cases.

  Records of this single propositionalised table often have to be organised in a specific way. For example, in association rule mining, a transaction table is expected (Agrawal et al., 1993). Another example is learning from time series, where a series of windows (fragments from the original series), instead of the given series of single values, may be needed to enable mining (compare section 1.1.1).

- Cleaning the data: As was mentioned before, the data may contain gaps due to the way it was collected. It is important to distinguish between *empty* and *missing* values (Pyle, 1999). Empty values represent absence of a feature, such as a non-existing driving license for underaged persons. Missing values are gaps that could have been filled, such as sensor data that is not collected due to malfunctioning of the sensor. Empty and missing values usually have to be removed or filled, as many mining algorithms cannot deal with them. Further, errors (like typing errors) from the storage process may have to be corrected, and outliers (records with extreme or rarely occurring values) should be taken care of. These tasks are frequently in themselves the subject of data mining projects (e.g. in (Loureiro et al., 2005)).

- Sampling the data: Large data sets can pose a significant performance challenge both for preparation and mining. It may be necessary to reduce the amount of

data used for analysis, but this has to be done without skewing the representativeness of the data, if possible. A comprehensive overview of sampling approaches for knowledge discovery is given by Scholz (2007).

- Accounting for technical constraints: The tools which are used expect the data in a specific format that may have to be created. These technical requirements stem from the tools that are used, not from the mining algorithms as such.

Most of these reasons for preparing the data are unique to KDD (or data analysis in general), i.e. they are not given in other application areas where data transformations are employed (discussed in section 4.1). Data cleaning is an exception, it is an issue that is also often solved for building a data warehouse, for example.

As explained in chapter 1, most data mining algorithms use a propositional data format, in which the examples that the algorithm learns from are given in a *feature*-based representation, each example taking a particular value for each feature. This format is also called *attribute-value* format, and this work, with its focus on data preparation, mainly uses the term *attribute* instead of *feature*, though the latter is more common in the machine learning literature.

Some of the above problems, like introducing background knowledge or exposing information content, are usually solved by *feature construction*, i.e. by introducing new attributes/features that are not present in the original data, but contain the added information. In a typical KDD application, many important features are constructed manually, and this is a major part of the preparation process. Specific operators for it are given in this work, see section A.5. Automatic feature construction methods also exist and may complement the manual preparation (Liu & Motoda, 1998).

With this background, the following *high-level preparation tasks* have been identified in this work. In a typical KDD application, some or all of these tasks may have to be solved. Chapter 4 introduces a number of basic operations (data transformations) for each task which can be used to solve it.

Data reduction Often the data may have to be reduced because the chosen mining algorithm does not scale to the available amounts of data. Besides random selection (sampling) and selection based on data properties, the *aggregation* of data can be useful. Aggregation changes the level of detail of the information in the data, for example by computing a monthly average for daily amounts, which would reduce the amount of data by a factor of 30.

Propositionalisation This is the task of integrating data spread over several tables, to allow the application of a learner that requires a single data table as input. See section 1.1.

Changing the organisation In many applications it is necessary to change the representation of the data rather fundamentally, as exemplified in the example application in section 1.1.1. This often involves introducing attributes, i.e. metadata (schema-level elements), based on values of a different attribute, i.e. based on data (instance-level elements), and/or vice versa. In other words, the way the data is organised is changed.

Data cleaning See above.

Feature construction As explained above, new information or new representations of given information are often essential to allow learning. Numeric data may be discretised or scaled to a new range, or new attributes may be computed in many different ways from given attributes. The term *feature construction* is used here to be consistent with the machine learning literature, although *attribute construction* could be used as well.

These tasks may help to structure a complex preparation process. For example, data reduction should be among the first tasks to be addressed in such a process, since it may reduce the time required to execute the following tasks. Propositionalisation may be another task to be solved early, as well as creating the required organisation of the data. Feature construction can then be among the last issues to be addressed.

At the heart of the KDD process, in the mining phase, lies a machine learning algorithm (the terms learning and mining are often used synonymously, also in this work; the term modelling is also used in the literature, but is used in this work to refer to the creation of data or process models). Data preparation changes the representation of the data, thus following the fundamental insight from machine learning research that the representation of examples to learn from has often more impact on the quality of results than the learning algorithm itself (e.g., (Langley & Simon, 1995; Morik, 2000)).

The data preparation phase is in the focus of the present work. As mentioned in the introduction, it is also very often the most time-consuming phase in the KDD process, consuming between 50 and 80% of the overall time, according to (Pyle, 1999) and a 2003 KDnuggets poll[1]. Chapter 4 refers to the tasks above and introduces specific data transformations that can be applied to solve them.

2.1.4. Mining

Once the data is prepared, a mining algorithm can be applied to it. CRISP-DM differentiates between the following mining problem types, of which several can be combined in a KDD project:

- Segmentation (more often called *clustering*), the division of a data set into meaningful or significantly different subsets;
- Concept description, the derivation of an *understandable* description of (a subset of) the data. Discovering an interesting subset of the data in the first place, before describing it, is called *subgroup discovery*;
- Dependency analysis, the search for significant dependencies between data items, or between events represented by the data;
- Classification, the assignment of class labels to unlabelled data, based on a model built from labelled data;

[1] `http://www.kdnuggets.com/polls/2003/data_preparation.htm`

- Prediction, also called regression, the assignment of a predicted, continuous value to data, based on a model built from data where this value is available.

CRISP-DM also mentions data description and summarisation as a data mining problem type, but assigns it to the data understanding phase because it is seen as preparatory to the actual mining; hence, statistical and visualisation techniques are used to address this type. More sophisticated methods, such as the discovery of rules to describe patterns in the data (e.g. (Münstermann, 2002)), are seen as concept descriptions. Learning in *structured* output spaces, like learning parse trees for natural language sentences, has recently been reduced to classification (using many possible class labels, for example one for each possible parse tree given an input sentence), by using a joint representation for input and output and learning a discriminator function that returns one label, given the input (Tsochantaridis et al., 2005).

Segmentation, concept description and dependency analysis are called *descriptive* mining tasks; classification and regression are *predictive* tasks.

For each problem type, a number of machine learning algorithms exist that automate the task. For this work, not much about these algorithms, nor further details about the problem types, needs to be known. Introductory material can be found in many textbooks from machine learning and data mining, including (Mitchell, 1997) or (Witten & Frank, 2000). Nevertheless, there are some important issues to be aware of in the context of this work.

- Selecting a problem type and machine learning algorithm determines only the technical requirements on the data representation that is used as input for mining. Other aspects mentioned in section 2.1.3 remain open. This is why data preparation is an explorative process, as mentioned in the introduction.

- Most machine learning algorithms have superlinear runtime complexities, which restricts the amount of data that can be used for training the models. For many algorithms, training set sizes beyond main memory capacities are ruled out, though specific implementations to work on databases have been created for some settings (e.g. (Münstermann, 2002)). Often this restriction introduces the need for data reduction (see the data preparation phase).

- For the tasks of classification and regression, two sets of *labelled* data are needed: one for training the model and one for controlling its generalisation performance (to avoid the so-called overfitting). These sets are called training set and test set. The label represents the class or the amount to be predicted. Acquiring labels can be expensive, but the two sets must be big enough to be representative of the underlying population. For data preparation it is important to note that both sets must be prepared in exactly the same way.

- For classification and regression, all the unlabelled data that is not used for training and testing has to be prepared in rather the same way as the labelled data, if it is to be used during deployment (deployment is explained in section 2.1.6). It would not make sense to train a model on one representation and have it make predictions based on a different representation. For training, the data set size is often simply

adjusted to the available main memory. But for deployment, the size of the data set poses a significant performance challenge on the data preparation phase, if, as is typical, the unlabelled part of the data set is large. For example, to predict marketing response behaviour of customers, a company with millions of customers may use only the data of some tens of thousands of customers for training, but then apply the model to all its customers. Thus all customer data goes through the preparation process.

- The tasks that need to be solved in the mining phase, which include training, testing and the tuning of certain algorithm-dependent parameters, can lead to complex experiments with nested applications of basic operations (Mierswa et al., 2003). Adequate support must be given to the user for such experiments; see section 4.5.

- Some mining algorithms allow, or even require, some post-mining operations, such as pruning of a decision tree or a rule set. Since these operations concern the learned model rather than the data sets, they are assigned to the mining phase in this work. Though in some literature a specific *post-processing* phase is introduced as part of the KDD process, in this work, the term "post-processing" is used to refer to data processing that follows the mining phase (see section 2.1.6), while "preparation" or "pre-processing" precede mining.

- In distributed mining settings, special algorithms may be applied that combine locally learned models. Assuming homogeneous data schemas at each local site, this requires to prepare the data at each site in exactly the same way before learning locally. Thus it makes sense to define the preparation process once and apply it several times. Chapter 6 describes how this can be achieved.

2.1.5. Evaluation

In the previous phase, the evaluation of the learned model using a test set or other measurable criteria serves to refine the model until a satisfactory quality is achieved. However, in CRISP-DM, there is an extra evaluation phase whose subject is the whole process so far. Each phase of the process offers a lot of options, so a lot of decisions must be made during a KDD project. In the light of the mining results achieved so far, those decisions should be reviewed. Each phase provides new insights into the data and new ideas about what could be mined. It may now be desirable to repeat some parts of the process with modifications. Also, the results so far should be documented, including the steps that led to them.

2.1.6. Deployment

In the final phase, the data mining results are mapped back to the original goals and objectives set out in the first phase. Because often the objectives are to improve operations and processes that the data owning organisation performs, this means to integrate the results into existing work flows. This is a nontrivial endeavour which should be included into the project plans from the outset.

When the mining problem was descriptive, its results are new insights into the entities represented by the data that was analysed. Deployment may here be limited to the creation of a report for the management of the organisation, who can decide about new policies for the operational processes of the organisation. For example, the physicians of a clinic may change or implement certain treatment procedures after KDD has identified problematic patient subgroups. This kind of deployment of mining results is beyond the scope of KDD.

When the mining problem was predictive, one of its results is a function that predicts labels for new, unlabelled data. The main deployment action is to apply this function on such data, and to use the predicted label in a business process. For example, if sales of a product are predicted for a certain time point in the future, acquisition and stocking of the product can be planned, like in the example application in section 1.1.1.

Another issue in predictive settings is that the label on which the mining algorithm is trained may need to be transformed during data preparation; for example, some neural net implementations require numerical input in the real interval [0..1]. Only reversible transformations can be used in these cases, because the predictions made by the algorithm have to be translated back to the original label values, before they are usable. Thus a data post-processing step is required after mining. See also section 7.2.6.

Both for descriptive and predictive problems, another deployment issue often arises. The KDD results reflect the state of the data owning organisation, or the entities it deals with, as far as they are represented by data; reports on this state become outdated over time. So quite often the KDD process, now that it is documented and justified by good results, will be executed on a routinely basis, in regular intervals, to update the model(s) on new data (Brachman & Anand, 1996) (one of the few KDD *application* reports that mention this is (Hereth & Stumme, 2001)). This kind of regular mining may be executed by nonexpert staff. It may require to integrate a predictive model into operational computer systems, or to regularly provide a descriptive mining algorithm with new training data. For example, when predicting customer response behaviour in marketing, one may want to send letters automatically to those customers that were predicted to respond positively. This kind of integration would in turn require to also integrate all data preparation steps that were applied. In many institutions this poses a problem as it concerns data from operational systems, whose capacities may not suffice to perform complex data preparations beside the actual business operations (Kohavi et al., 2004). A specific business process may thus have to be defined in order to perform regular deployment.

2.2. Two levels of KDD descriptions

Computer scientists are used to the idea of realising an abstract, task-oriented model of an application or a domain in lower-level languages. These languages are typically general-purpose programming languages: powerful formalisms that must be handled by highly skilled experts. Yet the application or domain in question may be rather simple. In this view there are two different description levels of the application. One of them is much closer to human understanding of the domain, but it has to be translated to a lower-level formalism. In the case of Knowledge Discovery in Databases, early work

had to rely on low-level approaches to data preparation and mining because nothing else was available. The field was just emerging and too complex to have developed a higher-level understanding quickly. The present work attempts to summarise and extend the higher-level concepts in data preparation that have emerged in the decade that KDD has existed. This section introduces the two description levels and their connection, while the following two chapters elaborate on details of both levels, but in particular the higher level. Chapter 6 then shows how both levels can be formalised in a metamodel of KDD processes to enable collaborative work on blueprints of KDD solutions.

The technical level describes the data and any operations on the data independently of any application purpose. The higher level deals with KDD concepts: the role that the data plays, and the purpose of applying a preparation method, are seen in the context of the knowledge discovery application. This level will therefore be called *conceptual*. Similar level distinctions have been made in the knowledge representation literature. Newell (1982) argues that different levels of computer systems have the following common attributes: there is a *medium* that is to be processed (for example, bit vectors or logical expressions); there are *components* that provide the primitive processing capabilities (like registers); and there are laws of composition and behaviour that assemble components into a system and determine how the system behaviour depends on the components' behaviour. Newell sees the introduction of a new level justified when a system can be defined in terms of the medium and components of that level alone, without reference to any of the previously used levels; at the same time, the new level must be reducible to the next lower level. Newell used the term "symbolic level" for general-purpose computation, and introduced a "knowledge level" to be used for the modelling of intelligent agents. The corresponding medium is knowledge, and the components are goals and actions. Adapting this idea to the present work, its goal is to introduce the conceptual level for data preparation in KDD, whose medium is given by data sets, and whose components are processing operators. It makes perfect sense to describe data preparation in terms of this level, without recourse to any other level. Yet there is also a possible reduction to the next level below, the technical, implementation-dependent level, and this reduction is discussed wherever appropriate in the following chapters.

An extension of the multiple-level approach is the multiple-model approach which underlies the work on the KADS project (Schreiber et al., 1993c). Various models are used there to highlight selected aspects of the (knowledge-based) system that is to be engineered; irrelevant aspects can be neglected in the construction of one model because they are accounted for by another model. For example, an organisational model can be set up to reflect the socio-organisational environment and its interaction with a system, or a task model shows how overall system tasks are decomposed into subtasks (Schreiber et al., 1993b). The term "model" emphasises the fact that these views on a system are products of an engineering process. It would not be inappropriate to say that a conceptual *model* for data preparation is presented in subsequent chapters; however, to emphasise the reducibility to the technical level, the term *level* is preferred.

Regarding the development of a KDD process, the two levels are characterised by different aspects:

- Technically, the syntactic well-formedness of all operations with regard to the underlying technical data model (see chapter 3) must be ensured.

- On the KDD level, what makes the KDD process successful can be more easily understood, documented and administrated (modified, stored, and re-used) by using the concepts relevant to KDD.

One may relate the different levels to different types of users of data collections: while for example database administrators are typically concerned with the technical level, KDD experts and statisticians (data analysts) tend to think and work on the conceptual level, as they cannot take the application out of their focus.

One of the purposes of this work is to argue that the general understanding of KDD has matured enough to allow explicit software support for the conceptual level, with automatic administration of the technical level. This has the following advantages:

- Many important aspects of the application at hand remain implicit if only the technical level is considered. This was demonstrated in a different domain by Clancey (1983), who analysed the rules used in the expert system MYCIN and found that they were difficult to understand or modify by people who had not invented them, even though the formalism in which the rules were expressed was explicitly chosen to be simple (in order to make automatically generated explanations of the system's reasoning understandable to humans). Clancey showed that rules played different roles and were based on different kinds of justification, and suggested to encode this type of background knowledge, as well as domain knowledge (from medicine), in MYCIN. This corresponds to an explicit introduction of conceptual-level elements.

- If the higher level is made explicit, the lower one can be hidden, as will be demonstrated in subsequent chapters. A software that hides the technical level can present the entire KDD process to a user in terms of the concepts introduced in section 2.1. This eases the development of and daily work on KDD applications.

- Several technical realisations of the same conceptual model can be supported by a system. Section 1.1.1 introduced a KDD application in which the same learning task had to be solved on 1000 data sets of the same kind. Similarly, in distributed data mining, frequently the same data schema is used at several local sites, so that a decision is made to prepare or analyse data locally before combining the results. In these cases all the identical technical processes can be hidden behind one conceptual model of the process.

- By making the conceptual level explicit, it is automatically documented and can be stored and retrieved for later reference. KDD applications without conceptual support have often produced good results which could later not be reproduced because essential know-how about, for example, the data preparations or model parameter settings, was lost, e.g. (Wirth et al., 1997). Thus the educational potential of conceptual software support should not be underestimated.

- Self-explanatory, task-oriented names for the data entities can be used on the higher level, extended by free-text annotations, rather than the cumbersome abbreviations often used on the technical level.

- The conceptual level lends itself well to graphical representations, allowing a largely graphical interaction between the user and the KDD system.

- The conceptual level allows to waive the use of formal languages for data processing, making solutions of processing tasks accessible to a wider audience. This is an important aim achieved by the framework of chapter 4.

- The conceptual level serves to focus a user's efforts on relevant analysis tasks while freeing them from technical details. It can help to develop clearer ideas of what is to be done, by giving "mental tools", or by providing constraints that disallow badly formed or semantically invalid models. An example can be found in (Schreiber et al., 1993a), where the development of precise (conceptual) models of problem-solving algorithms revealed a clearer picture of their differences and commonalities than existed before.

- Independence of the conceptual level allows to reuse parts or all of a conceptual process model on new data by simply changing the mapping to the technical level. Though this may require conceptual adaptations, it saves much effort compared to a development from scratch. This is true even if only solutions of subtasks are reused. Due to its work saving potentials, adaptability of KDD process models is becoming an important requirement for modern KDD software (see section 6.6 and criterion 1 in appendix C).

- At a conceptual level, different KDD projects can be compared much easier than on a technical one. This allows groups of KDD experts to work collaboratively by sharing collections of conceptual descriptions of successful projects, in which standard recipes for the solution of certain (sub-)tasks may then be identified; see section 6.6.

- The use of the conceptual level allows the comparison of different software tools by abstracting from technical details. Criteria for comparison can be formulated on the conceptual level, which makes their communication and application much easier. Chapter 8 presents such criteria and a comparison of tools based on them.

- In grid-based knowledge discovery, which is still a research area, the KDD process has to be set up declaratively before its execution, as the computational resources for execution are allocated on demand (AlSairafi et al., 2003). Conceptual modelling is very suitable for this declarative development.

The following chapters show that the connection between the two levels is well understood in KDD. Chapter 3 applies the two-level view to the given data, while chapter 4 is concerned with processing the data. Chapter 6 presents a KDD system that provides support for most of the conceptual aspects from chapters 3 and 4 in all its interactions with users. In particular, section 6.1 discusses more literature showing that the conceptual level has been missing or incomplete in many previous approaches to KDD systems. Chapter 8 compares current KDD software packages based on criteria that include the conceptual aspects and other important issues.

2.3. Summary

A complete KDD process has several phases. This work focuses on data preparation. Detailed reasons why data sets may have to be prepared have been given in section 2.1.3: technical requirements of mining algorithms are listed in table 2.1, but as noted in chapter 1, the main preparation task is to find a representation based on which a learning algorithm can find novel and interesting patterns. This task cannot be automated. However, important subtasks have been identified in section 2.1.3 that can be used to structure a preparation process, and thus to guide human users (the subtasks are: data reduction, propositionalisation, changing the organisation of the data, data cleaning, and feature construction).

In the explorative preparation phase, users can be supported through *models* of the preparation process that use KDD-oriented vocabulary. Section 2.2 has argued that such models can and should use a separate, KDD-specific description level, meaning that the KDD process can be sufficiently described using only elements of the models, without recourse to lower system levels. The term "conceptual level" is used in this thesis for the higher modelling level. The following two chapters introduce the conceptual level for the data and for the data transformations.

3. A Conceptual Data Model for KDD

This chapter examines the data as given for KDD and presents an abstract view of the data, a conceptual data model, that can be used to control the preparation process. Section 3.1 prepares the ground by making some observations about what kind of data are usually given for analysis, followed by a discussion of certain semantic abstractions that have been identified in the literature to classify various conceptual data models. Section 3.2 then identifies an Entity-Relationship (ER) model as providing very adequate support of the KDD process. Finally, section 3.3 specifies some information to be attached to the data model; this information is useful for controlling the development of the KDD process at the conceptual level.

3.1. Background

This section first gives some background on data models (section 3.1.1). It then considers the data as it is usually given for KDD and identifies a data model for the technical description level (section 3.1.2). Finally it introduces possible aspects of conceptual level data models from the literature (section 3.1.3), based on which a suitable model is explained (section 3.2.2).

3.1.1. Types of data models

A data model consists of a set of abstract modelling constructs used to describe the data from a part of the real world. Data models differ mainly in the types of modelling constructs they support explicitly, implicitly or not at all. The most common modelling constructs are listed in section 3.1.3. In every data model a distinction is made between the structural description of a database, called the database schema, and the database itself, which is called an instance of the schema.

Usually, three types of data models are distinguished:

Physical data models are used to handle the concrete storage of data. Such models may include information about data records, files, file locations, access rights etc. They represent the system view of the data.

Logical data models support views of the data that are more abstract but can be processed by a computer directly. The most important example for this group is the relational data model, which is implemented in relational database management systems. Other examples include the historical network and hierarchical models as well as the more modern object-oriented models. Logical data models are not concerned with concrete storage of the data, but still view the data as collections of records; they can be mapped directly to a physical data model.

Dependency on:	DBMS class	Specific DBMS
Dependency of:		
Conceptual models	no	no
Logical models	yes	no
Physical models	yes	yes

Table 3.1.: Dependency of the data model types on DBMS classes and specific DBMS, adapted from (Batini et al., 1992).

Semantic data models are the most abstract models. They allow designers to represent data rather in the way the data arises in the real world. They are independent of any realisation in a computer system. They provide a list of (often graphical) abstraction concepts used to model objects, attributes or relationships.

Semantic data models are sometimes also called *conceptual* models (Nijssen, 1977; Batini et al., 1992). Although this term clashes with its use in the well-known ANSI/X3/SPARC database management system framework (Tsichritzis & Klug, 1978), where conceptual schemas correspond to what are called logical models above, it will be preferred in this work because it matches the notion of a conceptual description level introduced in section 2.2.

Another source of confusion is the fact that certain data models can play the role of both conceptual and logical models. Table 3.1 (adapted from (Batini et al., 1992)) may help to clarify the terminology: it basically states that logical data models define classes of database management systems (DBMS) that support them directly, such as relational databases, while conceptual models are independent of any database system. In this sense, object-oriented models can be logical if a database implementation uses object-oriented structures, but may also be used as conceptual models and then mapped to a relational logical model, for example.

Data models concern what is called domain knowledge in the knowledge representation literature, as recognised there (Wielinga et al., 1993) (obviously there are many other types of knowledge that one might want to represent, such as inference knowledge). In knowledge representation, a classical distinction is made between "levels" of knowledge that can be represented by the same structure (Brachman, 1979): implementational, logical, epistemological, conceptual and linguistic. A structure represents implementational knowledge if it models data structures or pointers; it represents logical knowledge if its elements are predicates, propositions or logical connectives; it represents epistemological knowledge if it provides the *notions* of concepts, attributes, types of relationships etc.; it models conceptual knowledge if its elements are concepts of the domain in question, for example cats and dogs; and it represents linguistic knowledge if its elements are words of a concrete (natural) language, like "dog". Using this categorisation, conceptual data models as defined in the present work provide epistemological elements.

When talking about the building blocks, or constructs, of conceptual data models, a number of different *meta models* can be distinguished. A meta model prescribes the constructs available to form a conceptual model. Overviews and comparisons of classic meta models are given in (Hull & King, 1987) or (Peckham & Maryanski, 1988). Among

the most well-known meta model is the Entity-Relationship (ER) model with its various extensions.

The main advantages of using conceptual data models, rather than physical or logical ones, are as follows (Hull & King, 1987):

- Approximation to human thinking. Concepts and abstractions in conceptual models reflect the way humans organise their world more closely than logical models. Initially, conceptual models were developed for the design of information systems; they were expected to support the process of deriving logical schemas. It is hoped, with some justification (Formica & Missikoff, 2004), that formalising aspects of the world in conceptual models is more intuitive for humans than in logical models.

- Increased separation of semantic and physical components. Even in logical data models like the relational model, which provide a very useful abstraction from the physical data level, users must follow rather technical details in order to state moderately complex queries. Consider the task of finding all components of some technical device. In the relational model, typically information about the devices would be stored in a certain relation, but separately from information about the possible components which would be stored in another relation. The link between these two types of information is only available *implicitly*: it would typically be another relation containing pairs of identifiers of components and devices. The details of this implicit link must be known to anyone wanting to solve the given task. In contrast, in a conceptual data model this link could be *explicitly* represented. This is much more correlated to the way humans tend to think about such information, and simpler query languages can be formulated. Obviously the mapping to a given logical and ultimately a physical data model becomes more complex in turn.

- Reduced semantic overloading. Where logical models cannot express a semantic abstraction explicitly, they have to use implicit means. It may easily happen that the same implicit means are used to express different semantic abstractions. For example, the part-of relationship between devices and components is modelled in a relational model in the same way as other types of relationships like association or inheritance (see section 3.1.3). The aim of conceptual models is to represent such abstractions in a structural manner.

- Provision of several levels of detail. Since conceptual models use a set of explicit abstraction mechanisms, one may browse through such a model viewing only the most important structural types for a global overview, then include more details for a finer search.

The aim of this chapter is to identify a suitable conceptual meta model for data in Knowledge Discovery. As was mentioned earlier, data-related activities are central to the KDD process, and the design and combination of data transformations to prepare the data for learning consumes the bulk of the time spent on a KDD project. These activities can benefit greatly from the above advantages of conceptual models. Referring to the two levels of description from section 2.2, both logical and physical data models are seen as being located at the technical level in the context of this work.

3.1.2. Structure of the given data

This subsection takes a look at the data as it is given, before the KDD process starts. As mentioned earlier, data to be analysed using KDD has usually been collected for purposes very different from analysis. A useful distinction can be made between *structured* and *unstructured* data. Unstructured data forms include text documents, images, or video files. While such data items may have an inner structure, the structure is not explicitly represented and therefore unavailable for analysis. Structured data, in contrast, consists of small atomic pieces of information like strings or numbers and some structured way of organising them. *Semi-structured* data is inbetween, it includes, for example, text documents whose parts (title, introduction, chapter, etc.) are marked by tags but contain unstructured text.

At the heart of the KDD process is a mining algorithm; almost all mining algorithms deal exclusively with structured data. Unstructured data can be brought into a structured format by special-purpose preprocessing operations, though this is far from trivial. The often-used terms "Web mining" or "Text mining" indicate research areas concerned with such tasks (rather than with mining unstructured data directly, as the names suggest). The most wide-spread structured data format is the so-called *attribute-value* format, or *tabular* data. This is simply a table with columns storing particular data items. The format is also called *propositional*, as explained in sections 1.1. One example of structured data that is not in attribute-value format is graph data, often stored as adjacency matrices or lists, and indeed non-classical mining algorithms are sometimes used on such data (Washio & Motoda, 2003). An example is frequent subgraph discovery, which is discussed in section 6.5.4.

So far fixed data collections have been considered. In the *stream mining* scenario, continuously arriving data is considered; see e.g. (Babcock et al., 2002) or (Domingos & Hulten, 2000). The present work does not deal with the particular challenges of (real-time) stream mining, but presumes fixed data collections as input to a KDD project.

This input data for a KDD process is in the vast majority of cases given in a relational database or in tabular flat files. Databases with other logical models (for example hierarchical or object-oriented) still play a peripheral role. The literature on successful KDD applications clearly reflects this (Kitts et al., 2005; Soares et al., 2005). One reason is certainly that almost all mining algorithms require their input data in an attribute-value format, while hardly any mining algorithms that directly exploit hierarchical or object-oriented data structures have been developed yet. The attribute-value format also easily allows to include data from external, additional sources, like web pages. For example, currency change rates, provided by a web service, may be useful in certain preparation steps. The present work therefore considers only relational or tabular data, and its results do not apply to different logical data models.

The relational data model

Because relational databases provide the data for KDD in almost all applications, a brief description of the relational data model follows. More details can be found, for example, in (Biskup, 1995) or (Ullman, 1988).

The relational model was originally developed by Codd (1970). Its elementary con-

structs are *attributes* and *relations*. Every attribute has exactly one *domain* whose values it can take. A relation is defined as a finite subset of the Cartesian product of the domains of a sequence of attributes. The time-invariant structure of relations is described by relation schemas. A database schema is a set of relation schemas; the set of possible instances of this schema can be restricted by intra- and interrelational integrity constraints.

More formally, let A be a set of attributes and D be a set of domains. Each $d_i \in D$ is a domain, i.e. a set of values. Typical domains would be *integer*, *float* or *string* with corresponding values. Let $dom : A \to D$ be a function that denotes the domain for each attribute. A *relation schema* is a tuple $R = (X, \Sigma)$ with $X \subseteq A$, where Σ is a set of intrarelational integrity constraints which can be modelled by boolean functions that use only attributes from X. A *database schema* is a tuple $DS = (R_1, \ldots, R_n, \Delta)$ such that each R_i $(1 \leqslant i \leqslant n)$ is a relation schema and Δ is a set of interrelational integrity constraints, again boolean functions.

Let $X \subseteq A$. A *tuple* over X is a function $t : X \to dom(X)$ (where $dom(X) := dom(A_1) \times \ldots \times dom(A_m)$ if $X = \{A_1, \ldots, A_m\}$). The value(s) that the tuple takes for an attribute A_j (or an attribute set $V \subset X$) is (are) given by $t[A_j]$ (or $t[V]$, respectively). For a given relation schema $R = (X, \Sigma)$, a *relational instance* r is defined as a finite set of tuples over X that fulfil the conditions in Σ. For a given database schema $DS = (R_1, \ldots, R_n, \Delta)$, a database instance can be given by a set of relational instances $r_1, \ldots, r_n$ to the relation schemas $R_1, \ldots, R_n$ such that all these instances fulfil the interrelational integrity constraints Δ.

This definition assumes that for every tuple and every attribute a value from that attribute's domain can be given. In practice, the tuple may represent an object for which the corresponding feature is unknown. Usually, the domains are extended by a special symbol to represent this situation, but the exact interpretation of this symbol may vary (compare the discussion of empty and missing values in section 2.1.3).

As for intra- and interrelational integrity constraints, only those types of constraints are given here that are needed for this work. Let $S = (X, \Sigma)$ be a relation schema and s an instance to it; let also $V, W \subseteq X$. A *functional dependency* $V \to W$ holds if and only if $\forall u, v \in s : u[V] = v[V] \Rightarrow u[W] = v[W]$. An attribute set $K \subseteq X$ is called a *key* of S if and only if X is functionally dependent on K $(K \to X)$ but not functionally dependent on any strict subset of K. To require functional dependencies or keys in a relation is an intrarelational integrity constraint. A relation can have zero, one or more than one keys; in practice one of the keys should be designated as *primary key* if there are several.

Let $R = (X, \Sigma_1)$ and $S = (Y, \Sigma_2)$ be two relation schemas. Let $V \subseteq X$ with $V = \{A_1, \ldots, A_n\}$ and $W \subseteq Y$ with $W = \{B_1, \ldots, B_n\}$, so that $|V| = |W|$. Let r be an instance to R and s be an instance to S. An *inclusion dependency* between V and W, written $R[V] \subseteq S[W]$, holds if and only if

$$\forall t \in r : \exists u \in s : \forall i = 1, \ldots, n : t[A_i] = u[B_i]$$

As an example, consider the attribute set $A = \{Name, Salary, Department\}$, the domain set $D = \{string, integer\}$ and the function $dom : A \to D$ such that $dom(Name) = string$, $dom(Salary) = integer$ and $dom(Department) = string$. Figure 3.1 displays two relation schemas $R = (X, \Sigma_1)$ and $S = (Y, \Sigma_2)$ with $X = \{Name, Salary\}$ and $Y = \{Name, Department\}$, and with instances

Relation R:

Name	Salary
Jones	50000
Marks	55000
Smith	50000
Davis	60000

Relation S:

Name	Department
Smith	Marketing
Jones	Marketing
Marks	Management

Figure 3.1.: Two relation schemas and instances.

- $r = \{(Smith, 50000), (Jones, 50000), (Marks, 55000), (Davis, 60000)\}$
- $s = \{(Smith, Marketing), (Jones, Marketing), (Marks, Management)\}$.

The set $\{Name\}$ is a key of R and also a key of S. No other subset of A is key anywhere. The inclusion dependency $S[\{Name\}] \subseteq R[\{Name\}]$ holds (but $R[\{Name\}] \subseteq S[\{Name\}]$ does not hold). Corresponding boolean functions form the sets Σ_1 and Σ_2.

Research on the design of relational databases has identified a number of techniques that help to achieve a non-redundant design, which is more or less invulnerable to logical inconsistencies. Certain *normal forms* have been identified for this purpose. The first normal form requires to use only atomic domains for each attribute, rather than structured domains, and to use a primary key for each relation. The second normal form additionally requires that no non-key attribute is functionally dependent on a strict subset of a set of attributes that form a key. Bringing a relation into second normal form can require to split it into several relations, each of which corresponds to a subset of the key attributes on which other attributes are dependent. The third normal form requires, in addition to the conditions for the second normal form, that all non-key attributes are mutually independent, or in other words, only dependent on the key. Again, bringing a relation into third normal form can involve splitting it. Further normal forms exist, but are not needed here.

Since these design techniques are rather well-known, in most (but not all) KDD applications the input data is stored in a relational database, and is given in third normal form. This form eliminates much redundancy that might otherwise be present in the data. In many KDD applications, unfortunately it is necessary to re-introduce some redundancy, as explained in section 1.1.

Set semantics and bag semantics

In the relational data model, sets of tuples fill a relation. As figure 3.1 demonstrates, data stored in this model can easily be represented as tables. For this work the general term *tabular* data, or attribute-value data, is used to include structured data that can be represented as tables with named columns whose values are from a particular domain. However, there is a difference between the relational model and general tabular data: the former excludes the possibility of having two identical tuples in a relation, since sets of tuples form the instances; while the latter may contain duplicate rows, for example if produced in spreadsheets. For example, if the tuple $(Smith, 50000)$ was inserted again into relation R in figure 3.1, it should not appear twice in a tabular representation,

if the relational model is followed strictly. Yet in practice, existing relational database management systems generally allow the insertion of several identical tuples into one relation, and to disallow this requires specific actions by a user. The term *bag semantics* is sometimes used to denote a situation where duplicates are allowed, while *set semantics* denotes the opposite, the exclusion of duplicate rows/tuples (Garcia-Molina et al., 2002). Formally, using bag semantics can be modelled by populating an instance with multisets (sometimes also with sequences) of tuples, rather than sets of them (a multiset is a map from a set to the natural numbers, each element of the set being mapped to the number of times it occurs in the multiset).

One salient difference between the two interpretations can be seen when considering the relational operation called *projection*. In the relational data model, let r be an instance of the relation schema $R = (X, \Sigma)$. The restriction of a tuple $t \in r$ onto an attribute set $V \subseteq X$ is denoted by $t[V]$. The projection of r onto an attribute set $V \subseteq X$ is called $\pi_V(r)$ and is defined as $\pi_V(r) := \{t[V] | t \in r\}$. Thus it may hold that $|\pi_V(r)| < |r|$. In contrast, under bag semantics, selecting a subset of attributes from a relation will not reduce the number of tuples/rows. For this reason the term "projection" should not be used under bag semantics; instead, "attribute selection" is used in this work.

For KDD, bag semantics should be used because possible data sources include tabular data from spreadsheets etc. In practice, this is a minor issue since duplicate rows are rather undesirable, and if data from some source contains duplicates, then early on in the preparation process the duplicates are usually either removed or an artificial key is created. Yet nothing forces KDD users to do so, thus set semantics should not be a requirement for data models in KDD.

A richer, formal model for tabular data has been developed by Gyssens et al. (1996); it is essentially matrix-based, and thus distinguishes between different orders of rows of a table, which is not desired for this work as discussed now.

Ordered and unordered data

In the relational model instances are sets, thus they bear no internal order. Under bag semantics, multisets are used which also do not involve an order. But technically, data sets must be stored in some order, obviously. This section briefly discusses why one may abstract from a technical (implementation-dependent) order, to multisets at the logical or conceptual level.

For KDD purposes, the order of data elements (rows in a table, tuples in a sequence) is not important. The reason is that in principle, mining algorithms are insensitive towards the order of their input examples during training or testing, although some order may be preferred for technical reasons. This is true even for (time) series analysis, sequence discovery, incremental learning approaches, or for learning with concept drift, because the *choice* of a subset of examples is the same regardless of the given order in the superset. For example, in incremental learning (which is often also used to handle concept drift, e.g. (Klinkenberg, 2004)), a model is trained on some data set and then updated using additional data. This additional data is identified according to some criterion (often time-based), but no particular order is required to identify it, nor is any particular order needed within the additional set. In time or value series analysis, signal-to-symbol transformations can be done as long as the (time) index is given, independent of the order

of reading the signals. In fact, windowing (see section A.3.4) is an often-used technique whose purpose is to *encode* the order of tuples in such a way that it becomes exploitable by mining algorithms, since these algorithms make no assumptions on the order of reading tuples. Even when mining natural language text, often a word-order independent representation (the "bag-of-words" model (Salton & Buckley, 1988)) is used, and where the context of a word is considered, like in Named Entity Recognition (NER), techniques similar to windowing (which correspond to choosing a fixed-size context) are applied. Pure text is unstructured data, which this work does not consider; methods to extract structure from text may well be order-dependent. Whenever the technical realisation of a mining algorithm for structured data is order-dependent, this is seen as undesirable and its effects are minimised, like in the BIRCH clustering algorithm (Zhang et al., 1996).

Similarly, most preparation operators presented in the following chapter are independent of the order of tuples in their input; for any order, the same multiset of tuples is produced for the output. The only exception is SAMPLING (section A.1.3). Implementations of sampling techniques are usually order-dependent. However, at the conceptual level, only the fact that the data is somehow sampled is of interest. Thus this operator can also hide its technical implementation from users. Further, the operator ATTRIBUTE DERIVATION (section A.5.4) is a special case, in that it directly accesses the technical level and can thus deliver output that depends on the order of the input data.

Thus orderedness of tuples is a technical-level notion, and some *implementations* of mining algorithms or preparation operators may indeed depend on ordered input for technical or efficiency reasons. But conceptually, two data tables that differ only in the stored order of rows are equal, for KDD purposes. See also (Abiteboul & Vianu, 1991) (discussed again in section 4.1). This equality should be reflected in the conceptual data model to be used for KDD. If some sorting of data is needed to fulfil the technical requirements of a mining algorithm implementation, this can be done automatically at the hidden technical level.

As was said above, this work considers only such data for KDD that is given in a relational data base or in flat files that organise their data in tables. A generalisation of these two forms is the relational data model with bag semantics, in which the order of tuples within a relation is dependent on implementational issues. Thus the technical description level (see section 2.2) of the given data has been identified. The following sections are concerned with the conceptual level.

3.1.3. Semantic abstractions

Conceptual data models are a means to organise a part of the real world in a structural schema which correlates to some extent with the way humans tend to conceive that part. In order to classify and compare conceptual meta models, some general concepts of abstraction have been proposed in the literature early on (Abrial, 1974; Smith & Smith, 1977; Nijssen, 1977; Brodie, 1984; Hull & King, 1987; Storey, 1993). Such abstractions are sometimes called "epistemological primitives" in knowledge representation (Brachman, 1979). In the present work they are called *semantic abstractions*. Specific conceptual models differ in the set of abstractions they support, and how they support them. The most comprehensive list of possible meta constructs (semantic abstractions) is given in (Hull & King, 1987), where a General Semantic Model (GSM) is introduced.

The GSM is designed for tutorial purposes and encompasses a wide range of concrete conceptual models; in fact this makes it so general that it would often offer rather too many possibilities to model a concrete situation. Therefore the list below does not contain the concrete constructs of the GSM but the general abstraction mechanisms behind them (see the literature cited above). Some references to the GSM are made for orientation.

Entity An *entity* represents any thing that exists. It may be a concrete, physical thing like a person or car or an abstract notion like a legal corporation. In object-oriented models, entities are called *objects*. Entities are the instances of a conceptual data schema.

Classification Groups of similar entities can be viewed as belonging to the same *class*. For example, the class `Person` may be used to collect all entities that represent persons. In the Entity-Relationship model, classes are called *entity types*. In the present work also the term *concept* is used, because it is used in the MiningMart system (chapter 6) for historical reasons. In the GSM, classes are called *abstract atomic types* (in contrast to printable atomic types, which are low-level data types like string or integer). A class is described by attributes.

Attribute *Attributes* describe properties of classes. All entities in a class have the property that an attribute denotes. For example, if the class `Person` has an attribute `Name` it means that all entities representing persons can have a name. *Domains* can be used to restrict the possible values of attributes. (The GSM uses a more general notion of attribute because it uses attributes to model relationships.)

Relationship *Relationships* model any meaningful connection between entities of two or more classes. For example, a class `Person` could be connected to a class `Car` by a relationship that models ownership. The relationship instances then specify which persons own which cars. This is an example of a *binary* relationship. Binary relationships can usually be read in two directions: a person owns a car while the car is owned by the person. Some conceptual models explicitly model both directions. The *arity* of a relationship is the number of classes connected; binary relationships have arity 2. In general, conceptual models fall into either of two classes (Hull & King, 1987): those that explicitly model relationships, and those that use attributes pointing from a class to its related class instead. Relationships can be used for very different semantic interpretations (Storey, 1993), so that the exact interpretation of relationships in a given meta model should be prescribed in order to simplify the models.

Cardinality Relationships differ in the numbers of instances they may connect. For example, a person can own zero, one or several cars, but a car is (at least officially) owned by only one person. Conceptual models often allow to restrict a semantic schema so as to express such *cardinalities* explicitly. Combinations of atleast/atmost and zero/one/many are the most common cardinalities.

Role Classes that participate in a relationship play a certain role in that relationship. For example, a person can be said to play the `owner` role in the relationship representing

ownership. This concept is useful to distinguish different relationships a class takes part in, from the view of the class. Some meta models such as KL-ONE (Brachman & Schmolze, 1985) use only (binary) roles in the place of relationships.

Aggregation This abstraction allows to view a relationship between classes as a class in its own right (Brodie, 1984). When considering the aggregate, specific details of its components are suppressed. For example, the type `Lecture` might be an aggregate of the classes `Lecturer`, `Student`, `ScheduledTime` and `RoomNumber`. In this case, the instances of the four-ary relationship connecting instances of the four classes are seen as composite entities. It is a matter of some subjectivity whether such entities should be modelled as classes or relationships; aggregation gives the flexibility to allow both (Biskup, 1995).

Grouping *Grouping* is an abstraction which allows to view a particular set of entities as a different type of entity. It is also called *association* but today this term is used heavily in object-oriented modelling and is therefore avoided here. For example, a certain group of persons (instances of the class `Person`) may form a team which is modelled by the grouping `Team`. The difference to classification is that the member entities (persons in the example) are instances of their own class, and there is an extra class (here `Team`) that models the powerset of member entities, i.e. whose instances are sets of member instances.

Generalisation This abstraction is used when entities (instances) of one class (the subclass) are always also entities of a second class (the superclass). For example, each instance of the class `Lecturer` (each lecturer entity) is also an instance of `Person`. The relationship between the two classes is called an *Is-A* relationship (every lecturer *is a* person). The subclass has all attributes of the superclass, and can have its own additional attributes.

Model constraints Apart from supporting only a subset of the above concepts of abstraction, conceptual models may also use further explicit restrictions on how to use or combine their constructs. For example, in the entity-relationship model, attributes can only have domains whose values are printable (i.e. alphanumeric strings), but cannot point to other entities. Another common and useful constraint is to disallow cyclic *Is-A* relationships, or to disallow any class to be subclass of two other classes.

Derived components Some data models come with a language for specifying derivation rules. These rules allow to derive new structures and to fill them with instances (Hull & King, 1987). For example, a derived attribute for the class `Lecturer` would be one that contains the number of lectures this lecturer gives; it could be derived from the cardinality information of the relationship `Lecture` and its instances. Considering conceptual models for data preparation in KDD, the derivation of new structures is done using data transformations. One might informally see the transformation operations given in chapter 4 as derivation constructs of the conceptual data model to be identified below. Therefore no particular derivation mechanisms are given in this chapter.

The above notions of abstraction can be realised to varying degrees in conceptual meta models. For example, the Entity-Relationship model supports relationships *explicitly* while object-oriented models can only support them *implicitly*, either by using classes that represent an aggregation or by using object-valued attributes.

3.2. A conceptual data model for KDD

This section will identify a conceptual data model that is suitable for the purposes of KDD. Section 3.2.1 discusses which of the semantic abstractions from section 3.1.3 are useful for KDD-specific purposes. Section 3.2.2 then summarises a conceptual data model that supports these abstractions.

3.2.1. Semantic abstractions needed in KDD

The usefulness of a data abstraction concept for KDD purposes is judged under the following considerations. The data preparation phase can involve complex combinations of single data transformations as specified in chapter 4 and exemplified in chapter 5. Each of these data transformations produces a new representation of the data. Therefore in a long preparation chain many intermediate representations are produced. Each of these intermediate results can be the starting point of a new (sub-)chain of further processing, perhaps after a revision of the first KDD results (see section 2.1.5); it can be a useful source of information, for example for data understanding; it may allow fruitful analyses, whether or not these are related to the KDD project that produced it; it is a natural interface to other tools. In sum, these intermediate data representations are important artifacts of the KDD process. For a KDD user the problem arises, however, that there can be a rather large number of them. Therefore, the conceptual data model to be used must allow to *structure* the set of intermediate results. This structuring should ideally reflect the way the data representations are related to each other, by reflecting how they were created. This is exactly what will be achieved in this section, giving a much clearer overview of the KDD process and its results to the user than would be possible without this conceptual-level support. A further discussion is given in section 4.6.

Another topic for consideration is the complexity of the conceptual model. Ideally, the model should explicitly support all abstractions that are useful, without forcing circumventions (implicit representations) of them, while it should not offer any constructs that are superfluous for the purpose for which the model is used (Borgida & Mylopoulos, 2004). The overall goal is to make the *usage* of the model as simple as possible. On the other hand, for the intended usage in this work, the model must be operational in the sense that its mapping to the technical level can be clearly specified, and that transformations of the conceptual schema result in well-defined operations at the technical data model. Wielinga et al. (1993) state that often such formal precision impairs the conceptual clarity of knowledge representing models, therefore they argue for the use of both informal and formal models. Without debating the usefulness of informal models, the present work advocates clear semantics, and it will be demonstrated that good conceptual clarity is achieved with the chosen framework.

Using more semantic constructs could make the conceptual model more general, effectively allowing to model the application domain of the KDD process in a manner that is independent of the concrete KDD project, so that this model can be reused in other projects. The development of such models is the aim of research on *ontologies*. The word *ontology* is today used in computer science to denote a description of a shared conceptualisation of an application domain (Gruber, 1993). *Shared* refers to a group of users or machines. Ontologies are built using different formalisms of varying expressivity; for an introduction see (Staab & Studer, 2004). Formalisms for describing ontologies are (modern) conceptual meta models for data. Sometimes an ontology exists for the application domain from which the data is collected, and this can be useful for Knowledge Discovery, in particular for the mining step (see e.g. (Litvak et al., 2005) or (Svatek et al., 2005)), but also for some data preparation tasks as in (Bogorny et al., 2005), or even for designing parts of the KDD process (the "scenario"-based approach of (Brisson et al., 2004)). In fact, it would be very helpful to describe a KDD application in terms of a *given* ontology *throughout* the different phases of a complete KDD process (Euler & Scholz, 2004; Cespivova et al., 2004). This would also help to reuse existing KDD applications on similar data sets (Morik & Scholz, 2004); see also section 6.6.

However, realising this idea is fraught with two difficulties. First, standard ontologies simply do not exist yet for the vast majority of application domains.[1] Therefore their consistent use across KDD projects or across institutions is not possible. Second, not all ontology formalisms are suitable for supporting KDD-oriented data processing (most of the approaches from the literature mentioned in the previous paragraph are rather domain-specific). In particular, formalisms that are designed to allow automated reasoning, such as description logics, tend to render rather complex data models which are inappropriate to structure the many different data views that are created in a typical KDD data preparation process. Thus a trade-off between the expressivity of the conceptual model and its clarity or simplicity has to be found. The present work attempts to find a balance between these goals. It selects only a small number of constructs for the conceptual (ontological) level, with a canonical mapping to the technical level, but proposes some additional elements that allow basic reasoning about some applicability constraints of operators, to be described in section 3.3. While this conceptual meta model may not be able to capture all semantic aspects of an application domain, it does allow to set up a KDD process based on data schemas expressed in it, and it can capture at least basic semantic concepts so that the schemas are reusable. The important issue of reusability is discussed again in sections 4.3 and 6.6. The present work develops a coherent conceptual model for KDD, combining data- and process-oriented views (see section 4.6) in a single framework. Future work may explore other options for the conceptual data model and their implications for the rest of the KDD process (see section 9.2).

Turning now to the semantic abstractions listed in section 3.1.3, all conceptual meta models (and all ontologies) use the concept of classification, and almost all use attributes. (Entities are the instances of semantic schemas and are therefore not usually represented explicitly in conceptual models, whose purpose it is to specify the schema.) These abstractions are also needed for KDD purposes. They allow a simple and direct mapping

[1] The development of public "foundational" ontologies, open to be extended for specific applications, is the subject of ongoing research (Niles & Pease, 2001; Masolo et al., 2003).

to the technical level: classes correspond to tables (though some classes may be declared to just represent an *abstract* superclass of some given classes, without a corresponding table); attributes correspond to columns; and each row of a table represents an entity.

Conceptual meta models differ according to whether the association between a class and its entities is given by intensional descriptions or definitions of the class, or by an extensional approach that simply lists the associated entities. This is not a crisp but a gradual distinction. For example, description logics are a definitorial framework, in which the extensions of certain *atomic* concepts (classes) are listed, but those of *defined* concepts (classes) are derived from such lists. In contrast, the Entity-Relationship model allows no intensional descriptions other than the attributes of a class. For work in KDD, an extensional approach seems to be more natural, because data sets, the extensions, must be analysed as they are given, without any assumptions on properties that can be used for intensional descriptions.

Another reason for using an extensional approach is implied in the idea of using one conceptual model for the representation of several data sets of the same kind. If the conceptual model specifies only the schema of a class, several technical-level data tables with this schema can be represented by this class. As motivated in sections 1.1.1 and 2.2, this can be very useful for KDD applications. Thus another requirement that the data model should fulfil is to allow such one-to-many mappings between the two levels.

As was mentioned in section 3.1.3, conceptual meta models fall into two classes according to the way they represent relationships (Hull & King, 1987). For KDD an *explicit* representation of relationships is required. The reason is that in many applications the data to be analysed are distributed over several tables and have to be integrated. This is comparatively easy to be done if the way the tables are semantically connected is clearly represented in the conceptual model. Of course, the semantic connection between the tables may be hidden or too complex to be directly modelled. Nevertheless, support for relationships will be useful once the connections have been uncovered or created anew, which would invariably be the first subgoal of data preparation in such cases. Therefore conceptual meta models and ontologies that do not explicitly model relationships can be ruled out for the purposes of this work.

As regards the abstraction concept cardinality, it provides useful information, for example for the estimation of data set sizes (see section 3.3.3) after joins. Support for cardinalities is thus desirable.

Roles are somewhat redundant when relationships are given. They are a convenient means of communication but do not serve particular technical purposes, at least not in a KDD process. They are not needed for KDD.

Aggregation might be useful in some KDD projects, but it is not necessary to have it explicitly modelled. Given that relationships are present in the conceptual model, aggregation would allow to add attributes to a relationship so that it can also be seen as a class. However, in such cases it is also possible to model the respective type as a class with relationships to the other involved class. As an example, consider again the type `Lecture` from section 3.1.3. If it has extra attributes (say a maximum number of participating students), the type can be modelled as a class with relationships to `RoomNumber` and so on, rather than an aggregate type. This may be considered inconvenient in certain situations, but it is a consequence of the decision to keep the conceptual model rather

simple. Aggregation is a dispensable concept for KDD purposes, not least because none of the standard data transformations of chapter 4 makes any specific use of aggregations.

A similar argument holds for grouping, which is a special type of relationship: it can be modelled by a relationship with cardinalities *zero or one* for the member class, and *several* for the set class. There is no reason why this specific type of relationship should be explicitly modelled for KDD purposes, while a large number of other semantic interpretations for relationships (Storey, 1993) are not supported. Again no data preparation operation makes use of this concept, so it is not needed for KDD.

However, generalisation is an important semantic concept which should be explicitly supported for KDD. While it is similar to grouping in that it could be modelled by a relationship, its significance for human thinking makes it an important tool. In this work, two particular types of generalisation are considered very useful. They have been identified based on the characteristics of many important preparation operators from chapter 4, which produce an output class that is linked to the input of the same operator by one of these two types of generalisation. Therefore these two types help to achieve the important aim of structuring the many intermediate data sets produced by a KDD process.

The two types are called *separation* and *specialisation*. For this work they are defined as follows:

Separation A class is a separation of another class if and only if it is described by exactly the same set of attributes as the other class, and each of its instances is also an instance of the other class. For example, the class representing persons aged over 50 can be modelled as a separation of the class representing all persons.

Specialisation A class is a specialisation of another class if and only if it is described by a strict superset of the attributes of the other class, and restricting it to the attributes of the other class yields instances of the other class. For example, if adding the attribute `Income` to the class `Person` results in a new class `PersonWithIncome`, then `PersonWithIncome` is a specialisation of `Person`.

Separation and specialisation are thus two subtypes of Is-A relationships. They might be used together with or instead of Is-A relationships, but because the existence of either a separation link or a specialisation link between two classes implies the existence of an Is-A link, the latter is considered redundant in this work.

To sum up the discussion on useful properties of conceptual data models for KDD, mainly the following criteria were identified:

- the meta model must allow to give a clear structure to the many intermediate artifacts of the KDD process;
- it should not be too complex, yet have clear semantics and allow a precise mapping to the technical level;
- it must explicitly support relationships of arbitrary arity;
- it should allow a one-to-many mapping from classes to technical-level tables; and

- apart from classes, attributes and relationships, it must be able to express cardinalities and must support separation and specialisation.

Entity-relationship (ER) models (Chen, 1976; Teorey et al., 1986; Thalheim, 2000) have been successfully used in database design for a long time. They use classes, attributes and relationships. In most ER models, relationships of any arity are allowed. A number of tables with the same schema can be modelled by one class and its attributes, making ER models the natural choice for representing several like-shaped data tables by one concept. The two types of generalisation that are needed in this work, separation and specialisation, have already been used in ER models, albeit with slightly different semantics than here. The semantics needed here can easily be accommodated by an ER model. Based on this, different intermediate data tables in the KDD process can be represented by different concepts (entity types), and the way they have been created from other data tables can be indicated by relationships as well as separation and specialisation links.

On the one hand, KDD experts must understand the data they analyse very well; on the other hand, they must keep an overview of long processes of data transformations, as well as their intermediate results. ER models provide a level of abstraction that is well suited for this purpose. The schema of the transformation inputs and outputs, and the way they are linked semantically, are therefore represented at the conceptual level using an ER model in this work. The data instances are not explicitly represented, but can be easily accessed, in a supporting software, from an entity type that represents them. A few additional elements that are useful for data modelling in a KDD context are added in section 3.3, but the conceptual data model as such is specified in the following section 3.2.2.

3.2.2. Summary of conceptual data model

This section gives a formal description of the conceptual model as proposed for this work, for reference purposes. Also, how to create such a model from a relational database schema is specified. Thus this chapter has identified the two levels of description from section 2.2 for the data, as well as the connection between them that is needed in order to hide the technical level from the user. Chapter 4 will do the same for the KDD process elements.

The ER model

The model comprises the following elements. There is a global, finite, ordered sequence of *atomic attributes* $A = (A_1, \ldots, A_k)$, a set of *domains* D and a map $dom : A \to D$ mapping each atomic attribute to exactly one domain.

The following domains are available:

- $Binary := \{a, b\} \cup \{\bot\}$
- $Discrete := \Sigma^* \cup \{\bot\}$, where Σ^* is the Kleene closure of some set of alphanumeric symbols Σ
- $Time := \mathbb{N} \cup \{\bot\}$
- $Continuous := \mathbb{R} \cup \{\bot\}$

The domain $Time$ is useful for representing time-related information, such as dates, clock times, or time indices. The above selection of domains is discussed in section 3.3.1. Thus the set of domains is $D = \{Binary, Discrete, Time, Continuous\}$. The special symbol $\perp$ is an element of all sets in D and denotes the empty value.

The *class* notion (section 3.1.3) is realised by *entity types* in ER frameworks. However to be consistent with chapter 6 where the MiningMart system is described, the term *concept* will be used here. So, a *concept* over A is given by $C = (A_{i_1}, \ldots, A_{i_m})$ with $m \geqslant 1$ and $1 \leqslant i_1 \leqslant \ldots \leqslant i_m \leqslant k$ and $A_{i_1} \in A, \ldots, A_{i_m} \in A$, where C is the name of the concept and $(A_{i_1}, \ldots, A_{i_m})$ is the sequence of m attributes describing this concept. The notation $attrib(C)$ will be used to denote the attribute sequence $(A_{i_1}, \ldots, A_{i_m})$ of a concept C, which is also called the concept *signature*.

An *entity* e of C is an element of the Cartesian product E_C of the domains of all attributes of C: $e \in E_C := dom(A_{i_1}) \times \ldots \times dom(A_{i_m})$. An *instance* I of a concept C is a multiset of such entities: $I : E_C \to (\mathbb{N} \cup \{0\})$. For an entity $e \in E_C$, $I(e)$ denotes the number of times e occurs in I. To allow the one-to-many mapping from concepts to instances, a concept can have several instances; it can also have no instances, a situation that may arise during the development of a KDD process, before it is executed (compare section 6.4).

A *relationship type* R is given by $R = (C_1, \ldots, C_m, c_1, \ldots, c_m)$ where $m \geqslant 2$, R is the name of the relationship type, C_1 through C_m are concepts, and c_1 through c_m are cardinalities. Cardinalities are one of $\{one, zeroOrOne, zeroOrMore, oneOrMore\}$. Let $I_1, \ldots, I_m$ be instances of the concepts in R. Since the instances are multisets, obtain sets from them by applying the operator set, where $set(M) := \{x \mid M(x) > 0\}$ for a multiset M. A *relationship* r *of relationship type* R is then an element of the Cartesian product $set(I_1) \times \ldots \times set(I_m)$. A set S of relationships is an instance of the relationship type $R = (C_1, \ldots, C_m, c_1, \ldots, c_m)$ if each element of S is a relationship of relationship type R that is based on the same concept instances, and S obeys the cardinalities of R as specified in the following. (Note that for instances of relationship types, set semantics are sufficient while bag semantics are needed for concepts.) Let $1 \leqslant i \leqslant m$ and let S_i be the projection of S to all concepts except the i-th concept: that is, S_i contains a tuple for all combinations of entities $e_1, \ldots, e_{i-1}, e_{i+1}, \ldots, e_m$ (of concepts $C_1, \ldots, C_{i-1}, C_{i+1}, \ldots, C_m$, respectively) that occur in S. If $c_i = one$ then for every tuple in S_i exactly one matching tuple must exist in S. If $c_i = zeroOrOne$ then S may contain atmost one matching tuple for every tuple in S_i. If $c_i = zeroOrMore$ then any number of matching tuples can be in S for every tuple in S_i, and if $c_i = oneOrMore$ then atleast one matching tuple must be in S for every tuple in S_i. Like concepts, relationship types can have zero, one or more instances.

Certain set notations are used for sequences in the following, as defined now. Given two sequences V and W, $V \subseteq W$ holds if and only if every element of V also occurs in W. If then W has at least one additional element that is not in V, $V \subset W$ holds. For an element v, $v \in V$ means that v occurs in the sequence V and $v \notin V$ means it does not occur in V. The union of two sequences V and W, denoted by $V \cup W$, is found by appending W to V and then removing double elements. The intersection of V and W, $V \cap W$, is the sequence all of whose elements occur both in V and in W. The difference of two sequences V and W, denoted by $V - W$, is obtained by removing all elements

that occur in W from V.

The following definition will be useful below for defining specialisation. For an entity $e \in E_C$ of a concept C and a nonempty attribute sequence $X \subseteq attr(C)$, $e\lceil X$ denotes the *restriction* of e to the attributes in X. Thus $(e\lceil X) \in E_X := \times_{B \in X} dom(B)$. Similarly, for an instance I of the concept C, let $I\lceil X$ with $X \subseteq attrib(C)$ denote the multiset that is obtained from I as follows: $I(e) = n \Rightarrow (I\lceil X)(e\lceil X) = n$. Thus $I\lceil X$ can be the instance of a concept C' with $attrib(C') = X$.

Separations are given by a partial order $\leqslant_{sep}$ on the set of concepts with instances. Given instances I_1 and I_2 of concepts C_1 and C_2, $C_1 \leqslant_{sep} C_2$ holds if and only if $attr(C_1) = attr(C_2)$ and for every entity $e \in E_{C_1}$ with $I_1(e) > 0$, $I_2(e) > 0$ holds.

Specialisations are given by a relation $<_{sp}$ over the set of concepts with instances. Given instances I_1 and I_2 of concepts C_1 and C_2, $C_1 <_{sp} C_2$ holds if and only if $attr(C_2) \subset attr(C_1)$ and for every entity $e \in E_{C_1}$ with $I_1(e) > 0$, $I_2(e\lceil attr(C_2)) > 0$ holds. The relation $<_{sp}$ is not a partial order because it is not reflexive. Note that if $C_1 <_{sp} C_2$ holds, the instance of C_1 may have *more* entities than the instance of C_2, because the restriction of the entities of C_1 to the attributes of C_2 may "map" different entities of C_1 to the same entity in the instance of C_2. The instance of C_1 may also have *fewer* entities than the instance of C_2, because C_2 may have several occurrences of an entity whose correspondent occurs only once in C_1.

If a specialisation or separation holds between two concepts and their instances, a relationship type with suitable cardinalities can also be set up between them. However, a relationship type provides less semantic information than a separation or specialisation, so only the most specific type of link between concepts is always considered in this work.

Mapping from relational data model to ER model

As discussed in use case 1 in chapter 1, one of the purposes of using the ER meta model is to represent the initial data before preparation is started. This initial data is expected to be stored in a relational database in the vast majority of KDD applications. Thus it is briefly discussed in the following how an ER model can be automatically created from a relational database schema. In general, this is a difficult task since the schema may semantically be underspecified. For example, inclusion dependencies or primary keys may not have been declared. Separations and specialisations are certainly not declared. Reverse engineering an ER model from a given relational schema is discussed in depth by Fahrner (1996), for example.

For the present work, it is assumed that the KDD system supports the user in setting up an ER model, by importing as much information as possible from a relational database. Section 6.3.1 takes up this idea. It is also assumed, however, that the user is able to add missing information to the ER model. Since data understanding is an important task in KDD anyway (compare section 2.1.2), adding such information manually can assist both in understanding and documenting the data.

It is rather straightforward to represent each relation by a concept, and each attribute of that relation by an attribute of that concept. However, some attributes of a relation may be used only to refer to the primary keys of other relations, by way of an inclusion dependency. If the dependency is declared in the database schema, a many-to-one-relationship can be created for it, which links the two concepts involved. If the attributes

refer to another relation, but not to a key of that relation, a many-to-many-relationship should be created. Finally, if *all* attributes of a relation refer to other relations by inclusion dependencies, the relation can be considered a *cross table*. In this case it is not represented by a concept, but by a many-to-many-relationship linking all the concepts representing the relations referred to.

The creation of relationships in the ER model thus can only be done automatically if the inclusion dependencies are declared in the relational database schema. If not, the user of the KDD environment may have to add such information manually.

The names of the attributes and concepts are taken from the relational schema. It is important that the user can edit these names afterwards, but only at the conceptual level.

The KDD roles of attributes (see below, section 3.3.2) are not represented in the schema, but have to be added by the KDD user, except for the *Key* role which can be recognised from the inclusion dependencies and primary keys, if they are declared in the schema. Similarly, separation and specialisation links are not declared in the schema, but may be added by the user.

Every conceptual attribute must also be associated to one of the four conceptual domains, or data types, called *Binary*, *Discrete*, *Time* and *Continuous* (see also section 3.3.1). In a relational database schema, typically some technical data types like *string* or *number* are used. For technical data types that represent sequences of alphanumeric characters, typically called *string*, *varchar* etc., the *Discrete* type is used. For technical types representing date or time information, *Time* is the most suitable conceptual type. Attributes arising from numeric technical types should be declared *Continuous*. Some of these assignments of conceptual types may be wrong; for example, a numeric technical type may be used, via some encoding, for discrete values. Thus the conceptual types that have been found by this procedure must also be editable by the user.

If the relational database has an instance, i.e. it is filled with some data, characteristics of this data may also be used heuristically. For example, if only two values occur in some relational attribute, then the *Binary* type may be given to its conceptual counterpart. Again, the results of such heuristics must be editable by the user.

The implementation-dependent order of tuples in the given data does not influence the ER model created in this way.

After this initial model is set up, elements representing the results of data transformations are added to it when the KDD process is developed. The general approach is to transfer as much of the semantic information (like data types and roles) as possible to such results of transformations. This is possible based on the specifications of the operators that produce these results, as will become clear in chapter 4.

3.3. Additional KDD-specific information

Developing a KDD process is a complex endeavour involving much interaction with the data to be analysed. The conceptual data model developed in the previous section can be used to describe the data schema at a conceptual level. Another important factor are of course the contents of the data and the particular role that parts of it play in the KDD project. These issues are explained in this section. Section 3.3.1 discusses data types;

KDD roles are introduced in section 3.3.2; in 3.3.3, statistical information about data contents is considered. Such additional information on data sets must be administrated by a KDD-supporting software, as will be discussed.

3.3.1. Data types

This subsection focuses on attribute domains. A domain is a set of values that an attribute can take. Domains can be categorised along different dimensions. Pyle (1999) distinguishes three dimensions: measurement scale, discrete/continuous, and scalar/nonscalar. The scale of measurement refers to the way the values of a domain are organised; there are five scales: (i) nominal (for naming individual items without an inherent order), (ii) categorical (for naming groups of items without an inherent order), (iii) ordinal (for naming items with an inherent order), (iv) interval (for integer numbers), and (v) ratio (for real numbers). The first three are discrete, in that there is a finite set of values, the other two are continuous (conceptually, there are infinitely many values on these scales). Discrete domains can be further divided into constant domains, with only one value, binary domains, with two different values, and sets, with more values. Scalar attributes bear a single value while nonscalar ones, like vectors of numbers, combine several values.

Another important type of attribute domains serves to store time-related information. Time indices, clock times, or calendar dates can be represented in different ways, but the essential information they give is about the *time-related order* of data items.

Finally, some attribute domains exhibit an inner structure. A common example is hierarchical organisation of values, such as in product information: individual product items, for example green, red and yellow pepper, belong to product groups such as pepper and yet to larger groups like vegetables and then food. Special data mining approaches can directly exploit such hierarchical attributes (Srikant & Agrawal, 1995; Han & Fu, 1999; Domingues & Rezende, 2005). Kohavi et al. (2004) suggest a way of "flattening out" hierarchical structures into binary attributes, using an operation similar to DICHOTOMISATION (see section A.3.1). They also report experience according to which this method is recommendable. Knobbe (2004) transforms such hierarchies into a relationship to an additional concept, following the same aim of flattening the structure[2]. A second example is the cyclical nature of certain time attributes, such as the day of the week or the month of the year; here it is important to derive such attributes if they are not present from the outset, thus to ensure that this cyclical information is available for mining (Kohavi et al., 2004); see also the template TimeSeriesAnalysis in section 6.5.3. To be aware of such inner structures is of course important throughout the KDD process. Yet these structures are not explicitly modelled in this work, as they can usefully be mapped to flat attribute domains.

The different dimensions to describe domains are each useful, but using all of them together would be confusing rather than helpful for the conceptual overview of a KDD application. A software that supports KDD processes should allow to describe the data in a clear but flexible way. Thus, a simple but useful conceptualisation of data domains should be used which does not restrict the data preparation, but keeps it as clear as

[2]Such flattening operations can easily be specified as convenience operators in the framework of chapter 4.

possible. In this work, a choice of conceptual data types is suggested that is directed by the requirements of learning algorithms as listed in table 2.1 on page 17, and by the requirements of the data preparation tasks listed in chapter 4. Using the definitions from above, the proposed types are:

- *time*
- *discrete*
- *binary*
- *continuous*

It will turn out in chapter 4 that these data types allow to describe all data preparation operations at the conceptual level: that is, they are specific enough to enable the formulation of constraints which ensure the technical applicability of a preparation operator to its input. Further, together with information about data characteristics, discussed below, they allow to ensure the usability of a prepared data set by a mining algorithm. This choice of data types explains the fixed set of domains D in the ER model, as summarised in section 3.2.2.

At the technical level, data type restrictions are usually supported in databases, but not in flat files. The common data types here are numbers (integer or real), strings, and calendar dates/clock times. While all conceptual data types can be represented by these technical ones, respecting the technical data types during all data preparation operations is important, even if no database is used, because usage of a database may be introduced at later stages of the project. Writing a data set to a database always requires type correctness at the technical level. However, the technical data types can be hidden from the conceptual level; as will be seen in chapter 4, the technical data type of any output of a data preparation operation can easily be determined.

A conceptual domain type can often be realised by several technical data types. For example, a discrete domain can be realised by strings or by numbers; calendar dates can be represented by strings; and so on. Real-world data frequently exhibits such atypical forms of data type usage. Thus for a KDD process a "messy" use of technical data types must not pose a problem. Rather, the technical level should be hidden and a "cleaned" conceptual view should be provided, as elsewhere in this work. To hide the technical level, a flexible mapping is needed.

The distinction between the two levels is also used – for data types – by Romei et al. (2006), where the two levels are called *physical* and *logical*, respectively. However, they appear to mix attribute *roles*, introduced below, with conceptual or logical data types.

3.3.2. Attribute roles

As was explained in section 2.1.4, labelled data sets are needed to tackle predictive mining problems. In a labelled data set, the label is contained in one or more attributes (usually one). When a mining algorithm is trained on the data, the label attribute(s) must be specified; when the resulting model is evaluated, its predictions are compared to the actual label using the test set. During deployment no labels are available. Thus the label attribute(s) play a special *role* in the KDD process. Most of the other attributes are

used for prediction. In descriptive mining settings, there is no label attribute (with the exception of subgroup discovery approaches, e.g. (Klosgen, 2000; Scholz, 2005)).

Sometimes not all attributes that are important during data preparation are actually needed for mining. For example, *keys* are often necessary to integrate data tables and to identify entities, but they are useless for mining because each learning example corresponds to one entity, which has a unique value in the key attribute, so that no pattern can be based on this attribute. Yet key attributes are important during data preparation, to establish links between tables.

In sum, any attribute plays one of four conceptual roles in the preparation and mining phases:

- *Label*
- *Predictor*
- *Key*
- *No role*

These roles are introduced as another special tag attached to attributes in the ER model. They distinguish how attributes are used in the KDD process, so they are conceptual-level elements. Even the special role *no role* is useful because it may be desired to "switch off" attributes temporarily to see if mining is more successful without them. It would not be convenient to introduce an attribute deletion operator every time this is tried. In this way, the *no role* construct allows to work with the same set of attributes for training, testing and deployment.

3.3.3. Data characteristics (metadata)

Setting up a data preparation process requires not only schema-related information (which is given by the conceptual model) but also information on data contents. This is explained shortly, but will also become apparent in chapter 4, which describes some essential processing operations, and in chapter 8 where software support for these operations is analysed. Both kinds of information, schema- and content-related, are usually referred to as *metadata* (data about data). Section 2.1.2 has listed some metadata that should be collected during the data understanding phase. This section deals with the content-related metadata (data characteristics) that can be employed during the modelling of a KDD process. There are mainly three reasons why this kind of metadata is useful.

The first reason is that this information helps to ensure the usability of the prepared data set for mining; as table 2.1 on page 17 shows, the applicability of mining algorithms can depend on certain characteristics of the data itself (rather than only its data type as discussed above, in section 3.3.1). So when a user attempts to apply a mining algorithm to a data set that violates some of the algorithm's input constraints on data characteristics, the KDD environment can prevent this if the characteristics are known.

The second reason is that the data characteristics provide useful information about intermediate results, and thus give some orientation to the user as to further development

of the preparation process. Further, there are certain preparation operators, to be presented in chapter 4, whose instantiation in a KDD process depends on (is parameterised by) characteristics of the input data set. For example, the operator VALUE MAPPING (section A.5.3) maps the values of an input attribute to new values, thus these input values are a parameter of the operator. When the KDD environment provides these values, the operator instantiation can be simplified.

The third reason is that knowing the data characteristics allows to estimate the storage capacity required for the data sets that are created during preparation. The number of attributes times the number of entities of a concept and its instance already gives a basic estimate of the storage requirements. Knowing storage requirements is important because on the one hand, storing *all* data sets created during a preparation process consumes too much storage capacity in large applications (compare chapter 5), but on the other hand *some* intermediate data sets have to be stored to allow the efficient execution of the preparation process. This issue is discussed in more detail in section 7.3.

These three specific reasons for providing data characteristics are all motivated by an important aim of this work, which is to specify how a data preparation process for KDD can be developed *declaratively* without executing it. Separating the development of a KDD process from its execution is useful because the execution on large data sets takes a lot of time. Many currently available KDD environments (see chapter 8) force their users to interrupt the development repeatedly in order to execute the part that has been developed so far, since otherwise the further development is made impossible by the environment because it does not know the data characteristics it needs to allow the instantiation of certain operators. This situation can be compared to a programming environment that forces a programmer to test their program whenever a few lines of code have been added. Some existing data preparation systems, like Clio (Yan et al., 2001) or Potter's Wheel (Raman & Hellerstein, 2001), to be discussed in more detail in section 4.1.1, execute each single data transformation step immediately, and thus also suffer from inconvenient interruptions of the development process. These systems do not use a conceptual data model. In contrast, KDD systems like MiningMart (chapter 6) allow to set up a preparation process completely independently of its execution, by mechanisms which are based on the conceptual data model, and which are explained below in this section and in chapter 6.

In sum, certain data characteristics should be maintained by a KDD environment, and should even be available to the user since they describe the data as transformed up to a current point in the development. Computation and maintenance of such characteristics is known from database management systems (DBMS), where they are often also called "the statistics" (Haas et al., 2005). The statistics are used for several purposes in the DBMS, including query optimisation. They pertain to values of an attribute. Mannino et al. (1988) distinguish between four types of statistics or data characteristics: (i) descriptors of central tendency, such as mean or median (of the values of an attribute); (ii) descriptors of dispersion, such as minimum/maximum, variance or standard deviation; (iii) descriptors of size, like the count of tuples (entities) or the number of distinct values (of an attribute); and (iv) descriptors of frequency distribution, which include counts of the occurrence of each value, or counts of the occurrence of values within certain intervals (for continuous attributes).

Since information about intermediate data sets must be made available to the user, all descriptors from above could be useful for a KDD environment. But such descriptors that can be used to ensure the applicability of a mining algorithm, or can be used for the correct instantiation of an operator, are of particular importance. The following data characteristics (coming from the last three descriptor groups) have been chosen for the present work:

- the number of rows in every table (the size of the corresponding concept's instance);
- the minimum and maximum values of each attribute with ordered values;
- the list of values each discrete attribute takes;
- a list of equidistant intervals into which the values of a continuous attribute fall, for every continuous attribute;
- the number of occurrences of each value of each discrete attribute;
- the number of values that fall into each of the equidistant intervals, for each continuous attribute;
- the number of missing values in an attribute.

The characteristics of the initial, given data sets can be computed from them, though on large data sets this may take a lot of time. The characteristics of the intermediate data sets that result from some preparation operations, however, can only be computed exactly after these data sets have been created, which is only after execution of the process. Both execution and characteristics computation would consume a lot of time. Fortunately, the framework of this work allows a different method of arriving at intermediate data characteristics. Data preparation is done by *operators* which are specified in chapter 4. The specification includes how the data is transformed, but also often allows some statements about how the characteristics of the data are changed. Some of these changes cannot be given exactly for the output, but have to be *estimated*. Such estimates describe the post-conditions of an operator, i.e. the characteristics of its output. The list above also reflects which kinds of characteristics can be estimated comparatively easily, given the operator specifications of chapter 4. The usefulness of the estimates is a main reason why a KDD environment should maintain the data characteristics. Appendix A gives detailed estimates for each processing operator, while some general guidelines are given in the remainder of this subsection. Section 7.1.3 describes an implementation.

Characteristics (or metadata) of the initial data set (the input for the first processing operator(s)) are both required and useful even if it takes much time to compute them, though the computation can be done on a sample of the data, and some or all characteristics might be provided by hand from someone who knows the data sets from previous work. From then on, as much *inference* as possible should be performed to gain metadata about later data sets (results of intermediate processing operations). Inference here means to evaluate the post-conditions of operators, to arrive at statements about the characteristics of some particular output of an operator.

Concerning estimation, one can distinguish between *optimistic* and *pessimistic* estimation of metadata. For example, when a complex formula is used for the selection criterion in an instantiation of the operator ROW SELECTION (see section A.1.2), it is difficult or impossible to infer which values will occur in the output attributes without evaluating the formula on the data, i.e. executing the operator. Pessimistic metadata estimation does not deliver any values of the output attributes in such cases. However, in this example it is clear that no values are added to the output attributes that have not been in the input. So the list of values in the output can be optimistically assumed to be unchanged.

Pessimistic metadata administration makes the declarative set-up of a KDD process model more tedious, as often intermediate steps will have to be executed in order to analyse the data. Optimistic administration eases the development of the process, but when the process is executed later, conflicts may occur between estimated and actual metadata. The operators specified in chapter 4 must therefore be realised technically such that they are robust against such conflicts. That is, replacing the estimated with the actual (computed) metadata must not lead to problems. For instance, if the operator VALUE MAPPING (section A.5.3) is applied to an actually non-occurring value because this value was assumed to be in the data during the specification of the operator, it simply does not map the value. Some data characteristics, such as the number of entities, and the value distributions, are needed for size estimation only, anyway; misestimations of data set sizes affect the storage strategy, but not the syntactic or semantic validity of the developed process. Therefore optimistic administration of the value lists and data types is suggested in this work, and chapter 4 details for every processing operator how this can be achieved.

It should be noted that inferring and estimating characteristics will not give accurate results over long chains of preparation steps. Most steps lose some of their input characteristics information, so that the output information about characteristics is less detailed. However, any piece of information about data characteristics of a concept helps the user to make decisions, and the system to check the integrity of the process. Compared to current KDD environments, which do not support metadata inference at all (see chapter 8), providing optimistic metadata administration as presented in this work is a big progress.

Methods for estimating data characteristics have been presented in the database literature, but are restricted to estimating the output *size* of data sets (number of tuples) after application of relational operators. The reason is that the size is the major indicator for the cost of processing the data set, and an estimate of this cost is needed during query optimisation, which is the task of finding an effective way of executing a declarative query. In contrast, estimating the other data characteristics above after an operator application has not been addressed by database researchers. Such estimates become possible by the detailed specifications of the preparation operators provided in chapter 4.

Size estimation also plays a role in the present work, as storage issues may depend on it; compare chapter 8 and section 7.3. Research on size estimation has focused on the relational operators selection and join, and indeed these are the two operators for which size estimation is difficult (the other operators, at least in this work, leave the input size unchanged, or the output size can be inferred from the value distribution of certain input attributes). The term *selectivity* estimation is often used in the literature with respect to

these operators; the selectivity is the output size divided by the input size, or in the case of joins: the output size divided by the product of the input sizes, because this product is the largest possible output size of a join operation.

One can distinguish different approaches to selectivity estimation. A simple and currently widely used method is based on histograms (Poosala et al., 1996; Haas et al., 2005), which are tables of the (frequently occurring) values of an attribute together with their frequencies; for continuous attributes, value *ranges* are used. Many different methods of building a histogram, in particular of finding the interval boundaries for continuous values, are surveyed by Poosala et al. (1996). The histograms provide (often approximate) information about the distribution of the values of an attribute, and thus allow more precise estimates than some naive approaches based on a uniform distribution assumption. As indicated in the list of metadata above, this work also proposes the use of histograms, though they may be complemented by other methods for selectivity estimation. For simple selection operations based on equality or simple comparison to constants, and for discrete attributes, the output size can be determined accurately based on histograms. For example, when selecting all persons under the age of 18 from a concept that includes an AGE attribute, all frequencies of values up to 18 must be added from the histogram. If the attribute is considered continuous and the histogram uses value ranges, for example containing only the total frequency of age values between 15 and 20, a simple linear interpolation can be used to estimate the fraction of values within this range that are smaller than 18. Boolean combinations of such simple selections can sometimes also be evaluated accurately. However, when a comparison between attributes or a combination across attributes is involved, the combined distributions of attribute values are needed, which are usually not available. Estimates are usually based on the assumption that the attribute values are distributed independently in such cases (Mannino et al., 1988), because measuring the correlation of the values of different attributes is too expensive.

Another method of selectivity estimation uses the assumption that the data distribution follows some parameterised function, like a uniform, Poisson or Zipf distribution (Christodoulakis, 1983), or a polynomial (Sun et al., 1993). Then the parameters of the function are estimated from the data. This approach cannot be used in the present work because the data to which the parameters are to be tuned is not available before executing the KDD process.

An important approach to size estimation is based on *sampling* the data, and executing the operator in question on it in order to get estimates of the selectivity. There is a lot of research on sampling for this purpose; see (Haas et al., 1996; Acharya et al., 1999; Ngu et al., 2004) for overviews and current approaches. In the context of the present work, the data is often not available for metadata estimation, therefore sampling approaches cannot be used either (except for the few "first" operators that are applied to the initial data sets). Another approach that cannot be used here is based on past experience about queries and their output sizes; regression or other machine learning techniques are then applied to learn the prediction of output sizes (Chen & Roussopoulos, 1994; Harangsri et al., 1997).

For estimating join selectivity, Acharya et al. (1999) have presented a method that is tailored to the special case of joins based on foreign key links, which correspond to relationships in the present work. Many data warehouses are organised in star or snowflake

schemas, which use such links exclusively; since data for KDD also frequently resides in such warehouses, the method will often be applicable in KDD processes. Compare for example the model application described in chapter 5. The simple basic idea for selectivity estimation is that the result of a join of two concepts linked by a one-to-many relationship will contain exactly as many entities as given in the concept on the "many-side" of the relationship. Similarly, the result of a join of two concepts linked by a many-to-many relationship will contain exactly as many entities as given in the database cross table that stores the relationship keys. This assumes that the (foreign) keys that establish the relationship are used for joining. Unfortunately, in a data preparation process, (exact) information about relationships between data sets is lost when data transformations are applied to the data sets. Although the operators in chapter 4 attempt to preserve as much semantic information about the data sets as possible, the relationship links between processed data sets usually cannot be recovered. However, they can be declared to exist by the user, or created by a special operator, even for the transformed data sets, and thus can be made available for applying joins, supporting the estimation of selectivity.

From the above it is clear that only some of the simpler methods that have been developed for estimating selectivity can be applied in this work. Such estimates are used for KDD for the first time in the present work. Section 7.1.3 describes which methods were implemented; simple methods were implemented first, but more sophisticated ones can be integrated into the framework at any time.

3.4. Summary

The structure of the data as it is given for analysis has been examined in section 3.1. The relational data model (with bag semantics) has been identified as a suitable model for this technical level. For the conceptual level, a number of abstraction constructs have been presented in section 3.1.3, and a choice of constructs that are useful for the purposes of this work has been made in section 3.2.1. The main criteria have been the ability to structure the many intermediate results of the preparation process, and the simplicity of the model. Based on these criteria, an entity-relationship model has been suggested as the conceptual data model. In section 3.3, additional KDD-specific elements for the conceptual data model have been discussed. In particular, conceptual data types, attribute roles, and (estimated) data characteristics have been included in the conceptual model, since they provide useful information for the control of the preparation process. The following chapter examines this process in more detail.

4. A Conceptual Process Model for KDD

The previous chapter has developed a conceptual-level description framework of the data to be analysed in a KDD project, including its mapping to the technical level. The present chapter introduces *data transformations* that are specified in terms of these data descriptions, i.e. in terms of ER models as given in section 3.2.2. The outputs of these transformations are again elements of the ER model. The general idea is that an initial conceptual data model can be constructed by the user, assisted by the system, to represent the "raw" data sets before any processing has been applied. Then the transformations are applied to process the data. Each transformation is an element of the conceptual process model. The latter also shows how the transformations are linked (a link between two transformations is given if the output of the first is the input of the second). Further, each transformation adds a concept that represents its output to the conceptual data model. It also adds a relationship type, a separation or a specialisation link connecting the new concept to one or more of the previously given concepts. The new concept often "inherits" many semantic elements of the concept the transformation was applied to (the input concept), for example the roles and conceptual data types of attributes that do not take part in the processing. Thus the transformations attempt to keep as much semantic information from their input concepts as possible when creating an output concept. In this way the KDD process produces a growing web of elements in the conceptual data model which are linked in ways that indicate how they were created from each other. This web provides a different view on the preparation process, as an alternative to the process model itself; this is discussed in section 4.6.

The main part of this chapter is thus given by section 4.2 and the list in appendix A, which introduce the data preparation operators for KDD; before that, related work is given in section 4.1. The computational power of these preparation operators is examined in section 4.3. Section 4.4 then briefly introduces an abstraction mechanism used for combinations of preparation operators. Section 4.5 takes a short look at other phases of the KDD process, discussing how the conceptual-level approach to KDD extends to them. Finally, section 4.6 discusses two dual ways of developing a KDD process model, one based on the data model and one on the process model.

4.1. Related work

The basic idea that is taken to the conceptual level in this chapter, of defining data transformations in terms of *operators* that perform pre-programmed tasks on certain inputs, and yield certain outputs, shows up in numerous works both from KDD and from research on databases. In the database world, data transformations have mainly been examined in the context of federated databases and schema evolution. Section 4.1.1 discusses these approaches. Section 4.1.2 then turns to research on operators for knowledge

discovery.

4.1.1. Federated databases and schema evolution

Data integration

Today many institutions have more than one database. In various applications, of which data mining is but one example, they are faced with the challenge of providing a single interface to their distributed sources. Work on *data integration* addresses this challenge. Quite a number of data integration systems have been described in the literature; for overviews, see (Lenzerini, 2002; Halevy, 2001) and the systems listed there. The dominant architectural model for data integration is the federated database, in which various sources are mapped to a common *mediated* or *global* schema. The mediated schema uses a Common Data Model (CDM) that must be able to accommodate all data models used in the source databases. Users directly query the mediated schema without worrying about how the data needed to answer the query is distributed to the various sources. For each source database, a mapping or translation to the mediated schema must be found in order to be able to answer such global queries. Such translations realise data transformations. Note that the mediated schema is constructed manually for a data integration application (Halevy, 2001), and afterwards the mappings are constructed, also manually or semi-automatically (Doan et al., 2001). Even when both schemas (local and mediated) are given, finding mappings between them automatically is very difficult (Fiedler et al., 2005); see also section 7.1.4. This contrasts with the more exploratory scenario of constructing data transformations to arrive at various new representations, where the target schema is not defined in advanced, as in KDD processes. The latter scenario is taken up again further below, but first some approaches for data transformations are discussed that take both the source and the target data schema as given.

One may distinguish between data integration approaches that use a relational CDM and others with a more complex common model. A well-known example for the latter group is TSIMMIS (Garcia-Molina et al., 1997), a system that uses the specifically-developed Object Exchange Model (OEM) (Papakonstantinou et al., 1995) as CDM. This is an object-oriented data model. Other examples (Calvanese et al., 2000; Franconi & Ng, 2000) use description logics. More complex data models allow, and require, to use more complex mappings. Indeed, mappings between different *ontologies* are examined in a closely related research area – see Kalfoglou and Schorlemmer (2003) for an excellent survey. Such mappings allow to realise a variety of tasks that go beyond data transformation, but are the subject of ongoing research; see (Melnik et al., 2005) for an example. Below the focus is on aspects of data integration systems that involve actual data transformations.

Many of the systems which require complex translations of source data are based on the mediator paradigm (Wiederhold, 1992) (TSIMMIS is one example). The data transformations are done by *wrappers* in such systems. A wrapper encapsulates the source data and is able to answer queries on it that are formulated in the global query language. Wrappers have to be created manually for each data source, by programming them. While some research exists that attempts to simplify the creation of such wrappers (Hammer et al., 1997), it remains a non-trivial task involving the specification of formal expressions.

For example, Altenschmidt and Biskup (2002) present TYML, a formal language for expressing mappings between schemas, which is used in their data integration system called MMM. TYML allows to use OQL (Object Query Language) expressions (Cattell et al., 2000) for mapping a number of source attributes to a target attribute. TYML expressions must be developed by the integration administrator.

Davidson and Kosky (1997) describe another approach to data transformations based on given source and target schemas, using a rule-based formalism to describe the transformations. Their rules apply to an object-oriented data model, and are expressed as Horn logic rules; that is, they consist of a body and a head, with the body stating properties of the source data (schema) and the head stating how elements of the target data (schema) are built from the elements described by the body.

A remarkable contrast to these somewhat technical approaches to data transformations is set by Yan et al. (2001), at least as far as the user's view is concerned. These authors' system (called Clio) provides an interactive interface by which the user constructs a mapping from source to target elements step-by-step, without specifying the mapping explicitly, but instead by relating data examples from source and target to each other. The system attempts to always show the most illustrative, or distinctive, examples to the user when ambiguities arise.

Relational extensions

In the following, approaches based on the simpler relational data model are discussed. Using the relational model, or extensions of it, for all schemas involved in a data integration application means that the mappings from sources to mediated schema can be simpler: often they consist only of a one-to-one or many-to-one correspondence between elements of the two schemas. Such correspondences can sometimes be discovered automatically, if there are enough syntactic clues in the two schemas; this is called *schema matching*, see (Rahm & Bernstein, 2001) and section 7.1.4. The data in the schemas can also provide clues, an idea which has been exploited in a schema matching approach that involves machine learning (Doan et al., 2001). However, in a KDD setting the "target" or mediated schema, the one with the prepared data, is not available in the beginning but must be constructed, so schema matching approaches do not help.

Yet even with simple correspondences between schema elements, at first the task of creating the mediated schema has to be solved, and it has to be done manually. One distinguishes the two approaches of describing the mediated schema in terms of views over the sources ("global-as-view"), and of describing the sources as views over the mediated schema ("local-as-view"). Thus the correspondences between the schemas are given in the view definitions. The TSIMMIS system mentioned above, like many others, follows the global-as-view approach. Using local-as-view, the translation of queries on the mediated schema to the sources can be seen as a query rewriting problem (Duschka et al., 2000; Halevy, 2001). In contrast to query rewriting for query optimisation, for data integration the goal is to rewrite a query such that it uses only the source relations, and returns all tuples that the particular sources provide and that fulfil the query conditions. More details can be found in the survey by Halevy (2001), who also discusses three algorithms for query rewriting in the data integration context.

To draw an analogy to KDD (considering the data preparation phase), one might want

to see the schema of the initial, given data as a source schema, and the target schema of prepared data, which is used directly for mining, as a mediated or global schema. The discussion so far indicates that no methods exist to automatically find transformations between the two schemas: the corresponding task is always solved manually in data integration systems, either by programming wrappers or by finding view definitions that express one of the schemas in terms of the other. So it is currently not possible to have a user simply specify the desired target schema, and to discover the necessary transformation from the sources automatically. Rather, the user will have to specify how to transform the data in order to arrive at the desired target representation. To support this at the conceptual level is the task that will be solved in this chapter. Fortunately, there is some research that is involved with data transformations at the technical level, to be discussed now.

Without specifically tailored formalisms, data transformations can be done using standard SQL, standard programming languages, stored procedures of the database management system used, or so-called ETL[1] tools (Carreira & Galhardas, 2004). The disadvantages of employing technical-level elements (mainly costly development and bad maintainability, see section 2.2) apply here to SQL, programming languages, and stored procedures. On the other hand, ETL tools (which usually offer a graphical interface to data transformation, with many of the conceptual-level advantages) do not provide enough functionality to create arbitrary transformations. For example, the computation of new attributes is often restricted. Hence several researchers have proposed frameworks, discussed in the following, in which data transformations are easy to express and realise. In particular, one objective was to use declarative languages for data transformations, in view of the success of the declarative query language SQL, and the independence of implementation techniques it offers. Several proposed extensions to SQL are discussed below. But first one different approached is mentioned.

Potter's Wheel (Raman & Hellerstein, 2000; Raman & Hellerstein, 2001) is a system that offers a graphical way (using menus) of applying data transformations to tabular data. The input and output of a transformation are immediately visualised in spreadsheets (thus without an abstract data model). The system provides many useful operators, including an operator that is similar to Attribute derivation (an operator presented in section A.5.4) in that it adds an attribute to the input. In contrast to Attribute derivation, the new value of each entity may only depend on one particular old value of the same entity. All operators available in Potter's Wheel can be specified as convenience operators in the framework of this chapter (see section 4.2). The authors of the system also analyse the computational power of their operators, establishing that any mapping from one entity in the input to several entities in the output can be realised using their operators. This result also holds for the operators of the present chapter, see section 4.3.

Carreira and Galhardas (2004) have suggested an extension of the relational algebra by a new, very general "data mapper" operator for computing new attributes and new tuples for a relation. The operator Attribute derivation introduced in section A.5.4 is a specialised version of the data mapper: it produces a new attribute but no new tuples. Introducing new tuples allows to add data to a data set which does not represent

[1] Data extraction, transformation and loading

real world entities or phenomena, and is therefore not useful for the data analysis purposes of this work. The application example motivating the introduction of new tuples in Carreira and Galhardas' work can also be handled by the operators presented in this chapter. Carreira and Galhardas' report does not examine the computational power of their operators, unlike the present work (section 4.3). But it contains algebraic rules that involve their operator, to be used in the optimisation of query execution, of which many apply also to ATTRIBUTE DERIVATION.

Another relational extension is proposed by Sattler and Schallehn (2001). They observe that approaches like the above rely on programmed scripts, provided by users, to realise their transformations. Thus they introduce new SQL constructs for a few types of data transformations, to avoid the need for programming such transformations. Their constructs mainly allow to pivotise relations (see section A.3.2), or to sample from them. For some other data preparation tasks like cleaning or specialised aggregations, the authors also rely on programmed extensions of their framework, providing Java interfaces that allow to insert user-defined functions into their language.

Schema evolution and schema independence

An important aspect for data integration and similar applications is *schema evolution*, which refers to any changes to the schemas of the source databases. Schema evolution is common in operational databases, as demands for data to be stored change with the real-world phenomena that produce the data (Roddick et al., 2000). Any changes to the schema of a source database have to be reflected in the transformations based on it. At the same time, to perform a schema evolution in the first place is nothing else than constructing a mapping, or transformation, from the old to the new schema. Thus the frameworks for schema evolution are rather similar to the data transformation frameworks discussed in this section. For example, Claypool et al. (1998) use "schema evolution primitives" for an object-oriented data model. These primitives are taken from (Banerjee et al., 1987) and apply to their object-oriented data model; they consist of simple atomic changes like adding an attribute (compare the operator ATTRIBUTE DERIVATION, section A.5.4), changing the name or domain of an attribute, changing the superclass of a class, and others. Claypool et al. (1998) combine the primitives to "templates" that can perform more complex tasks.

The necessity to adapt existing data transformations, or mappings in the data integration applications, to evolved schemas has led to the idea of designing data transformation languages that are robust against schema changes. This can be achieved by designing languages that allow to query and manipulate both data and schema elements, and in particular, to translate data to schema elements and vice versa. An example for a data transformation that involves such a translation is given in section A.3.2, and illustrated in figure A.1 on page 205.

For a well-known example, Lakshmanan et al. (1996; 2001) have introduced SchemaSQL, a language that is downward compatible with SQL, but introduces variables that can not only range over relations (like SQL's tuple variables), but also over relation names, attribute names, and values of a column. Thus the language treats data and metadata alike. Among other things, SchemaSQL allows to restructure a data schema, to use "horizontal" aggregation functions, or the creation of views whose structure changes when

the structure of the input data (the input schema) changes. SchemaSQL has an expressive power that is independent of the schema by which a data set is organised (*schema independence*). As an example, consider figure A.1 on page 205: in SQL a query asking for all values of the attribute `Week` is possible, given the relation on the left, but not the one on the right; in SchemaSQL, attribute names can be queried and thus the query is possible on both relations.

A more algebraic view on data and metadata transformations is taken in (Wyss & Robertson, 2005b). These authors propose an extended relational algebra called Federated Interoperable Relational Algebra (FIRA). It is schema independent, like SchemaSQL. The naming stresses the possible application of an implementation of such an algebra in federated databases, for data integration purposes. Wyss and Robertson introduce a notion of "transformational completeness" which is explained below.

A brief discussion of the FIRA operators follows, because some of them are similar to the operators introduced in this chapter (the latter have been proposed independently in (Euler, 2005c)), and because section 4.3 refers to them. The discussion is kept informal.

Besides the operators of the standard relational algebra, FIRA contains six further operators. *Drop projection* is a modified projection operator whose parameters do not contain the attributes to be projected, but the ones to be dropped (left out of the resulting projection). This allows to express certain queries without exact knowledge of the attributes in the input or result. The *Down* operator allows to "pull down" relation names or attribute names into the data; that is, these names become values of new attributes. This is an operator that changes the status of metadata to data. *Attribute dereference* is an operator used to interpret values of tuples as attribute names, so it can refer to attributes whose names are listed as data values. The dereference operator accesses the values of the so-referenced attribute(s). Thus this operator partly reads data as metadata. *Generalised union* is an operator that unifies all relations within a given database (which is a set of relations), using an outer join. The result contains all the information from the input relations in one single relation. *Partitioning* splits a relation into several relations according to the values of a specified attribute, such that one output relation corresponds to each distinct value of that attribute. The operator SEGMENTATION (section A.6.1) from this chapter provides the same functionality. Finally, the *transpose* operator changes data to metadata: each distinct value of a specified attribute is transformed into a new attribute, whose values contain copies of the values of another specified input attribute. This operator corresponds to PIVOTISATION (section A.3.2) without aggregation.

The idea of designing schema independent languages has also been used for non-relational data models. For one example, Su et al. (2000) have proposed MetaOQL as an extension of the standard query language for object-oriented data, OQL.

Transformational completeness

Wyss and Robertson (2005b) do not justify their particular choice of operators for FIRA, except that they introduce a rather informal notion of transformational completeness, which basically involves standard relational completeness (for example through the availability of the standard relational algebra operators), plus the presence of operators that can change the status of metadata to data and vice versa. The authors propose FIRA

as a "formal archetype" of what it means to be transformationally complete, similar to the way that standard relational algebra is a formal archetype of what it means to be relationally complete. Section 4.3 will show that the operators presented in this chapter provide transformational completeness in this sense.

A more powerful notion of transformational completeness is to require from a list of operators that it can be used to transform any data schema, together with data, into a new data schema, if the transformation is computable at all. This degree of completeness is achieved by the tabular algebra introduced in Gyssens et al. (1996), which is based on the tabular data model. The data model essentially models spreadsheet-like tables, or matrices. The tabular algebra involves two special "tagging" operators and a looping construct; they are necessary to achieve the indicated computational power. But they also introduce a complexity which makes this algebra unsuitable for the present work, whose purpose is to ease data transformations for end users. An interesting open question is how precisely the non-looping part of the tabular algebra and FIRA are connected (Wyss & Robertson, 2005b).

Summary

Research on data integration and schema evolution has shown that data transformations are required in many applications, and that non-trivial challenges, such as schema independence, have to be met. The design of a declarative, easy-to-use but powerful data transformation language has been a particular motivation for many researchers. With respect to the two description levels used in the present work, elements of such languages could be seen as conceptual because they are tailored towards the particular purpose of data transformation, replacing specifically programmed constructs from general-purpose languages. However, the proposed mechanisms are still somewhat technical in that they require experience in dealing with formal languages. The aim of this chapter is to free users from handling formal languages for data transformation. The only approaches that also achieve this are (Raman & Hellerstein, 2001) and (Yan et al., 2001), but they do not represent the transformation process; instead they visualise the results of each particular transformation, using no abstract data model, which makes it difficult to keep an overview in the complex preparation processes that are needed for KDD (compare chapter 5).

A common idea in many approaches discussed above (and below) is to implement data transformations as sequences of previously specified operators, with well-defined inputs and outputs to achieve compositionality. This approach is also followed in the present work, as it provides a high degree of flexibility. The operators are represented graphically, and nesting them is represented by forming directed acyclic graphs with the operators as nodes. One of the proposed transformation languages could then be used to realise the operators technically.

An important notion from this area of research is schema independence. Schema independence is a property of a language, not of a particular query. It has not been defined formally by the authors who introduced it (Lakshmanan et al., 2001), but it involves a robustness against changes of the status from metadata to data and back between different representations of (essentially) the same data set, so that a query can be formulated on each representation that returns the same answer. This kind of robustness is provided

by the operators used in this chapter.

For a set of operators, the question of which types of transformations can be realised with them is important. The notion of transformational completeness was developed to handle it. The computational power of the operators presented in this work is briefly examined in section 4.3.

4.1.2. Operators for knowledge discovery

The operator-based approach from data transformations has been transferred by KDD researchers to the whole KDD process. Indeed, the importance of compositionality, as a technique to construct complex analyses from basic building blocks, has only recently been pointed out in a position paper on current challenges in KDD (Ramakrishnan et al., 2005). In this respect the KDD world is clearly inspired by the success of the relational algebra in the database world. However, as the following discussion will reveal, the proposed approaches rely on formal languages, so that the conceptual level as conceived in this work is missing in these approaches.

Note in the following that the discussion is not concerned with methods of data preparation, or the justification for these methods. Such issues can be found in the literature, mainly in (Pyle, 1999), also in (Famili et al., 1997). Instead the focus here is on the operationalisation of preparation methods.

Mining operators

The first attempts in defining operators for KDD were made for the mining phase. Some approaches concentrated on particular mining paradigms, while others tried to incorporate several types of mining algorithms. A particularly active area has focused on frequent itemset or (association) rule mining (Han et al., 1996; Meo et al., 1998; Boulicaut et al., 1999; Imielinski & Virmani, 1999). Similar to some approaches mentioned in section 4.1.1, these authors have proposed SQL extensions, that is, constructs to be used in SQL queries which mine a data set (specified by parts of the query) for rules, and which return such rules (as relations or in other output formats).

Another line of work has identified the SQL operator "group by" as a primitive operator that is useful in efficient implementations of some mining algorithms (Freitas & Lavington, 1996; John & Lent, 1997).

Operators for the whole KDD process

The SQL extensions are taken further by Kramer et al. (2005), whose operators provide not only frequent itemset mining options, but also clustering, k-nearest neighbour prediction, and some of the most common data preparation operators. Interestingly, their language adds the results of mining algorithms as a new attribute to the relation from which they were mined. They see it as a step towards integrating the preparation and mining phases in a data-oriented view. The new attribute contains the predicted class or value when the task was classification or regression, or a cluster identifier when clustering was applied. In frequent itemset mining, a new pattern relation with boolean attributes is created, with one attribute for each item and an entry (row) for each frequent itemset.

But there is also an additional operator that joins the pattern relation to the relation from which the patterns were mined, such that the data relation is extended by boolean attributes indicating for each example whether it is covered by a particular pattern. This approach demonstrates how data and patterns mined from the data can be viewed under a single (data-oriented) framework, both during training and deployment. The operator ATTRIBUTE DERIVATION, introduced in the present work in section A.5.4, exploits this idea to accommodate mining algorithms in the KDD process. It is similar to the `extend` operator used by Kramer et al. (2005) (it was proposed independently in (Euler, 2005c)).

The preparation operators that Kramer et al. (2005) have included in their frameworks are sampling, automatic attribute selection, computation of distances between examples, discretisation and transposition (exchange of rows and columns; refer to appendix A for descriptions of the other preparation operators). Kramer et al.'s language could serve to implement the technical level for the conceptual level elements introduced in this chapter.

The data preparation language by Sattler and Schallehn (2001), which was discussed in section 4.1.1, has got some elements which are useful for KDD, as it includes constructs for data cleaning, sampling, and discretisation, and is extensible by user-defined grouping or aggregation functions.

Clear et al. (1999) have also extended a database query language with specific knowledge discovery constructs. The language is SQL/MX, the query language of an object-relational database management system (DBMS) called NonStop SQL/MX. The authors point out that extending query languages offers the opportunity to implement the extensions at a low (system-near) level within the DBMS, to gain efficiency. They also provide guidelines as to when a language extension should be directly supported by the DBMS; particular issues are generality (applicability for many tasks), and potential for performance improvement. The operators implemented for data preparation in SQL/MX are: transposition, which is here a concise form of computing multiple data aggregations at once; sampling; sequence functions, which provide access to previous tuples from a current tuple when iterating through the tuples; and partitioning, whose functionality is equal to that of SEGMENTATION (section A.6.1).

A special attention to data cleaning was given by Galhardas et al. (2001). They distinguish between a logical level of describing cleaning operations, where SQL together with their proposed extensions is used (in a declarative way), and a physical level that provides implementations of the operations, such that a logical operation (like clustering) can be realised by various physical methods (clustering algorithms). However, even at the SQL level these authors employ (call) a number of specifically programmed external functions. These functions serve particular data cleaning purposes. The application area considered in (Galhardas et al., 2001) is to sort and clean bibliographic references extracted automatically from the web. A number of special functions are used by the authors to describe a data cleaning process even at the logical level. Thus the distinction of the two levels is not very precise in their work.

On the commercial side, Microsoft has included data mining functionality in its SQL Server 2005 software (Tang & MacLennan, 2005). It comes with a query language called DMX. Its focus is on prediction functions; some data preparation tasks can be performed, but they are not always independent modules (discretisation and automatic attribute selection are examples).

Particular preparation operators

Apart from query languages, there are also some research reports on particular preparation operators. One family of operations that has received much attention is the group of aggregation functions. Apart from theoretical studies (e.g. (Cabibbo & Torlone, 1999)), the use of aggregation in data mining applications has been examined. Aggregation is a useful tool for *propositionalisation*, the process of combining information from several data sets into one (Knobbe et al., 2001). Since data sets are often in a one-to-many relationship, adding information from the "many-side" to the "one-side" requires to aggregate tuples[2]. Common aggregation functions are to take the maximum, minimum, count or average of values on the "many-side". Flexible, user-defined aggregation functions have also been proposed, for example in (Schallehn et al., 2001); incidentally, aggregation functions have been shown to be useful in the efficient implementation of mining algorithms (Wang & Zaniolo, 1999).

Propositionalisation is used in order to get a single data table that can be mined, as many mining algorithms deal only with single input tables (compare table 2.1 on page 17). The alternative is to directly mine several data tables using multirelational learning algorithms, see section 2.1.3. However, there are reports showing that propositionalisation does not lead to worse results, and can improve results, in terms of mining quality (Krogel & Wrobel, 2001; Krogel et al., 2003), but it can speed up mining because the propositionalisation has to be done only once, while mining experiments are typically run a number of times. Besides, rather intelligent forms of propositionalisation can be used that expose previously hidden information to the mining algorithm. Such intelligent ways of aggregation have been examined by Perlich and Provost (2003). As is typical for propositionalisation, they suggest to automatically apply a variety of aggregation methods, each of which adds an attribute to the central mining table, and then to leave it to the mining algorithm or a feature selection method to weigh the relevance of each added attribute. They propose aggregation methods that take the frequency distribution of values of an attribute of interest in the related table into account. As an example, consider the mining of data about customers of a company who have bought certain products; there is a concept for customer data and one for products, linked by the relationship type "bought". The attribute of interest from the product concept could be the type of product, so that its frequency distribution (based on the relationship) shows which types of products have been bought how often by any customers. Similarly the frequency distribution of product types bought by particular customers can be found. The aggregation methods then compare the particular distribution of each customer with the general frequency distribution, deriving a sum of the differences as the aggregated value, for example. They may also take the target attribute for mining into account (a classification task is assumed), comparing the distribution of a particular class of customers against the general distribution. A simpler variant of their methods, suggested by the authors, is to compare not the frequency distributions but only the frequency of the most frequent value (the most frequently bought product), for the different single customers or for classes of customers. This simpler variant has been specified as a convenience operator below (section A.2.2), as a representative of this kind of aggregation.

[2]The same operation is called "reverse pivoting" in (Hereth & Stumme, 2001).

The other variants can be specified in a similar way for the present framework.

Another important group of operators is given by pivotisation operators. For a description of pivotisation see section A.3.2. Such an operator changes the status of data to metadata and vice versa, and has thus been included in FIRA – see section 4.1.1. Cunningham et al. (2004) have introduced an additional SQL statement for this operator, and have studied algebraic optimisations that involve this operator. A more formal account is to be found in (Wyss & Robertson, 2005a). Pivotisation and reverse pivotisation are called "fold" and "unfold" in (Raman & Hellerstein, 2001).

Computational power of operators

A question that has received little attention in the KDD literature so far is how to decide on a good choice of preparation operators. Most of the reports discussed above simply propose lists of operators without justifying them. This is also true for (Kietz et al., 2000) and (Gimbel et al., 2004), which are two reports that are not centred on data preparation but mention such lists in passing. In fact, a good choice of operators can be characterised by a trade-off. On the one hand, there is the aim of allowing highly complex data transformations. This leads to the requirement that the set of operators be computationally complete, or Turing-complete. Many KDD tools offer proprietary programming languages to manipulate the data, in order to provide this high degree of flexibility. On the other hand, one important aim of this work is to facilitate the development of KDD processes by abstracting from low-level programming, to a conceptual or task-oriented level. This abstraction entails a simplification, rendering less powerful but more understandable operations.

Many of the above approaches have started from the relational algebra (RA), or SQL. RA is far from being computationally complete (Aho & Ullman, 1979), but includes some important and useful operators. Nevertheless, the above approaches have all extended SQL by specialised operators for various purposes. Thus the relational algebra alone does not seem powerful enough to express the various data transformations that are needed in practice. In particular, as pointed out in section 4.1.1, there is a need to manipulate both data and schema elements, and to change their status from metadata to data and back, which the relational algebra is inadequate for. Section 4.2 explains how the present work arrives at a powerful list of preparation operators for KDD without requiring formal programming from users.

Summary

While many researchers have proposed lists of operators for data preparation, few have arrived at clean extensions of SQL (without mixing in specially programmed functions), few have justified their choice of operators, few have examined the computational power of their operators, and no approaches have taken data preparation operators to a conceptual level by freeing users from dealing with formal languages. In contrast, this work provides a list of operators that can be used, through a supporting system, without formal programming, and that is found by a systematic examination of the major preparation tasks in a data mining context. The following section explains this.

4.2. Data preparation operators

While section 2.1.3 has listed the *reasons* for data preparation and a number of high-level *tasks*, this chapter concentrates on the operationalisation of preparation methods. Appendix A lists many specific operations needed for data preparation for KDD; this section gives an overview, and explains the schema of descriptions used in appendix A. Thus this work provides an ontology of data preparation operations. When expressed in a suitable formalism, such as the one presented in chapter 6, this ontology can support existing approaches to offer KDD methodology over Web or Grid Services (Cannataro & Comito, 2003); see also section 6.1.2.

Usually, data preparation is seen as the execution of basic steps, each of which applies some predefined data transformation to the output of the previous step(s), resulting in *dependency graphs* of data preparation (see also section 4.4). The data transformations are defined through *operators*, which are specified by their input, their transformation task and their output. It is important to note that these specifications are given in this section using the conceptual data model from chapter 3 (section 3.2.2). Previous work on data preparation operators is given in section 4.1.

The approach taken in this work to finding a suitable set of data preparation operators has been as follows. In comparison with other fields where the representation of given data sets must be changed or mapped to other representations, like data integration (see section 4.1.1), there are two particularities of knowledge discovery that must be accounted for. One is that background knowledge may have to be introduced, or information content may have to be exposed more explicitly (section 2.1.3). The second is that the goal, the final representation of the data, is not always known beforehand, nor does it necessarily remain fixed in the course of a knowledge discovery project, due to the exploratory nature of new KDD projects. The first aspect means that ways of adding new data values, computed from the given data, must be available. Apart from a rather general operator which can be used for arbitrary computations of such new values, some operators that provide typical computations are included for convenience (section A.5). The second aspect leads to the requirement that data preparation operations should be simple to deploy and change, so that the human analysts can concentrate on actually mining the data. Recall from section 2.1.3 that the data representation is one decisive factor for being able to find interesting knowledge. Creating suitable data representations is in most cases a matter of intuition that cannot be automated, thus it is an important goal to support this task as far as possible.

For this reason, every operator specified in this work is associated to one of the *high-level preparation tasks* that have been identified in section 2.1.3. These tasks are: data reduction, propositionalisation, changing the organisation of the data, data cleaning, and feature construction. One further task group is added in section A.6: it is used to control the kind of pseudo-parallel processing that was motivated in section 1.1.1. Since the high-level tasks reflect the typical structure of a KDD process (in which data reduction is followed by propositionalisation and creating the right organisation of the data, followed by data cleaning and feature construction), the association of operators to high-level tasks is very useful for guiding less experienced users through the preparation process. Further, for every operator, its relevance to data mining is briefly discussed, by

explaining why and in which kinds of situations the operator might be useful. Some of this latter type of information is based on (Pyle, 1999).

It should be noted, however, that this operator list is not closed, but is open for extension by further operators. The list of operators presented in appendix A includes all data preparation operators that are mentioned in the literature on KDD (see section 6.1.2) and on KDD tools (section 8.1.2), all operators that were needed when implementing the model case (chapter 5), and all operators that any of the tools examined in chapter 8 (section 8.5) provides. It is based on the list given in (Morik et al., 2001), but the specifications here are more detailed, and are adapted to the refined conceptual data model from chapter 3. For instance, they include the semantic links between input and output of the operators. Also, some additional operators, as well as the associations to the high-level preparation tasks, are provided by the author of this work. The only major data transformation from the literature that is not included is *transposition*. This is the transformation that is analogous to exchanging rows and columns in a matrix. Kramer et al. (2005) argue that this operator is needed in some applications. It can easily be included in the list of operators below, but since it plays no role elsewhere in this work, this was omitted. Nevertheless, it should be emphasised that it makes sense to encode a certain functionality from data preparation in a specific operator, if this functionality is frequently needed.

In appendix A, all operators are listed and grouped according to the tasks. The following paragraphs explain the schema of their presentation. For every operator, its input and output in terms of the conceptual data model (section 3.2.2) are given. As noted above, each operator produces a new output concept as well as links (relationship types, specialisations or separations) between this output and its input concepts; as explained in section 3.2.2, only the most specific type of link that the operator adds is given. The new elements (concepts and links) are added to the semantic schema that represents the shape of the data sets available so far in the preparation process.

The *parameters* of the operators specify the kind of information that a user has to give when applying the operator to concrete input. For example, an operator that is used to scale the values of a particular input attribute (SCALING, section A.5.2) has a parameter to specify which concept it should be applied to, one parameter to specify the input attribute, and two numeric parameters that specify the new range of the values. In addition, a name for the newly constructed attribute must be given; a parameter for this exists for every feature construction operator (see section A.5). The name of the output concept is a parameter of all operators, thus this parameter is not listed specifically for each operator. Minor variants of an operator are sometimes given as "special options"; the reason for not introducing separate operators for such variants is that the input and output are the same, and the transformation is very similar.

Further, for every operator, preconditions that specify when it is applicable and postconditions that further specify its output are given. For the preconditions, a distinction is made between *constraints*, which represent schema-level input requirements that must be met, and *conditions*, which represent data-level (instance-level) input requirements. The constraints mainly concern type checks, based on the conceptual data types (which are known for each attribute, see section 3.3.1). The conditions concern data characteristics (section 3.3.3). Obeying the constraints and conditions ensures that an operator is

technically applicable.

For the postconditions, *assertions* are distinguished from *estimates*. Assertions concern the shape of the output, such as names, types and roles of attributes. Estimates concern the data characteristics of the output, as discussed in section 3.3.3. Note that both assertions and estimates give statements about the operator's output that can be made before the operator is actually executed on its input data. Thus these statements can be made as soon as the operator's parameters are specified. Similarly, the constraints (schema-level input requirements) can be checked as soon as the operator's parameters are specified; in contrast, the conditions (data-level requirements) cannot be checked before executing the operator on actual data. While the *estimates* of data characteristics could be used for checking the conditions before execution, they will in general be too imprecise to allow enforcing the conditions: In a longer chunk of operators, usually some of the information about data characteristics that is available about the input of the first operator, will not be available at the output of the chunk due to incomplete estimates (for some operators, some output estimates are generally unknown).

All descriptions of estimates assume that the information about the input data characteristics is complete. In operator applications where this is not the case, some of the described estimates may not be available even after specifying the parameters of the operator.

The conceptual data model suggested in the previous chapter allows to represent several data tables that share the same schema by a single concept. The operators in this chapter must be able to handle this. The descriptions of the operators in appendix A are given for single instances of the concepts, but when applied to a concept with several instances, the operator simply applies to all instances and creates as many instances for the output concept as are given with the input. A problem may arise for those operators that use more than one input concept, if the input concepts have differing numbers of instances; this situation is excluded in the preconditions of these operators.

4.3. Computational power of the operators

This section compares the kinds of transformations that can be done using the operators from this chapter, with the transformations that other data transformation formalisms that have been suggested in the literature are capable of.

Section 4.1.1 has introduced the notions of schema independence and transformational completeness, which are two requirements proposed in the literature that data transformation operators should fulfil. Both concepts have not been precisely defined so far, but Wyss and Robertson (2005b) have proposed the FIRA algebra as a formal archetype of a transformationally complete language, which is also schema independent. They have stressed that such a language must be able to perform transformations between data and metadata; in particular, transformations must be possible in all directions between relation names (for this work, concept names), attribute names, and data items.

The list of operators given in appendix A, which is based on (Morik et al., 2001) and (Euler, 2005c), includes operators that "promote" data items to attribute names, for example Pivotisation. The reverse direction, introducing data items based on attribute names, is possible with Reverse pivotisation. The operator Attribute derivation

(section A.5.4) can be used to introduce data items based on the concept name. However, promoting data items or attribute names to concept names is not done by any operator in this chapter, because the names of the output concepts are always given by the user. In FIRA, the *partitioning* operator introduces new relations that are named based on data or attribute names. This operator is very similar to SEGMENTATION (section A.6.1), but the latter only introduces new data tables at the technical level, in order to hide the complexity introduced by this kind of operation from the user. Thus the present framework keeps a stricter separation between schemas and instances (or metadata and data) than FIRA, in particular from the view of the conceptual level, but allows essentially the same operations as FIRA at the technical level (it is easy to see that the FIRA operators can be realised with the operators of this work, the only exception being the naming of concepts as just discussed). Separating the two description levels thus makes the framework presented here more user-friendly than other approaches.

Among the operators of this work, ATTRIBUTE DERIVATION has a special status: it does not provide standard functionality for KDD applications, but is needed to allow the flexible addition of information for mining (feature construction, see section 2.1.3). Also, it forces the user to work at the technical level, since the ways of adding information that users might need for their application cannot be foreseen to be modelled at the conceptual level. The operator allows to employ a computationally complete programming language to access the data and compute new values for each entity, but it does not allow the introduction of new entities, and it does not allow to access instances of concepts other than the input concept.[3]

Computing new data values is a facility that enhances the computational power of the language defined by the operators, compared to classical query languages, which may transform the data but do not compute new data items (Abiteboul & Vianu, 1991). In recent studies summarised in (Libkin, 2003), Libkin has examined the expressive power of SQL (version 2, without recursion); the inclusion in SQL of aggregate and grouping functions, and arithmetic operations on numerical values, deviates from relational theory and makes SQL more powerful than relational algebra. These devices are also provided by the operators considered here. It is well-known that reachability queries, like the transitive closure of a directed graph, are not expressible in relational algebra and Libkin proves that this is true also for SQL. Among others he considers a function application operator which is somewhat similar to ATTRIBUTE DERIVATION, in that it adds an attribute to a relation, but it applies only to functions on tuples (which correspond to entities here). It corresponds to virtual columns in SQL. ATTRIBUTE DERIVATION is more powerful as it can realise functions on whole concepts (with instances). It is easy to see that this capability makes the list of operators from this chapter strictly more powerful than the relational algebra, or SQL, or FIRA, for example. Indeed, computing the transitive closure can be done by encoding the computation in a function that can be used by ATTRIBUTE DERIVATION; the function would have to be applied to an argument concept whose instance provides all combinations of nodes in the graph, so that ATTRIBUTE

[3]Because a computationally complete language is used, the output of this operator may depend on the order in which the input data happens to be given due to implementational specificities. The only other operator for which this is true is SAMPLING, because its exact output depends on the way random selection is implemented. In any case, at the conceptual level, the order of entities does not play any role for mining.

DERIVATION can mark for each combination whether an edge between them belongs to the transitive closure or not. This argument concept can simply be created by joining the concept that represents the original graph with itself.

The relational algebra essentially corresponds to first-order logic using Horn clauses without recursion, negation or functions. Introducing recursion leads to a well-known query language that is more expressive than the relational algebra, Datalog (Ullman, 1988)[4]. Since Datalog can use recursion, it can be used to compute functions without requiring a bound on their output size. In contrast, there are only two operators in this chapter, the join operator (section A.2.1) and REVERSE PIVOTISATION, that increase the number of entities in the output with respect to the input. Join can produce a number of entities up to the square of the input size, while the second operator produces a number of entities that is bound by the product of the input size and the number of attributes. A constant number of applications of these operators, like in a fixed expression from the language that is formed by these operators, can only produce a number of entities that is polynomial in the input size. This is a major difference to Datalog.

Another extension of first-order logic that was suggested to overcome some limitations of the relational algebra is to introduce a least fixpoint operator (Aho & Ullman, 1979; Chandra & Harel, 1982). The resulting logic is called fixpoint logic. In terms of relations, a least fixpoint of an equation of the form $R = f(R)$ is the smallest relation (with respect to the subset hierarchy) that fulfils the equation. A unique least fixpoint always exists if the function f is monotone, that is $f(R_1) \subseteq f(R_2)$ holds if $R_1 \subseteq R_2$. Many interesting queries can be formulated as least fixpoints of monotone functions. For example, the transitive closure of a directed graph encoded in a binary relation R_0 is the least fixpoint of the equation $R = f(R)$, if f is such that it computes the join of R_0 with R using different attributes as keys, projects the result onto the first and last attribute, and unifies it with R_0 (Aho & Ullman, 1979).

Datalog has been shown to be equivalent to the negation-free existential fragment of fixpoint logic (Chandra & Harel, 1985; Kolaitis & Vardi, 1995). Indeed, queries like the transitive closure of a graph are easy to express in Datalog using recursive Horn clauses. However, non-monotone queries cannot be expressed in Datalog; for example, the complement of the transitive closure of a graph is not expressible (Kolaitis & Vardi, 1995). In contrast, it is easy to see, based on the above computation of the transitive closure by ATTRIBUTE DERIVATION, that the complement of the transitive closure can also be computed by ATTRIBUTE DERIVATION.

In fact, it can be shown that most of the operators listed in appendix A can be replaced by a combination of ATTRIBUTE DERIVATION with a few other operators. The two other operators needed are the join operator, which is needed to combine concepts and in order to create new entities (by self-joins), and ATTRIBUTE SELECTION. Since ATTRIBUTE DERIVATION can be used to create the attributes that form the output of the other operators, these three operators could suffice. One could see these three operators as *primitive* operators; the other operators would be used for convenience. However, the functions needed in ATTRIBUTE DERIVATION to replace a convenience operator by a combination of the three primitives are not trivial. Also, the number of primitives needed

[4]The SQL standard version 3 also includes recursion, but not as part of the core standard, so that only a few DBMS vendors support it.

for replacing a convenience operator is not always constant, but depends on the number of attributes in the output concept.

It follows that by using ATTRIBUTE DERIVATION and the other operators from appendix A, any concept that is computable from some given concepts (with instances) at all, and whose instance size is polynomially bounded in terms of the input sizes, can be created. However, the way to create it may depend on the number of output attributes.

4.4. Data preparation graphs

The remainder of this chapter now turns to a more global perspective on preparation. As was said in section 4.2, a data preparation process consists of a number of *steps*, or operator applications, executed in a particular order defined by the inputs and outputs of the operators. That is, the output of any step can be used as input by another step. This data flow induces a directed acyclic graph (DAG) on the steps (and also on the input and output concepts, see section 4.6).

When modelling this DAG, the user can be supported by having the system allow only connections that do not violate any of the constraints or conditions of operators, as listed above. Since most of the constraints concern the conceptual data type of certain input attributes, this amounts to a basic type checking mechanism. Apart from this type checking, joining two concepts into one is safeguarded, in semantic terms, by requiring a relationship to be declared between the concepts (see the remarks introducing section A.2). The validity of parameters can also be checked. Thus the interplay of the data model with the rather strongly specified operators can provide much more guidance to human users than would be possible at the technical level. Invalid data preparation paths are excluded. At the same time, the necessary freedom for exploring the possibilities of data preparation remains. This freedom is indispensable during the first development of a new KDD application, as explained in section 1.1 under "exploration". It is a characteristic of preparation for mining that this freedom exists. Little guidance about successful paths of preparation can be given to new users, except by pointing to solutions that have been published previously. This is the topic of chapter 6.

For large KDD applications (compare chapter 5), the graph of steps can be rather complex. However, often some parts of the graph form a conceptual unit, in which a specific task is completed using a certain number of steps. In fact, some such subtasks tend to reoccur, given several KDD applications (see sections 6.5.3 and 6.6.2). Continuing the approach of conceptual-level support to these larger units, it is useful to allow the division of the graph into *chunks* of steps, to build conceptual units. These chunks can be hierarchically organised, corresponding to tasks and subtasks that are solved in each chunk. For example, the highest-level chunks could be organised to correspond to the KDD process phases introduced in chapter 2, or to the high-level preparation tasks given in section 2.1.3. This provides a clear overview of the complete process and helps to organise both the development and the maintenance of the KDD application. There is no correspondent at the technical level to these chunks.

From outside, a chunk can be seen as a special kind of operator; its input is the set of concepts that the first step(s) of its inner steps take as input, and its output can be the output of any of its inner steps. Internally, a chunk is again a directed acyclic graph.

Often, chunks will have only one input and one output, as this is a conceptually simple structure and chunks serve conceptual simplification, but this is by no means required. In chapter 5, the use of chunks is demonstrated on a large KDD application, while section 6.6 underlines the conceptual importance of chunks for the re-use of KDD applications.

One might consider the introduction of new kinds of operations at the level of chunks and graphs. Their arguments would not be concepts but chunks. This work provides such operations indeed, they are discussed in sections 6.6 and 7.1.2; they adapt a chunk to a changed model of its input data.

4.5. Other phases of the KDD process

In this section, a brief look is taken at other phases in the KDD process, before and after data preparation, to see how conceptual support can be extended to them.

Like data preparation, both business and data understanding can benefit from the existence of a domain ontology (Cespivova et al., 2004). Given the ER framework suggested in chapter 3, whose aptness for data preparation does not at all make it the first choice in general to build domain ontologies, it may be necessary to map a given domain ontology to an ER model. This process can at best be partially automated; however, doing it by hand is actually advantageous, as it provides the necessary understanding of both the domain and the data that represents it, without which the development of a successful KDD process is hardly possible.

An important part of data understanding is working with a number of visualisation tools. Often, visualisation and data preparation are integrated in a software; it makes sense to use the same conceptual view of the data for both tasks – see also section 4.6. The same is true for the mining and deployment phases. Because data preparation usually consumes the bulk of work dedicated to the development of a KDD task, support for the process should be centred on this phase, and extended to the other phases where possible.

During mining, conceptual-level support is mainly needed for training, testing (evaluation of models), and parameter tuning, as well as the visualisation of models. Conceptual support here means again to present solutions to these tasks in suitable terms; for example, standard operations should be offered to split a data set into training set and test set, to learn, evaluate and apply a model, to automatically find optimal parameter settings, and so on. Since mining is in itself a complex process, in fact this often leads to a separate graph of processing tasks. According to Mierswa et al. (2003), trees of nestable operators are a suitable, conceptual representation for these tasks. The leaves of the trees represent operations such as the learning or application of a model, while the inner nodes correspond to more abstract, control-oriented tasks such as cross validation or meta learning. This representation provides great flexibility for the design of complex mining experiments, which are independent of the data preparation in that they take a single, fixed data table as input.

Concerning deployment, section 2.1.6 has shown that it is closely linked to mining. As discussed in section 4.1.2, many mining algorithms can be seen as special cases of Attribute derivation; the same is true for the deployment of such algorithms to new data. See section 7.2.5 where a realisation of these ideas is discussed technically.

Correspondences between an instance of a mining operator and the instance used for deployment must be clearly indicated. Further, a post-processing step for the predicted label must be available if the original label was reversibly transformed (see sections 2.1.6 for an explanation and 7.2.6 for a technical solution). In descriptive settings, the model itself must be presented to the user in an understandable way. This task, model visualisation, is beyond the scope of the present work.

4.6. Two dual views of the preparation process

Traditionally, the KDD process has been thought of, and represented in software tools, as a graph of operator applications. The graph represents the data flow. This is a useful and intuitive approach. With the framework of the present work an alternative view is possible, one that is centred on the data that is being processed. Table 4.1 shows that every operator listed in the earlier sections produces exactly one of the three types of links between concepts foreseen in the conceptual data model from chapter 3: relationship type, separation or specialisation (recall that always the most specific type of link is produced). It also shows that these links are always *directed*. This leads to the alternative view which displays the KDD process as a web of concepts and links between the concepts; the concepts represent initial and intermediate data sets, while the links reflect how the concepts are related to each other. The graph in which these concepts are nodes and their links are edges is again directed and acyclic.

A duality between the two views can be established. Whenever an operator is added to the process-oriented view, its output concept can be automatically created and added to the data-oriented view together with the corresponding link, which is possible due to the well-defined semantics of operators. Conversely, whenever a new concept and a directed link (either separation, specialisation or relationship type) are created in the concept-oriented view, the system can offer the operators whose specification allows to realise this link; when an operator is chosen and its parameters are specified, it can automatically be added to the process-oriented view. Further, if an operator has n incoming and m outgoing edges in the process view, then in the concept view its output concept is connected to the output concepts of the n preceding operators by n edges, all of which are either incoming or outgoing, and is connected to the output concepts of the m following operators by m edges which are again either all incoming or all outgoing[5]. This means that the graph structures in the two views are very similar. The figures in chapter 5 illustrate this. Therefore a graphical user interface of a KDD system can be imagined which offers to control the KDD process from both views. In addition to the traditional interface, it would provide a *concept editor* that is used both to set up the initial ER model, and to create further concepts with links to the present concepts. The attributes and conceptual data types of the output concept can be determined automatically, just like in the process-oriented view. The concept of chunking (section 4.4) can be applied to both views; a chunk in the concept editor contains all concepts involved in the corresponding chunk in the process view, so that chunkings are easily transferred between the views.

[5]The only exception are the initial concepts that represent the given data, since they are not output concepts of any step.

Operator	Relationship	Separation	Specialisation
Attribute selection			$I <_{sp} O$
Row selection		$O \leqslant_{sep} I$	
Sampling		$O \leqslant_{sep} I$	
Aggregation	$n:1$		
Discretisation			$O <_{sp} I$
Scaling			$O <_{sp} I$
Value mapping			$O <_{sp} I$
Attribute derivation			$O <_{sp} I$
Join by relationship			$O <_{sp} I$
Aggregate by relationship			$O <_{sp} I$
Union		$I \leqslant_{sep} O$	
Missing value replacement			$O <_{sp} I$
Filtering outliers		$O \leqslant_{sep} I$	
Dichotomisation			$O <_{sp} I$
Pivotisation	$n:1$		
Reverse pivotisation	$1:n$		
Windowing	$1:0..1$		
Segmentation		$O \leqslant_{sep} I$	
Unsegmentation		$I \leqslant_{sep} O$	

Table 4.1.: Operators and the type of link between concepts they produce. I = input concept(s), O = output concept(s), $x:y$ = relationship type from input to output concept with given cardinality.

Developing a KDD process based on the data-centred view has the following advantages, compared to the traditional process-oriented view:

- The (intermediate) data sets are important artifacts of the KDD process, as discussed in section 3.2.1. All these artifacts are directly represented in the concept view in a *structured* way. If there is only a process-oriented view, the data sets are hidden; when they are inspected using additional tools, they appear to be unstructured. Only by consulting the process representation can they be related or structured. This involves an inconvenient switch between tools or views.

- In the process-oriented view, important semantic information about intermediate results gets lost easily. For example, consider two concepts A and B related by a relationship type. Now A is used as input to a ROW SELECTION, resulting in a concept C that is a separation of A. C is in fact also linked to B by the relationship type, because A is. By following the links in the concept web this can easily be seen (one might display the relationship type between C and B explicitly, but this would clutter the graphical representation too much). In the process view this information is not available, even if the relationship type between A and B was known and explicitly represented in a different tool (say a database management tool).

- An integration of data visualisation tools and data querying tools into the KDD environment or system becomes much easier; these tools help to understand the data and to discover new options for KDD approaches offered by intermediate results. The concept view can thus become a single interface to all development tasks needed for a KDD project.

- Data sets are the natural interfaces to other tools, like additional implementations of mining algorithms. From the concept view this interface can easily be controlled or shaped. Data sets have already been suggested as the "bridge" between preparation and mining (Ramakrishnan et al., 2005; Kramer et al., 2005); see section 4.1.2. The concept view supports this important role directly.

- When creating a link in the concept view, both input and output concept are immediately fixed. In the process-oriented view, creating a link from a source operator to another operator does not prescribe which concept that was produced in the graph leading to the source operator is to be used as input, so the user may have to choose from a large number of concepts.

In sum, the concept view gives as much structure to the representation of the KDD process as the process view, but offers better integration of the many data-centred tasks needed during a KDD project. This does not leave the process-oriented view superfluous. Both the process editor and the concept editor alone are a sufficient means to develop and execute KDD processes, but together they provide a maximum amount of information and flexibility to the user. Chapter 5 gives examples for both views and illustrates their complementarity and the correspondence of chunks in both views. Chapters 6 and 7 introduce the MiningMart system which is the first system to support both views.

4.7. Summary

The transformation of data plays a role in other application contexts besides KDD, such as data integration. In a KDD context, the necessity to compute new values based on the given data, and the exploratory nature of data preparation, are important issues that must be accounted for. By providing many pre-specified operators (appendix A) that can be combined to complex preparation processes, users can avoid formal programming and can concentrate on their main task, which is the development of a representation that allows successful learning. For the computation of new values, a general operator is available, but several frequently occurring ways of computing such new values are provided by specific "convenience" operators (section A.5).

Parts of a preparation process can be "chunked" together to form own units, with the same kind of input and output as single operator applications (section 4.4). These preparation chunks, which can be organised hierarchically, help to organise large processes, for example by designating solutions to specific subproblems (see section 6.5.3).

Each operator produces a particular type of semantic link between its input and output concepts. In this way, a dual or orthogonal view on the transformation process arises in the conceptual data model (section 4.6). Together these two views provide a high amount of information and flexibility to KDD users.

5. An Illustrating Example: KDD for Telecommunications

In this chapter, an example for a complex KDD process with extensive data preparation is given. This example can also be examined online: see section 6.5. The KDD process illustrates the concepts introduced in the previous chapters, in particular the data preparation operators from chapter 4 on the one hand and the dual data views they produce on the other. Section 5.1 introduces the application domain and gives an overview; the following sections each describe one chunk (compare 4.4) of the data preparation graph. Section 5.8 draws some conclusions and discusses limitations of this model application.

Appendix D briefly explains some extracts of the technical level SQL program that realises this model application. It was automatically created using the MiningMart software described in chapter 6. It can be compared to the screenshots of the graphical representation of the conceptual level given in this chapter, for a demonstration of the advantages of the conceptual-level approach taken in this work.

5.1. Overview

This KDD application was modelled based on two real-world applications (Chudzian et al., 2003; Richeldi & Perrucci, 2002a) (see also (Euler, 2005b; Euler, 2005d)) which were developed in the European project MiningMart (Morik & Scholz, 2004). It has been implemented on several KDD platforms (see chapter 8) using a large set (2 GB) of artificial, random data which was created based on the real data schemas used in the original applications. More precisely, small data samples from the original applications were provided for the project and these samples were multiplied many times, and integrated using newly created keys, to gain the artificial data sets.

The application is from the telecommunications domain. The business goal is the prediction of churn, that is, predicting whether a customer is likely to discontinue the subscription to the telecompany soon, and move to a competitor. In telecommunications, churn behaviour is quite common and involves high costs; a small increase in the accuracy of churn prediction can therefore result in substantial cost savings. The companies try to retain customers likely to churn by using specialised marketing campaigns.

The application uses information about customers who have left the company in the past, to predict churn for customers who are still in contract with the company. Thus the labelled data set that can be used for training and testing is limited by the amount of data available for past customers. For deployment, all current customers can be used. The model application demonstrates both training of models and their deployment; both current and past customer data is prepared in the same way.

In general, churn behaviour is learned and predicted based on monthly information

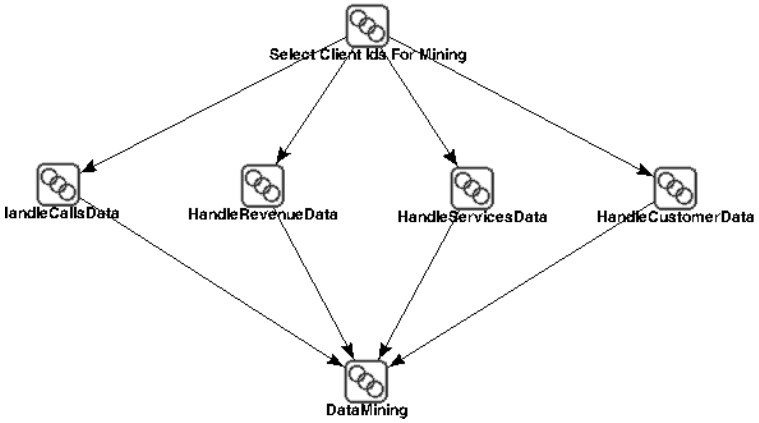

Figure 5.1.: The six preparation chunks of this application and their dependencies.

from the past six months, given the month when the customer churned, or the current month if the customer has not churned. The data in this model application covers two years, or 24 months. From these two years, two months are chosen (called the churn months below) and all customers who have left the company in one of those months are taken as positive examples for churning. The negative examples are gathered by taking all customers who have not left the company in the two years. During deployment, only this last group is available. During training, adding the two groups of churners results in three six-months-periods that provide past data for the customers. The model case is designed such as two easily allow to change the two churn months. Thus it can happen that the three periods overlap, since a given month may be the first month of one period and the fifth, say, of another. Therefore some parts of the preparation graph must be applied to the three periods separately, as detailed below.

The general goal of this data preparation process is to transform the given data such that one entity describes one customer in the resulting mining table. Data from the past six months is therefore given a representation that provides some attributes for each of the six months, so that all the information for one customer is attached to a single entity. Thus the resulting mining table has many attributes (98 to be precise) and the choice of the relevant ones is left to the mining algorithm, a typical approach when there is little intuition as to which attributes might be most important.

The data sets that are used in this KDD process are described in the following sections, because each section deals with one chunk of the preparation graph, and in this application each chunk corresponds to the preparation of one data table. In the final chunk, the results of each chunk (their last output concepts) are joined and mining is applied (section 5.7). Figure 5.1 shows the six chunks of the application and their dependencies, which are given by the data flow. Each chunk corresponds to one section below.

Various parts of this example process exemplify the high-level data preparation tasks introduced in section 2.1.3, as will be indicated.

Attribute	Type	Explanation
Caller	Key	Customer Identification
ServStart	Date	Date when service started to operate for this customer
ServEnd	Date	Date when service ended; missing if still operating
PayMethod	Set	Method of payment used by this customer
Handset	Set	Type of device used by this customer
TariffType	Set	Type of tariff booked by this customer
TariffPlan	Set	Type of tariff booked by this customer

Table 5.1.: The attributes of the Services concept, an input to the KDD process.

5.2. Selection of data for preparation

This part of the process exemplifies the high-level preparation task *data reduction* (see section 2.1.3). The input data to this chunk is a table from the service department of the telecommunications company that contains information for each customer about the services offered to them. Table 5.1 explains the attributes of the corresponding concept.

Since this table provides the information whether a customer has left the company or is still in contract, two tasks can be solved based on this data: the selection of a suitable subset of customers for preparation, and the construction of the label for mining. Figure 5.2 shows the process-oriented view of this chunk, that is, the graph of operator applications, as realised in the MiningMart system which is described in the following chapter. In MiningMart the nodes of the preparation graph are called *steps*; they can be named, and represent the application of an operator.

The two steps `MarkSomeChurnedCustomers` and `MarkNonChurners` mark all customers that belong to the first or second churn month or to the non-churners by a special value of the new attribute. `MarkSomeChurnedCustomers` is realised by an operator that allows to discretise attributes of the type *date/time* into discrete values, by giving time intervals. `MarkNonChurners` uses a general ATTRIBUTE DERIVATION; it marks all customers who have not left the company, as indicated by a missing value in the `ServEnd` attribute. This extra marker is needed for later unification with the churned customers data set, see below.

Note that in MiningMart, operators that create a new attribute do not also create a new concept, but simply add the new attribute to the input concept, in contrast to the discussion in section 4.2 where all operators are proposed to create a new output concept. This exception produces fewer concepts in the concept web, allowing a clearer overview of the process artifacts. At the same time it may require to update semantic links: if a concept that is a separation of another concept is extended by an attribute, the semantic link between them is changed to a specialisation. Such updates are not made when the concept web is displayed in MiningMart's concept editor, in order to reflect the *creation* of concepts; a different approach is possible here, namely to adjust the semantic links where necessary.

Since the two markers for the churners and non-churners are added to the same input concept, they must have different names. After the two selection steps have created new

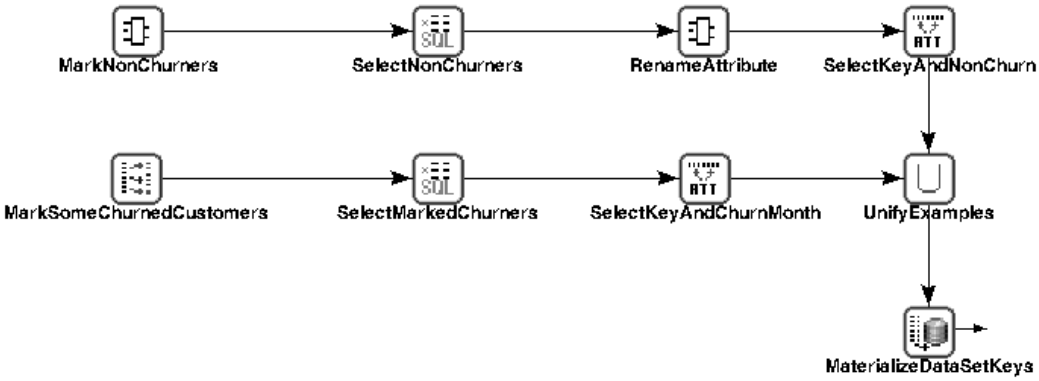

Figure 5.2.: Selection of customers and construction of the label for training.

concepts, one of these attributes can be renamed which allows the unification of the concepts. But before unification, the two attributes that provide the unique identifier for each customer and the churn marker are selected, so that the result of this chunk is a concept that maps customer identifiers to churn information. This concept will be joined with the other input data sets in other chunks, using the customer identifier attribute `Caller` as key, in order to provide the selection of customer data for the preparation process. This is a modular design meaning that to get a different selection, only this chunk (indeed only the first two steps) has to be changed, which makes it easy to reuse the application on updated data on a regular basis. Finally the resulting concept is materialised in the database, which is useful because MiningMart realises intermediate concepts as database views, and the nesting of views should not be too deep.

Figure 5.3 shows the data-oriented view of this chunk. The two concepts `InactiveClients` and `ActiveClients` are separations of the input concept `InputServices`, created by the two steps `SelectNonChurners` and `SelectMarkedChurners` from figure 5.2; at the moment of selection there are eight attributes (called *BaseAttributes* in MiningMart) in the input concept, but one is different for the two separated concepts as explained above. The ninth attribute in `ActiveClients` is the renamed attribute created by the step `RenameAttribute` (figure 5.2). The two concepts `Churners` and `NonChurners` are specialisations of `InactiveClients` and `ActiveClients`, respectively, as explained above, and they are unified to get the concept `TrainingSetKeysAndChurnInfo` whose materialisation is `TrainingSetKeyAndLabel`.

Comparing figures 5.2 and 5.3 illustrates the dual approach to KDD introduced in section 4.6, and the complementarity of the process- and the data-oriented view. In the following sections, usually only one view will be given as this suffices to understand the application, but another comparison of both views will be given in 5.5. The complete concept web of this KDD application is given in figure 5.10 on page 88.

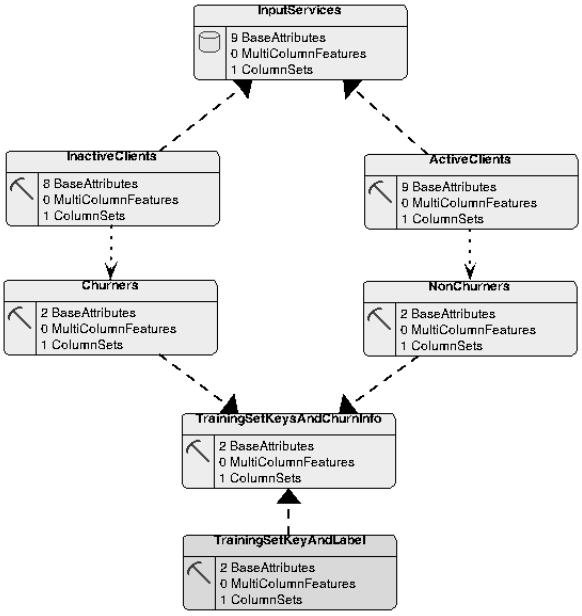

Figure 5.3.: The concept web created by the preparation graph in figure 5.2. Dashed arrows represent separations, dotted arrows represent specialisations.

5.3. Creation of the label

This chunk is shown in the process view in figure 5.4. Like the previous chunk it also handles the services data (see table 5.1). Mainly three new attributes are computed which are useful for mining. The first step joins the services concept with the result of the previous chunk which selected the customer data to be used for mining. As a result, the churn marker attribute is available in the output concept, and its instance contains only those customers that are in the selection for mining.

The next three steps serve to compute the number of years a customer has been with the company (the service length), since this is hoped to be an indicator of customer satisfaction. Two steps extract the year from the dates that mark the beginning and end of service; the following step computes their difference, taking the current year instead of the end-of-service year for those customers who are still with the company. After a materialisation of the data set, the service length is discretised into three intervals using DISCRETISATION (A.5.1). The next step constructs a binary label for training (positive or negative) based on the churn marker attribute; this step is not needed during deployment. Then some spelling errors in the `TariffType` attribute are corrected in the step `Repair-TariffType` which employs an instance of VALUE MAPPING (section A.5.3). Finally some attributes which are not needed for mining (for example those that were only used as intermediate steps to compute the length of service) are removed using ATTRIBUTE SELECTION (A.1.1) to get the final result of this chunk.

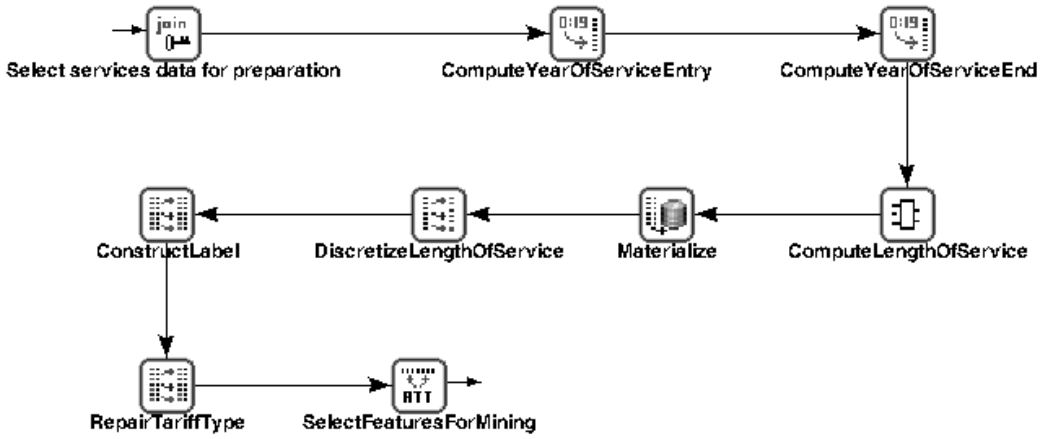

Figure 5.4.: The steps to prepare the services data.

Attribute	Type	Explanation
Caller	Key	Customer Identification
Birthday	Date	Date of birth of this customer
Gender	Binary	Gender of this customer
Name	Set	Name of this customer
Address	Set	Address of this customer

Table 5.2.: The attributes of the Customers concept, an input to the KDD process.

5.4. Preparation of customer information

The input to this chunk is a table with personal customer information; table 5.2 explains the attributes of the corresponding concept. Figure 5.5 shows the four steps of this chunk.

After the join to realise the data selection, this chunk only computes the age of the customer (*feature construction*, see section 2.1.3), using the difference between the year in which the analysis takes place and the year extracted from the `Birthday` attribute, and removes the superfluous attributes `Birthday` (replaced by `Age`), `Name` and `Address`.

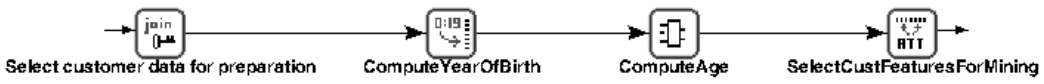

Figure 5.5.: The steps to prepare the customer data.

Attribute	Type	Explanation
Caller	Key	Customer Identification
Month	Date	Month for which revenue is given
Revenue	Continuous	Revenue generated by customer in given month

Table 5.3.: The attributes of the Revenues concept, an input to the KDD process.

5.5. Preparation of revenue information

This chunk prepares some information about the revenue (profit) made by the company from each customer. The data set for it comes from the accounts department of the company and provides the revenue that the company could generate for each customer in each month of the two years under consideration. This data has already been processed by the accounts department, but needs different preparation for the KDD process. Table 5.3 explains the attributes of the concept used as input for this chunk.

The figures 5.6 and 5.7 show the two views on this chunk, and provide a more comprehensive example of the duality discussed in section 4.6. The join that realises the data selection for mining (step `Select revenue data for preparation`) results in the concept `RevenuesToPrepare`, which is a specialisation of the input data (`InputRevenues` and the customer selection result of the first chunk, `TrainingSetKeyAndLabel`. In the concept web all join results can easily be recognised because they are the only concepts with more than one outgoing specialisation.

The next step deletes some entities from the concept's instance because the `Revenue` value is missing. There are not many entities where this is the case, so replacement of missing values was not deemed necessary by the analyst. This is *data cleaning*. The resulting concept `RevenuesNoMissingValues` is a separation of `RevenuesToPrepare` because it includes fewer entities but the same attributes.

The following steps have to be applied separately for the three six-months-periods that provide the past data for the three customer groups (from two churn months plus the non-churners). The reason is that different months act as the first, second and so on month of the three periods, and there might be some overlap. So the three groups are selected; because the following steps create a new attribute (`AbstractMonth`) for the resulting three separated concepts, they have an additional attribute but the link to `RevenuesNoMissingValues` is a separation (see the remarks above in 5.2). The abstract month attribute serves to give identical markers (numbers 1 to 6) to the six months in the three periods. Then only these months are selected, resulting again in three separated concepts.

The main aim in this chunk is to provide the revenue value for each customer in six new attributes, corresponding to the relevant six months on which the prediction of churn behaviour is to be based. This is an example for changing the organisation of the data, one of the high-level preparation tasks identified in section 2.1.3. To this end, Pivotisation (A.3.2) is now applied. The pivot attribute is `Revenue`, the index attribute is `AbstractMonth`, and the Group By-attribute is `Caller`; the aggregation operator can be summation or maximum, as there is only one entry per month and customer in the

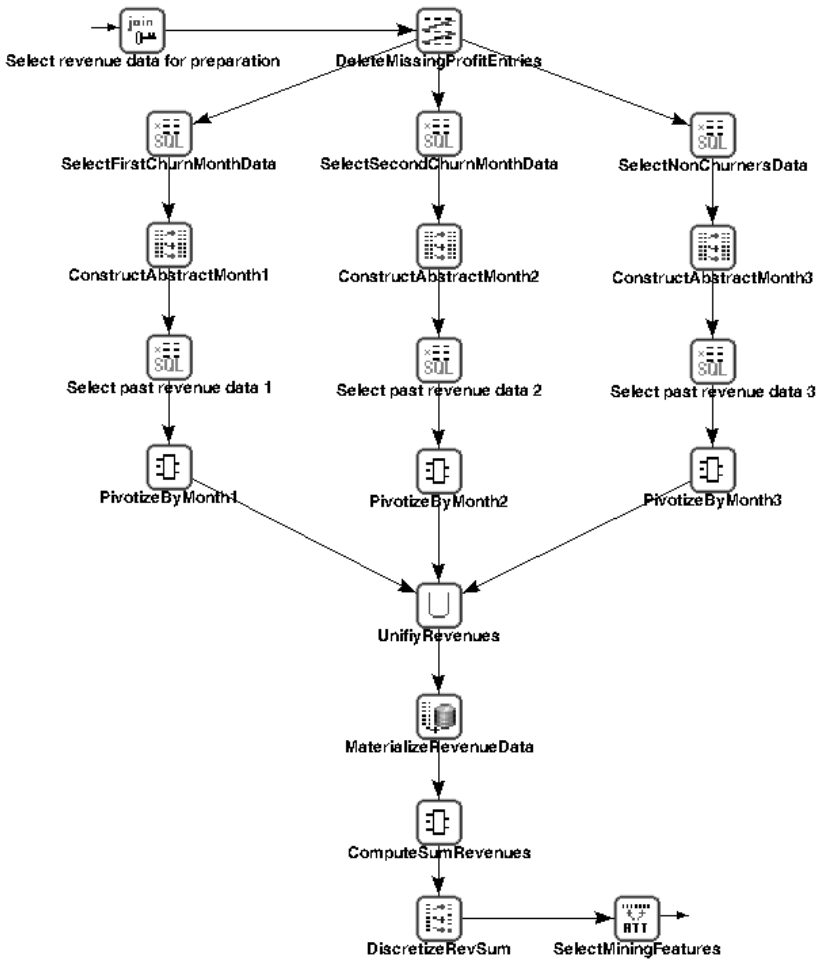

Figure 5.6.: The steps to prepare the revenue data.

input. The outputs are three concepts with the key attribute `Caller` plus six attributes containing the revenues for the six-months-periods. PIVOTISATION relates these concepts to the inputs by one-to-many relationship types, represented by the solid lines in figure 5.7 (there are several entities, namely one for each month, for each customer in the input, but only one per customer in the output).

Next, these three concepts are unified (operator UNION); the result is materialised; the sum of revenues during the six months is computed (ATTRIBUTE DERIVATION) and this sum is discretised (DISCRETISATION). Finally an ATTRIBUTE SELECTION removes the undiscretised sum. The corresponding steps and resulting concepts can easily be found in figures 5.6 and 5.7. The two views display a similar structure, demonstrating that both are equally suitable to represent data preparation processes.

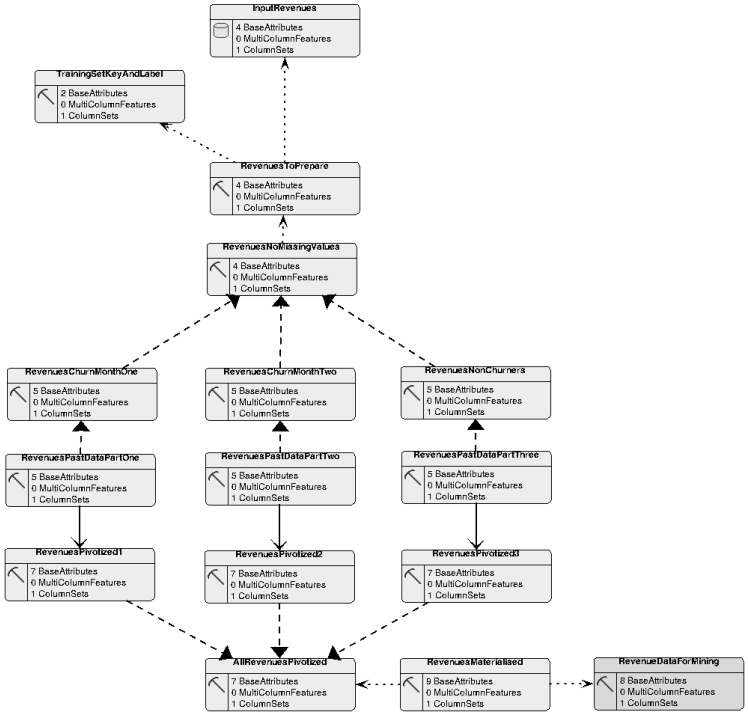

Figure 5.7.: The concept web created by preparing the revenue data. Solid lines represent relationship types.

5.6. Preparation of phone call information

The most important table describes the telecommunication behaviour of customers by storing features of each single phone call made by a customer, such as number called, length, tariff used, etc. This table is called the *Call Detail Records* table (CDR). Table 5.4 explains the attributes of the corresponding concept.

This chunk is the most complex one; figure 5.8 on page 85 presents its steps. This chunk also deals with the largest data set, as the CDR concept contains more than 61 million rows, holding the phone call details of 20200 Customers over a period of 24 months. The first step is therefore to join CDR with the concept representing the data selection for mining, `TrainingSetKeyAndLabel`. About 40 million rows remain. To reduce the data further, only the entities corresponding to any of the months in one of the three periods are chosen; to this end an attribute `Month` is derived from `Day` to indicate in which month each call took place. The step `SelectRelevantMonths` then selects only these months. Further, a new attribute `Peak` indicating whether the call took place during daytime or nighttime tariff is derived from `Hour`, using DISCRETISATION with user-defined interval bounds. The different types of calls are subsumed in a few groups like internet calls, mobile phone calls, abroad calls etc. by using VALUE MAPPING in the step `GroupCall-Classes`. The result is materialised because it is the basis for several sub-chunks of further preparation.

Attribute	Type	Explanation
Caller	Key	Customer Identification
Called	Set	Phone number called in this phone call
Day	Date	Date of phone call
Hour	Date	Time of phone call
Length	Continuous	Length of phone call in minutes
Units	Continuous	Number of tariff units used in the phone call
Class	Set	Type of call: call to internet provider, mobile phone etc.
State	Set	Indicates interruption of call due to malfunctions

Table 5.4.: The attributes of the CDR concept, an input to the KDD process.

Similar to the revenue chunk (section 5.5), unique numbers from 1 to 6 are given to the six months of the three periods, using a similar structure of three parallel preparation lines and their unification (in the step `UnionOfPeriods`). The steps `CountDroppedCalls`, `ComputeNumDiffCalledPersons` and `CountCallsPerClass` are examples for *feature construction*; they compute the number of calls that were dropped for technical reasons per customer, the number of different phone numbers each customer has ever called, and the number of calls to internet providers and free numbers; the latter are changed to binary flags indicating whether any such calls have taken place in the two steps `Check-Occurrence...`.

The step `ComputeLengthWholePeriod` and its successors compute some important attributes concerning the sum of minutes of call lengths customers have made in each of the six months. The step itself computes the sum for each customer and month (using Aggregation); then Pivotisation is used to compute six attributes for the six months. The sum of all call lengths in all months is computed in `ComputeSumAllMonths` and discretised afterwards. Some attributes that have turned out to be decisive for the success of the mining algorithm in the original application are computed in the step `ComputeUsage-Change`. This step applies Attribute derivation several times (a feature offered by the MiningMart system), and computes differences in the phoning behaviour, measured by the sum of call lengths, between the first and sixth month, the second and sixth and so on. Thus these attributes can give an indication of any abrupt changes in the usage of the telecommunication service.

The three steps `PivotizeLengthBy...` compute more detailed statistics based on the lengths of phone calls. The sum of these lengths is derived not only per month but per month and per type of call, where type of call includes: internet providers, distance calls, local calls, calls to mobile phones, calls to free lines, calls abroad, calls disrupted for technical reasons, calls during peak time and during nonpeak time. These nine types lead to 54 new attributes (nine sums of call lengths for each of the six months) using two-fold Pivotisation (see section A.3.2). Clearly, the availability of n-fold pivotisation in the MiningMart system simplified the computation of these attributes drastically; in another system where this application was implemented by the author, no pivotisation was available so that 54 applications of Attribute derivation had to be set up. This

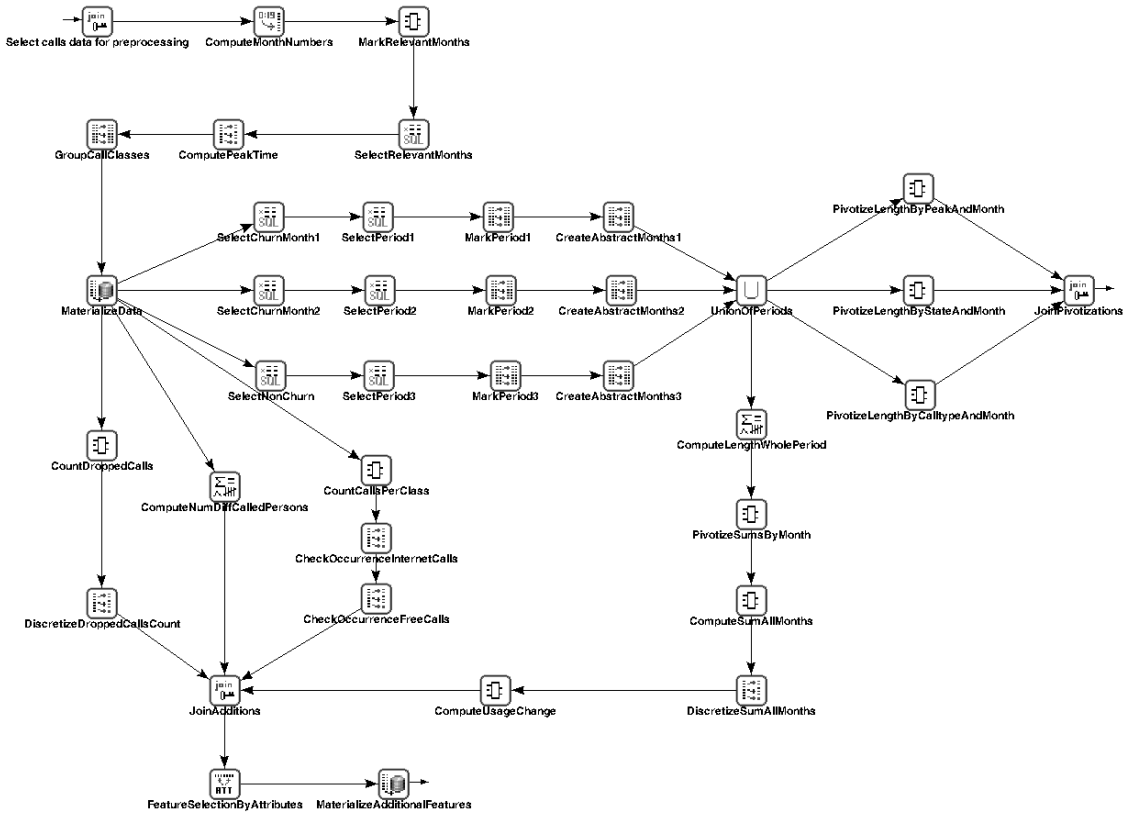

Figure 5.8.: The steps preparing the call details data.

demonstrates the usefulness of more complex operators. Those systems that offer pivotisation (compare table 8.3 on page 186) only support $n = 1$; in such systems, nine plus six simple pivotisation operators are needed.

The two joining steps `JoinAdditions` and `JoinPivotisations` are not necessary, strictly speaking, because their outputs are to be joined again in the following chunk (their inputs might as well be joined there). But they help to get a clearer structure of the chunk and a clearer connection to the following chunk.

5.7. Mining and deployment

The previous chunks have all produced several attributes with information about each customer. For final mining, five output concepts of the previous chunks are combined using a JOIN, using the customer identification as the key for joining. This is an example for *propositionalisation*. The result is a concept with more than 90 predicting attributes, one label and one key. Figure 5.9 shows the concept web produced in this chunk.

Two row selections then separate the positive from the negative examples; recall that negative examples are customers that have not churned. During training, a number of

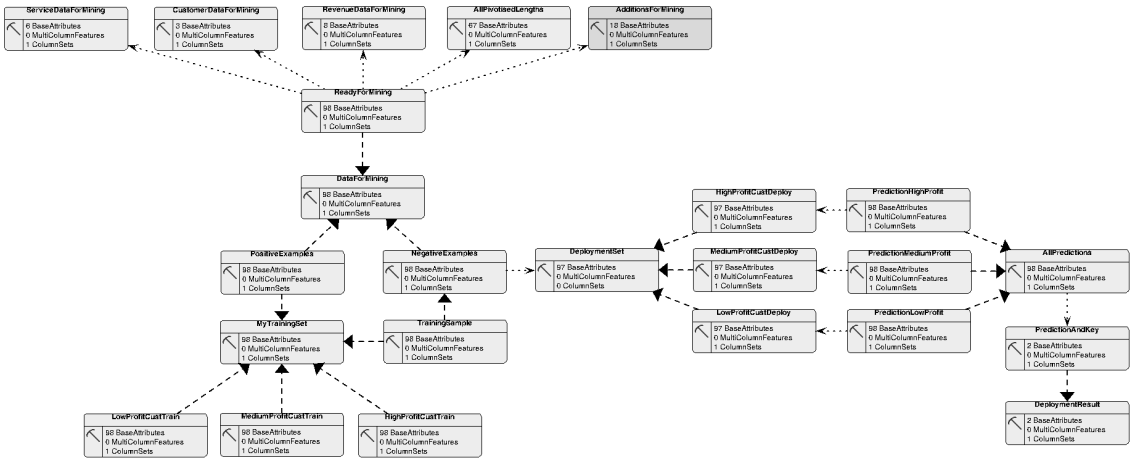

Figure 5.9.: The data view produced during training and deployment.

negative examples that is roughly equal to the number of positive examples is used, selected by a sampling row selection (the concept `TrainingSample`. During deployment there are only negative examples and there is no label; the label attribute is therefore removed from `NegativeExamples` for this phase.

The positive and negative training examples are unified, resulting in the concept `MyTrainingSet`, and used to train a decision tree. More precisely, the KDD analysts who developed the original applications got best results when splitting the training set into three parts according to three groups of customers according to how much revenue they generated for the company. Thus three row selection operators are applied to split the training set, and three copies of these row selection operators are used to split the deployment set in the same way; the resulting concepts have names ending in `...Train` or `...Deploy`, respectively.

The actual machine learning experiments consist simply of a 10-fold cross validation for each of the three decision trees, to evaluate their generalisation performance using the accuracy measure. In the original telecommunication application that this KDD example is based on, this performance was used to guide the creation and selection of the predicting attributes described above. Here only the resulting process is modelled, which led to good performance in the original application but whose performance on the random artificial data used here is uninteresting. The mining experiments were executed with the YALE system (Mierswa et al., 2006) to which MiningMart offers two interfaces, one operator for creating a YALE experiment that loads a data set created with MiningMart into YALE, and one operator for applying a YALE-learned model to a concept in MiningMart. The first operator is used on the three training concepts and the second on the three deployment concepts here. This second operator adds an attribute with the predicted value to the input. Note that its input must provide exactly the same predicting attributes as are used for training, otherwise the learned model cannot be evaluated. The three concepts resulting from prediction, named `Prediction...Profit`, are thus special-

isations of the deployment concepts. They are unified to collect all results. Finally, to ease the use of the predictions in arbitrary business processes (compare section 2.1.6), only the customer identifier and its prediction are selected and materialised in the database (the final concept `DeploymentResult`).

5.8. Discussion

The total number of steps used in all chunks of this model application is 98. Although some design decisions (such as when to materialise) are not determined exactly by the tasks that were performed, but can be varied, this application example clearly demonstrates the complexity of a longer KDD data preparation phase. Figure 5.10 shows the complete concept web from all chunks. While the chunks are not explicitly visualised in this figure, it is easy to recognise the general structure of the application: at the top are the four initial concepts (bearing a special database icon) with the typical star structure given by three connecting relationships; in the top left corner the concepts involved in the selection of data (section 5.2) can be seen, with the final result of a concept with the keys and labels in the top centre. Then the four chunks that prepare the four initial data tables begin by joining this central concept to each of the four initial concepts. In the bottom of the figure the five output concepts of the four chunks are joined and the resulting mining concept is further processed for training and deployment. Without the use of the two views and of chunking, it would be very difficult to keep an overview of the whole process; without the provided operators this would not be much simpler than with direct programming. The latter situation is demonstrated in appendix D.

However, some issues that may arise in real applications are not addressed in this model scenario. For example, the data for it was artificially generated; though some missing values and misspellings occur, real data is notorious for including other surprises. Another point is that no representativeness issue arises, while in real applications, the question whether the available data is representative of the phenomenon to be examined needs to be addressed. Also, the data for this use case was generated using consistent key relationships between the tables, whereas it may in reality be a problem to achieve this, or to get the data into relational tables in the first place. Further, this model application is used for the prediction of a binary label on unseen data, and the KDD process ends there. Thus no post processing of the label is needed (see sections 2.1.6 and 4.5). For example, if the label attribute had been scaled during data preparation, this scaling would have to be reversed for the predicted value before it can be used. Compare section 7.2.6 where an operator for this is discussed.

The mining phase is not included in this demonstration, in spite of its importance, because it is not in the focus of this work. Interfaces to the YALE mining tool box are given. Finally, the actual use of the predicted label, for example in a marketing campaign, is not modelled, though software support might well be useful here as well, for example for the generation of marketing letters using the addresses of customers predicted to churn soon.

In spite of these limitations, the model application served to collect relevant and significant experiences by realising it in different software tools. Chapter 8 contains evaluation criteria based on these experiences.

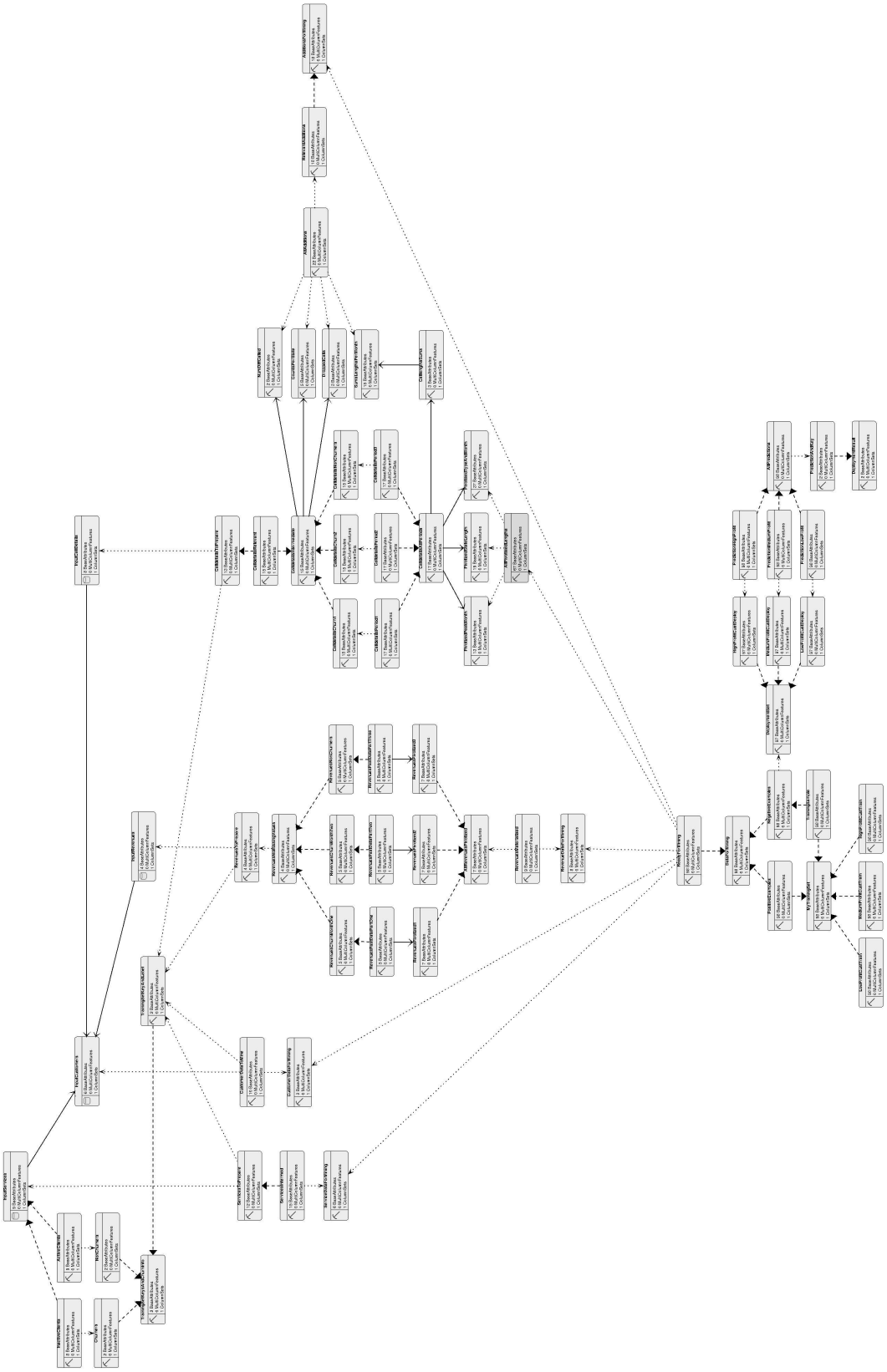

Figure 5.10.: The complete concept web of the model application.

6. Publishing Operational KDD Process Models

Chapters 3 and 4 have introduced a conceptual or task-oriented description of KDD processes and argued that modern KDD software should support this level explicitly. As chapter 8 will show, some elements of this level are supported by several modern KDD tools (if to a limited degree). This eases the daily work of KDD experts and allows a growing number of non-experts to attempt at developing challenging KDD projects. Though both experts and inexperienced users can find guidelines for their work in the CRISP-DM model, they are still faced with some essential problems, in particular those of finding a suitable data representation and of choosing and tuning a learning algorithm to give acceptable results. As mentioned in section 2.1.3, data preparation, as the subprocess that leads to the desired data representation, still consumes the largest part of the overall work. The main reason is that what is a good data representation depends on the mining task and data at hand, which poses a challenging problem. Knowledge Discovery is still more an art than a science (Pyle, 1999) as it involves many decisions that only humans can take (Brachman & Anand, 1996), so that inexperienced users need a lot of training (Kohavi et al., 2004). Such users would benefit greatly from sources of knowledge about how experts have solved past KDD problems, especially from exemplary, executable KDD solutions. Even the experts might find inspirations in solutions from other business domains if these were available to them. The need for an environment to exchange and reuse KDD processes has long been recognised in the literature on KDD, see section 6.1.2.

This chapter presents such an environment, called MiningMart. A brief overview is given in section 6.2. It is based on a *meta model* which is described in section 6.3, and which realises the two levels of description for all aspects of the KDD process as developed in previous chapters. Thus this chapter relies strongly on the concepts of the previous ones. An implemented system that directly translates KDD process models expressed in this meta model to executable SQL code is available. This system is mainly described in chapter 7. The present chapter concentrates on the aspects of MiningMart that are related to modelling and exchange of models. Thus, a web platform to publicly display the KDD process models in a structured way, together with descriptions about their business domains, goals, methods and results, is described in section 6.5 (based on (Haustein, 2002; Euler, 2005d)). The models are downloadable from the web platform and can be imported into the system which executes them (in this case, on a relational database). To support the claim that this web platform is useful for the exchange of knowledge about successful KDD processes, section 6.5.3 and appendix B provide implemented, publicly available, and reusable solutions of frequently occurring problems; further, the issues of reuse and adaptation, which are important for this exchange, are discussed in detail in section 6.6.

6.1. Related work

6.1.1. Related fields

To solve a problem by remembering a previous similar situation, and by reusing knowledge from that situation, is the core idea of case-based reasoning (CBR, e.g. (Aamodt & Plaza, 1994)). CBR approaches require (at least) to model previous problems with their solutions, and to match new problems to the collection of previous ones. In this work, a problem corresponds to a business question and given data, to which a KDD process is the solution. Section 6.3 describes how the data and the KDD process are represented, while section 6.5 explains how such representations are linked to descriptions of the business tasks and collected in a public, web-accessible repository. For problem matching, the system described in this and the following chapter includes basic schema-matching algorithms (Rahm & Bernstein, 2001) that map the data representation of an existing KDD process (from the public collection) to a new data schema. Matching of business task descriptions is not automatically supported yet, but left to the user of the web repository.

The idea that conceptual models of an application are easier to reuse than low-level implementations is an important motivation for the KADS project (Schreiber et al., 1993c). A particular idea from KADS is to make *control knowledge* (knowledge on how to control processes in a system) reusable by providing explicit models of it without the *domain knowledge* models that usually accompany it in the KADS framework. These templates are called interpretation models in KADS, because they can be used to guide the interpretation of new domains (Wielinga et al., 1993). Control knowledge in KADS corresponds to preparation graphs in the present work while domain knowledge mirrors data models. Thus there is a clear relation to work in knowledge representation.

Being an integrated environment for the creation, collection, retrieval and reuse of knowledge about KDD processes, and since it offers web access and is based on metadata, the web repository described in section 6.5 can be seen as a knowledge portal (Staab, 2002) to successful KDD processes, which broadly relates this work with knowledge management (e.g. (Holsapple, 2003)). The latter aims to make the right knowledge available to the right processors in the right representation (Holsapple & Joshi, 2003). It comprises the identification, acquisition or creation, distribution, utilisation, and preservation of knowledge (Probst et al., 1999), usually but not necessarily within an organisation, where knowledge is structured information, for example based on an ontology. In this work, an ontology of essential steps in KDD processes is given in chapter 4. The meta model explained in section 6.3 identifies the information that is used here to preserve, utilise and distribute knowledge about KDD processes.

A different area which is somewhat related to the present work is data warehousing, which deals with the collection, storage and non-learning based analysis of large volumes of data (Inmon, 1996; Meyer & Cannon, 1998). Clearly, knowledge discovery projects benefit from the presence of well-maintained data warehouses since the raw data can be expected to be cleaner and more complete. Also data warehouses tend to use internal data models, often under the term "metadata" (Vaduva & Dittrich, 2001). Metadata frameworks in data warehousing allow to model relational and object-oriented data (Vetterli et al., 2000); detailed, but slightly outdated surveys can be found in (Staudt et al.,

1999a; Staudt et al., 1999b). Standardisation efforts described in those references have led to the Common Warehouse Metamodel (CWM) defined by the Object Management Group[1]. An interesting idea would be to reuse data models expressed in such standard formalisms for KDD projects on the technical level. This idea suggests to extend these formalisms by the means to express a (KDD-specific) conceptual level. Unfortunately, at the time when the meta model described in section 6.3 was conceived, the standardisation of warehouse metadata modelling had not yet been mature enough to choose a widely used meta model that would ensure a high degree of usability, as Staudt et al. explain in the references above. Therefore the meta model to be presented here includes its own devices to document the data to be analysed on the technical level.

6.1.2. Related work in KDD

Reusing KDD solutions

The idea of collecting KDD solutions to enable their adaptation and reuse was already mentioned, as a plan for future work, by Wirth et al. (1997). However, no publications describing a working environment based on this approach seem to be available. The same is true for (Kerber et al., 1998), where an interesting methodology for the documentation and reuse of successful KDD projects is presented whose motivation is identical to the one for this work. So-called active templates are proposed there to link actions, results and documentations related to a KDD process, which is very similar to the web repository realised for this work (section 6.5). The importance of the reusability of KDD models is also stressed in (Zhong et al., 2001) and (Bernstein et al., 2005) (see below). Sections 6.5.3 and 6.6 cover templates and reusability in this chapter.

One reason why using existing KDD solutions as a template for new applications can be advantageous is that it is difficult to select a suitable machine learning algorithm for the mining step. It is a well-known theoretical result that no learning algorithm exists that can generally outperform any other learning algorithm on arbitrary data sets (Wolpert & Macready, 1995). Machine Learning research has experimentally confirmed that the choice of the learning algorithm to use depends highly on the data set at hand; see, for example, (Michie et al., 1994). Indeed, as was already mentioned in section 2.1.3, the *representation* of the learning problem (using the given data) is crucial for the success of a mining algorithm (Langley & Simon, 1995; Morik, 2000). Finding a representation on which a particular algorithm is successful usually involves much trial and error. This has motivated research on *meta learning*, as reported in (Pfahringer et al., 2000; Brazdil et al., 2003; Vilalta et al., 2004) for example. Meta learning attempts to generalise from characteristics of data sets and the respective performances of learning algorithms on these data sets, with the aim of providing advice on the choice of a learning algorithm given a new data set. Meta learning thus uses data characteristics and past solutions to assist in the development of new data mining and KDD applications; see (Giraud-Carrier & Provost, 2005) for a theoretical analysis of the soundness of this approach. Meta learning focuses on the mining step, in contrast to the present work which aims to support the whole KDD process.

[1] `http://www.omg.org/cwm/`

It has been argued that the accumulated results of the "first-level" Machine Learning research (on finding successful combinations of representations and mining algorithms) enable implicit meta learning by the Machine Learning research community (Giraud-Carrier & Provost, 2005). The framework presented in this chapter allows to document in detail which representation was used in a particular application, and also how it was created. It can thus help to make this implicit meta learning more explicit, by collecting detailed, operational models of successful pairs of data representations and mining algorithms.

KDD modelling languages

To document and store KDD processes requires a modelling language, or meta model. A well-known standard to model the KDD process is CRISP-DM (Chapman et al., 2000). While it gives an overview of different, interdependent phases in a KDD process and defines some terminology (see chapter 2), it is not formalised, nor detailed enough to model concrete instances of data preparation and mining operations based on it, and does not include a data model. An early sketch of a formal model of the KDD process was presented by Williams and Huang (1996); the process is represented by a four-tuple (D, L, F, S), where D represents the data sets, L is a knowledge representation language, F is an evaluation function that scores the interestingness of discovered patters, and S is a set of operations executed on the data. D and S are relevant here, but D is not developed to any detail in (Williams & Huang, 1996) while S provides only a rough classification of necessary operations without specifying the operations to any detail.

The new PMML version 3.0[2], a standard to describe machine-learned models in XML (Raspl, 2004), includes facilities to model the data set and data transformations executed on it before mining. However, it is not process-oriented, thus it does not allow to model a data flow through a complex KDD process, and the data model is restricted to one table. Other standards around data mining are Java Data Mining (JDM (Hornick et al., 2004)), which includes web service definitions, and SQL/MM Data Mining. Though extensible, they currently provide interfaces to mining algorithms rather than to complete KDD processes. Similarly, Cannataro and Comito (2003) present a data mining ontology to enable grid-based services, but it is currently restricted to the mining phase of the KDD process.

Recently, some new research attempts to employ grid infrastructures for knowledge discovery. A good overview is given in (Cannataro et al., 2004). To enable the execution of KDD processes on a grid, these processes have to be modelled independently from the machines that execute them, and heterogeneous data schemas and sources have to be modelled. In (AlSairafi et al., 2003), a Discovery Process Markup Language (DPML) is used, based on XML, to model the complete KDD process. Unfortunately, from the available publications it is not clear how comprehensive and detailed DPML is. In the GRIDMINER project (Brezany et al., 2003; Brezany et al., 2004), each step of the KDD process is provided by Grid services which can be dynamically composed into execution plans using a Dynamic Service Composition Engine (DSCE). The input for this engine is also an XML derivation called Dynamic Service Composition Language (DSCL), which is

[2] `http://www.dmg.org/pmml-v3-0.html`

used to specify the activities to execute together with their parameters. The meta model presented in section 6.3 could serve as a basis for grid-based processing in a similar way, as it declaratively models complete KDD processes independently of their realisation.

KDD process models are also useful in distributed data mining scenarios (see e.g. (Park & Kargupta, 2002)), where one often decides to realise parts of the KDD process, in particular data preparation, at each local site that stores parts of the data. Assuming homogeneous data schemas (Park & Kargupta, 2002), the same subprocess will be applied at each site, so that modelling it once while executing it at all sites can save a lot of efforts.

Recently, an XML-based middleware language, called KDDML, for the support of KDD applications has been developed (Romei et al., 2005; Romei et al., 2006). Middleware languages are used to exchange data between different applications, hence KDDML is designed to allow the description of KDD processes independently of their realisation. Elements in KDDML are operators, with functional semantics; this allows to nest operators like in Yale (Mierswa et al., 2006). Some operators return data tables while others return learned models, scalar values, or generic XML strings. For data access, special KDDML elements store the actual data location as well as metadata, including conceptual data types and data characteristics (see section 3.3.3). These elements are returned by data-reading operators. For both SQL data sources and flat files in the ARFF format, KDDML operators exist to read the data. Elements to model data preparation operators are also available; the current list is not long but easily extensible. Learned models are represented based on PMML (see above). Some other KDDML elements allow to apply, evaluate and post-process certain models, and to specify that some operations can be executed in parallel if this is possible in the interpreting environment. In sum, KDDML is a recent, rather powerful and extensible declarative language to describe KDD processes, with functional semantics. The approach shows some similarities to the Yale approach (Mierswa et al., 2006), but uses an explicit data representation. This is similar to M4, the declarative meta model used in MiningMart (see below). Thus KDDML uses many ideas that are also present in MiningMart, but has been developed and published several years later. Also, MiningMart comes with a complete system that includes a user-friendly interface, while KDDML provides a middleware that may be used by other applications, similar to a library of functions. To this end, a KDDML interpreter system is available, but no system that supports conceptual-level access to KDD applications modelled in KDDML has been developed. As another major difference to MiningMart, the interpreter system can access relational databases but does not leave the data inside the database, as MiningMart does. Instead, the data is read into main memory. Thus MiningMart can process much larger data sets.

Systems for KDD processes

Early knowledge discovery systems (see e.g. (Matheus et al., 1993)) were focused on the mining step, in that they provided mainly a set of learning algorithms without much support for other phases. Brachman and Anand (1996) saw the need for more support early on, and proposed to use the term *knowledge discovery support environment* for systems that would provide at least a closer integration with databases and support for other phases (they included requirements on the mining phase that need not be detailed here). This chapter presents such an environment; other attempts to construct such

environments are discussed in the following.

There have been two approaches that provide some intelligent assistance to KDD users in setting up their processes. Such approaches also require some model of the KDD process (see above). Basic steps in a KDD process are realised by agents in the work of Zhong et al. (2001); meta-agents (planners) organise them to a valid process using their input and output specifications. The authors provide an ontology of KDD agents that distinguishes between three phases of the process, namely preprocessing, knowledge elicitation (mining) and knowledge refinement (which corresponds to post-processing as explained in section 2.1.4). Another distinction is that between automatic agents and agents that need some human assistance (interactor agents). Concerning data preparation (preprocessing), data collection, cleaning and selection are mentioned as interactor agents. Automatic data preparation agents are restricted to discretisation and several segmentation algorithms (in the terminology from chapter 4). This particular choice of agents is not explicitly justified in the published articles.

Zhong et al. (2001) also provide a data model, which is rather different from the one described in this work (section 6.3.1) in that it stores the stage of the KDD process that a data set results from. Thus their data model distinguishes between raw data, clean data, selected data (a subset of the whole data set), changed data, and segments of a data set (e.g. clusters). However, it does not model tables or columns. Similar to the way in which the input and output of the operators of chapter 3 is specified in terms of the semantic data model given in section 3.2.2, the operating agents in (Zhong et al., 2001) have input and output specifications that use the data model given there. The limited data model thus translates to limitations on the possible processes, which is probably necessary to enable the automatic planning of such processes.

The authors stress the aspects of reusability and adaptability (compare section 6.6). Instead of including facilities to publicly collect and exchange process models, however, their approach relies on the planner to adapt an existing process to changed circumstances. This approach to adaptation is similar in (Bernstein et al., 2005), where a system to systematically enumerate and rank possible KDD processes is presented, given some input data and a mining goal. These authors have also developed a meta model for KDD processes, but it does not include a meta model for data which makes reusing their processes more difficult. The only type of information concerning the data that they model seems to be the continuous/discrete distinction, whether a column contains missing values or not, and a qualitative (binary) indication of whether the number of records or the number of attributes of the data set is large. Similar to Zhong et al. (2001), this model limits the possible, valid KDD processes because each operator specifies conditions on its input and output. For example, a logistic regression mining operator does not take discrete attributes as input.

Concerning the KDD process, Bernstein et al. (2005) also use the distinction into preparation, mining and post-processing (of models). Their list of preparation operators, which they do not claim to be complete in any sense, includes sampling, discretisation, dichotomisation, attribute selection and principal component analysis. At a higher level, their ontology of the KDD process includes *schemata* for complex processes which allow to constrain the search for valid processes by providing a template structure for the operator graph that the final process must have. The idea of such schemata is related to

the subgraphs that solve particular KDD tasks introduced in sections 6.5.3 and 6.6.

It was already noted in section 1.1.1 that the planning-based approaches cited above suffer from scalability problems: larger, real-world KDD projects are unlikely to be successful if their data preparation is limited to meeting the technical restrictions imposed by a mining algorithm, rather than also creating "meaningful" mining input, i.e. input that allows to discover interesting patterns. However, little has been done in the planning approaches to account for this.

A special focus on data preparation is taken in the SUMATRA project (Aubrecht et al., 2002), which developed a special scripting language called Sumatra which is designed for data transformation. There is also a tool called SumatraTT (Sumatra Transformation Tool) that interprets data preparation tasks written in this language. SumatraTT uses abstract data objects to represent various data sources, resembling the M4 data model (section 6.3.1) in that this mechanism allows to formulate the data transformations uniformly (independently of the concrete data sources). These data transformations are programmed in the Sumatra language; however, SumatraTT provides an extensible library of *templates* of Sumatra code that can be reused on new applications by changing some template parameters. There is also a graphical user interface to connect data sources to abstract data objects, and to set up data preparation chains using the template library.

Knobbe (2004) has developed a tool called ProSafarii that supports preparation tasks with a focus on multirelational data mining. The tool is based on relational database technology, but uses an abstract data model where the foreign key relations are enriched by multiplicity (cardinality) information. The specialisation and subconcept relations proposed in chapter 3 are missing, though. The tool provides a few preparation operators that support data transformations of multiple concepts, in particular aggregation by relationships (section A.2.2), pivotisation (A.3.2), and joins. An additional operator available in ProSafarii, not described in the present work (but easily specifiable as a further operator), is normalisation, an operation that is well-known from database design. It splits an input concept into two concepts if the input concept contains a functional dependency between two attributes (see section 3.1.2), "sourcing out" the dependency into a second concept. Details can be found in (Knobbe, 2004) or any database textbook. Incidentally, Knobbe also describes a rudimentary methodology for data preparation, which is tailored to his focus on multirelational mining; it basically lists some high-level steps for selecting relevant information, adding derived attributes, discretisation of continuous attributes, joins, aggregation and propositionalisation (see also section 4.1.2).

Yale (Mierswa et al., 2006) is a system whose focus is on the mining phase; it supports the subprocess of mining experiments which are executed on a fixed input data table, and thus do not belong to the data preparation phase. Nonetheless Yale includes some data preparation operators, like discretisation or dichotomisation, which can be applied to single data tables.

Commercial systems that support the development of KDD processes are listed in section 8.5.

Integrating data and patterns

The present work attempts to model complete KDD processes, including many administrational issues, but does not focus on the central mining step. However, there have recently been some attempts to integrate preparation and mining aspects in a single, data-oriented view. The work by Kramer et al. (2005) has already been discussed in section 4.1.2; results of mining are seen as an additional attribute of the mining table. The operator ATTRIBUTE DERIVATION (A.5.4) proposed in this work allows this kind of integration.

The idea of offering unifying views on both the data and the mined *patterns* in the data is a little older, though. Most famously, inductive databases have been proposed as a single environment for the two (Boulicaut, 2004). This research concentrates on the frequent itemset mining paradigm. Within this paradigm, both data and patterns can be accessed by the same query language (Boulicaut et al., 1999), which allows to view a complete KDD process as a sequence of such queries – an appealingly compositional approach, though no conceptual-level counterpart has been developed for it yet. A declarative framework for inductive databases called XDM is available (Meo & Psaila, 2003), which is based on XML and related standards. It can represent the data, the process and mined patterns in the data. However, XDM is independent of specific formats for modelling the data or patterns: it provides a generic and flexible framework which needs to be filled. Currently, there do not seem to exist comprehensive realisations of the framework, nor systems to support it, and the framework does not employ a conceptual level.

A different interesting contribution in this area is (Johnson et al., 2000), where data *regions* are proposed as a single formalism to describe relational data, including various stages of its preparation, and the results of mining it. For example, the operator ROW SELECTION (section A.1.2) can be seen as cutting a region out of a given data set; and similarly, decision trees partition the input data into a number of regions. This framework is appealing conceptually but has never been implemented.

MiningMart

This chapter describes some results of the MiningMart project, whose earliest stages have been described in (Kietz et al., 2000). An early design of the meta model presented in section 6.3 has been given in (Kietz et al., 2001), a more complete documentation can be found in (Morik et al., 2001) while the final version is (Scholz & Euler, 2002). The compiler, which operationalises the KDD models (section 6.4), has been presented in (Morik & Scholz, 2004) and in more detail in (Scholz, 2007). Details on the web repository have been presented in (Haustein, 2002), and on the current version in (Euler, 2005d). Compared to the Sumatra project, MiningMart uses SQL as the low-level data access and transformation interface, instead of the newly invented scripting language Sumatra. While the Sumatra language perhaps allows easier and richer manipulations at the technical level, it seems less apt to hiding the technical level from the user even considering the templates. Data resides inside the database under MiningMart, instead of being read into main memory like in SumatraTT or in the KDDML system.

This chapter contributes MiningMart's public repository of KDD process models and its technology (section 6.5), based on (Euler, 2005d), after explaining the basics of Min-

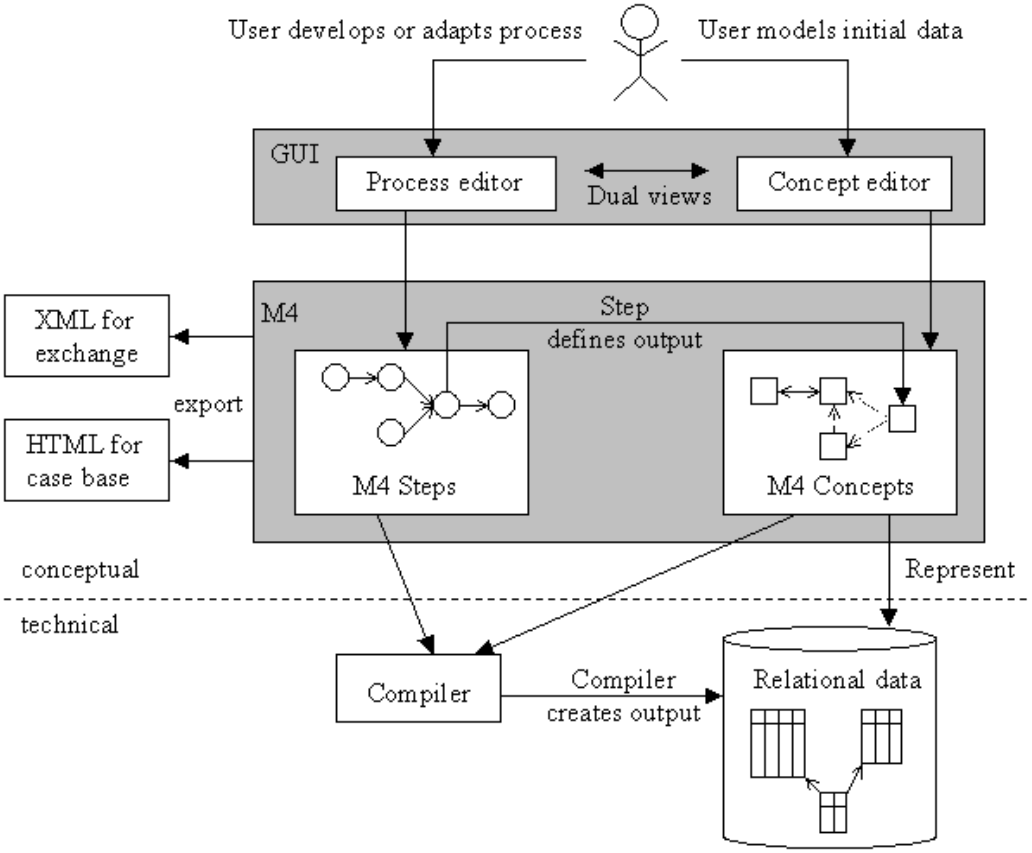

Figure 6.1.: Overview of the main components of MiningMart.

ingMart, especially the meta model M4. Further, it focuses on the reuse of KDD processes and the kinds of adaptation that may be necessary (section 6.6).

6.2. MiningMart overview

Figure 6.1 provides an overview of the main MiningMart components. The meta model, M4, is used to model both the data, via concepts, attributes, etc., and the preparation process, which consists of a directed acyclic graph of *steps*. Every step represents the application of one specific operator to one or more concepts. As the figure illustrates, the user's work at the conceptual level defines the steps of a KDD process model, and the conceptual data model of the output of each step is created by the system as soon as the step itself is created. When the MiningMart *compiler* is executed, it creates the actual output at the technical level (the actual views or tables in the database).

To give a clearer picture of how MiningMart's components interact, the following paragraphs sketch the two main use cases that MiningMart supports, the development of a new KDD application and the reuse of an existing one.

Use case 1: Developing a new KDD process

The given data, which must reside in a relational database, is imported into MiningMart, where it is represented using elements of the conceptual data model. During import, all the information in the relational source is exploited, in order to get an adequate representation; see section 3.2.2 where mappings between the technical and conceptual data models are discussed. Another way of modelling the initial data (at the conceptual level) is to do it manually, and connect the resulting model to actual data. In any case the conceptual model can be extended, by the user, with information that is not present in the database, such as the mining-related attribute roles, the conceptual data types, or any missing semantic links between concepts. All these tasks are performed using the concept editor.

Users can then set up a model of the process in the process editor, using the concepts and attributes of the data model as parameters of the steps. As soon as the input for a step is defined, by the step's operator it is determined what the output concept must look like. Another way to put this is to say that the *schema* of the output is created as soon as the user specifies the operator, but on the instance level, the actual data that fills the output schema is created later, by the *compiler*. Thus, only when the process is executed, the compiler creates the actual data that the process produces. This way of development separates the *modelling* from the *execution* of a preparation process. This is very useful for handling large data sets. It is enabled by employing the rather specific, powerful operators from appendix A, whose output schema is determined by their input schema and their actual parameter settings.

Throughout the process the user is supported by the administration of the conceptual data types of the attributes of the various concepts. The data types allow to ensure the technical applicability of the preparation operators, by observing the operators' constraints; this supports the explorative nature of the development, since it helps to avoid invalid experiments. However, the constraints can only help to observe the *technical* requirements of preparation operators and mining algorithms. Such issues as are related to the "semantic" validity of the process would concern the question whether the achieved representation has the potential to help the mining algorithm discover valid, novel and useful patterns. As noted in chapter 1, human understanding is indispensable in this area, and support can only be given by a case-based approach, as motivated in the beginning of the present chapter. Use case 2 below deals with this approach.

Use case 2: Reusing a previously modelled KDD process

Complete models of preparation processes can be exported from and imported into MiningMart. Only the conceptual level is concerned here. The web repository of such models (section 6.5) is the central platform for the exchange of models between users. Importing a KDD model into MiningMart means that the concept web representing the input data and all intermediate data sets is available; further, of course the process model (operators with their parameters) is available.

The next step is to choose a point in the modelled, imported process where the intermediate result, in terms of the created data representation, is most similar to the user's own, local data. It can also be the model of the original input data. The concepts from

this point can then be connected to the own data, and all operations after this point can directly be executed. Details about these issues follow in section 6.6.

In the remainder of this chapter, section 6.3 explains MiningMart's meta model M4 in detail. Section 6.4 briefly discusses the compiler component, while section 6.5 explains the export functionality of figure 6.1 and the web repository of KDD models. More dynamic and technical aspects of the MiningMart system are discussed in chapter 7.

6.3. A meta model for KDD processes

This section explains the structure and some details of the meta model (called M4) used in the MiningMart environment. More details can be found in the technical report (Scholz & Euler, 2002). M4 is structured along two dimensions: the data vs. process dimension as discussed in chapters 3 and 4, respectively, and the technical vs. conceptual dimension that is introduced in section 2.2. Section 6.3.1 shows the data meta model, while section 6.3.2 explains the process part of the meta model. A data model and a process model together describe an instance of a KDD process and are called a *case*.

The meta model provides a list of *types* of *objects*, and defines possible references from one object to another. It can be expressed in various ways, for example a relational database schema or an XML DTD. In a database schema, there is a database table for each type, and the table entries represent objects of that type; possible references between objects are expressed through foreign key constraints. In XML, an XML tag can be provided for each M4 type, and special tags can represent the references between M4 objects if each object has a unique identifier, but the constraints on the references are not so easy to express. Because of this, the MiningMart system currently uses a database to store the case models while working with them, but uses XML for their import and export as this eases their exchange between platforms. In this chapter, the M4 examples are displayed using the more legible database representation of M4. A further possibility to inspect (the conceptual level of) M4 is given in the web repository of successful KDD cases, see section 6.5.

6.3.1. Modelling the data

The data model in M4 directly realises the application of the two description levels. It provides specific types for the lower logical data model and further types for the conceptual level. Also, data characteristics and data types as discussed in sections 3.3.3 and 3.3.1 are modelled.

At the technical level, the relational data is modelled exactly as it resides in the database. The two basic types M4 offers for this are Columnset, which represents tables, and Column. The term "columnset" is used as an abstraction for data tables and database views. Figure 6.2 shows how a data table `EmplData` with columns `EmplId`, `EmplName` and `Salary` is represented in M4[3]. The technical data types are included for each column. In M4, only the difference between numbers, strings and dates/times is made at the technical level, to be able to decide whether certain operations on the data must involve

[3]Examples in this chapter are slightly simplified in that they show only those fields of the M4 tables that are relevant for the example.

Database table `EmplData`:

EmplId	EmplName	Salary
13	Smith	1300
...	...	...

Table Columnset:

ID	Name	Concept
41	EmplData	42

Table Column:

ID	Name	Columnset	Feature	Type
43	EmplId	41	46	1
44	EmplName	41	47	2
45	Salary	41	48	1

Table TechnicalType:

ID	Name
1	Number
2	String
3	Date

Figure 6.2.: M4 data model, technical level example. See also figure 6.4.

inverted commas for strings, or special characters like colons for dates/times. A further differentiation is not necessary, since the system that interprets the meta model deals with the data source-specific details.

This list of technical data types that MiningMart supports is also stored declaratively in M4, in what can be called the *static* part of M4. See the table in the lower right part of figure 6.2. The static part provides the information that does not change across KDD models (cases), in contrast to the *dynamic* part which stores the M4 objects that together form cases. Figure 6.3 illustrates that the static part stores knowledge about operators, data types, etc., and is read by the MiningMart system in order to correctly instantiate the dynamic M4 objects. The types Column and Columnset above, for example, are dynamic M4 types. Chapter 7 goes into more detail concerning how the MiningMart system and the two parts of M4 interact.

Not shown in figure 6.2 is the storage of (dynamic) information about the data characteristics of each column. This kind of metadata is needed at several places in a KDD process, as section 3.3.3 argues. In M4, the count of data records for a table can be stored as well as the minimum, maximum, average and median value for continuous columns, the number of unique and missing values for any column, and the number of occurrences of each value of a column. The MiningMart system computes this information on request by the user.

Another type of (dynamic) technical-level information that can be stored in M4, also not shown in figure 6.2, is information about primary and foreign key references that may be declared in the database. This information is needed to support the representation of relationship types at the conceptual level.

This lower data level of M4 is not exported, and thus not exchanged between users.

The higher level types in M4, which implement the conceptual data model from section 3.2.2, are Concept, Feature, ConceptualType, Role, Relation, Projection and Subconcept. The first four form the higher level correspondence to data tables, the concepts. Figure 6.4 exemplifies (dynamic) M4 objects of these types. The latter three M4 types model the three types of links between concepts used in the conceptual data model, by simply linking IDs of M4 concepts. The names are different for historical reasons, but the type Relation is used for relationships, Projection for specialisation and Subconcept for sep-

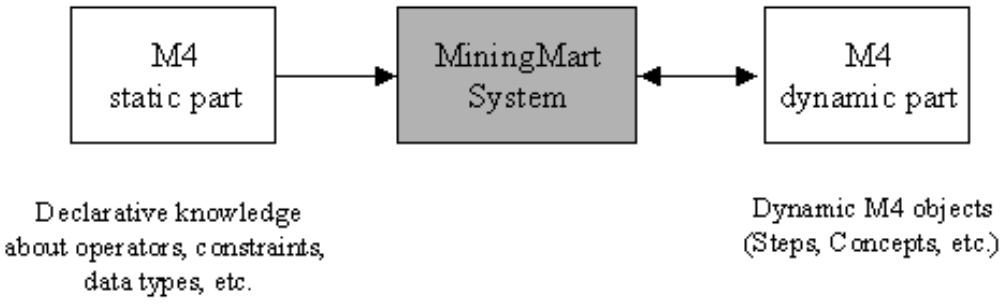

Figure 6.3.: Illustration of the way MiningMart makes use of declarative knowledge about its operators, stored in the static part of M4, to create dynamic M4 objects that together provide an actual KDD model. See also chapter 7.

aration. Cardinalities of relationships are not yet supported in M4. The conceptual data type of each Feature is stored as a reference to the static M4 type ConceptualType. The mapping between the levels is represented by the link between Columnsets and Concepts shown in figure 6.2 (field ConceptID of the Columnset table).

The conceptual-level elements can also be annotated using free text; such annotations are stored as objects of another M4 type, Docu. These documentation objects refer to the M4 object they annotate by the ID of that object (M4 IDs are globally unique; see (Scholz, 2007) for more details).

This realisation of the two-level approach in the data model allows to reuse the higher level elements, on new data, by simply changing the mapping (and perhaps adding or removing elements as discussed in section 6.6). For the mapping, each Concept corresponds to one table or view[4], a Feature corresponds to one column, and Relations correspond to foreign key links between tables. Not all Features of a Concept must be connected to a Column and not all Columns must have a Feature. This enables a more flexible use of the conceptual level. Subconcept links and Projections do not have a correspondence at the technical level, as they realise separations and specialisations, respectively.

The mapping between the levels is in general provided by the user. The concept editor supports the creation and manipulation of higher level elements and their mapping to given data. Further, as in use case 2 in section 6.2, concepts and relationships can be *automatically* created from a selection of database objects. This enables a quick set-up of the model that represents the initial data to be prepared. When this functionality is used, the system creates a concept for each database object (table or view), unless a table consists only of columns that refer to other tables by foreign keys, in which case the table is considered a *cross table*, i.e. one that realises a relationship. Such tables are not represented by concepts, but by many-to-many relationships. Similarly, many-to-one relationships are automatically created to represent direct foreign key links between tables. The conceptual data type of each feature is guessed from the technical types of

[4]To enable the kind of pseudo-parallel processing motivated in section 1.1.1, in fact a Concept can represent several tables or views of the same schema. Section 6.4 explains this, but here it is not discussed for clarity of presentation.

Table Concept:

ID	Name	Type
42	Employees	DB

Table ConceptualType:

ID	Name
4	Discrete
5	Continuous
6	Time
7	Binary

Table Feature:

ID	Name	Concept	Type	Role
46	IdNum	42	4	8
47	LastName	42	4	11
48	Salary	42	5	9

Table Role:

ID	Name
8	Key
9	Predictor
10	Label
11	No Role

Figure 6.4.: M4 data model, conceptual level example. The example relates to figure 6.2. In the example, the names of the concept and the features have been edited by a user to be more explanatory.

the corresponding column. All conceptual-level elements can be edited by the user at any time.

Once the mapping is done, all user work on the KDD process continues using the conceptual level, as can be seen in section 6.3.2.

6.3.2. Modelling processing steps

The difference between the static and the dynamic part of M4 (see above, figure 6.3) is more salient in the process model, to be discussed now. First the static part is explained, then the dynamic instantiation of operators.

The *static* part of M4 includes a schematic *specification* of all operators that are available in the system. The specification of an operator lists its name, its parameters, constraints, conditions, assertions, and a semantic link between input and output concept(s). This information is found in appendix A for each operator.

The M4 type Operator defines the name of an operator while the type Op_Param is used to define the allowed input and output parameters for each operator (compare the parameter definitions in appendix A). Further, the type Op_Constr can be used to define constraints on the instantiated parameters, and the type Op_Cond holds the conditions (see section 4.2 for explanations of constraints, conditions and assertions). Figure 6.5 exemplifies this part of M4 with the operator Discretisation (section A.5.1) (more specifically, the operator shown here discretises a continuous attribute given only the number of target intervals). The type Op_Param specifies for each parameter the name, minimum and maximum number of instantiations, IO type (input or output), and M4 type of the M4 object that can instantiate the parameter. In the example, the operator must be given, among other things, the number of intervals that it creates. This is only one value, thus the minimum and maximum number of instantiations are both 1. Another possible input to the operator are symbolic names ("labels") for the intervals it creates.

Table Operator:

OpID	Name
96	Discretisation

Table Op_Param:

OpID	Name	Min	Max	IO	M4Type
96	InputConcept	1	1	In	Con
96	TargetAttrib	1	1	In	Fea
96	NoOfIntervals	1	1	In	Value
96	Labels	0		In	Value
96	OutputAttrib	1	1	Out	Fea

Table Op_Constr:

OpID	Type	Obj1	Obj2
96	IN	TargetAttrib	InputConcept
96	TYPE	TargetAttrib	Continuous
96	TYPE	OutputAttrib	Discrete
96	GT	NoOfIntervals	0

Figure 6.5.: M4 process model, operator specification example.

The number of these labels is not limited beforehand, and the operator can also create its own labels, so that the parameter `Labels` is optional, thus its minimum number of instantiations is 0 and no maximum number is given.

To check the validity of an operator instantiation, constraints on the parameters can be specified. In the example in figure 6.5, for instance the conceptual data type of the target attribute (the one to be discretised) is constrained to be continuous, or the number of intervals is constrained to be greater than ("GT") 0. This declarative feature helps to ensure that only valid sequences of operators can be set up. It thus supports the execution-independent development of KDD process models.

Some operators can be applied several times to the same input. For example, the MiningMart versions of SCALING scale one attribute to a new range, but can also be set up to scale several attributes simultaneously. This facility is called *looping*. Only certain parameters of certain operators are loopable; a special constraint in the type Op_Constr signals this. For more details refer to (Scholz, 2007).

M4 also provides the type OperatorGroup which is used to bundle operators that solve a similar task. This feature is included for the convenience of the user, as the MiningMart system currently offers more than 80 operators; by the OperatorGroup type a taxonomy over these operators is defined, to provide a better overview. For example, the different discretisation operators (discretising attributes based on, e.g., the number of intervals to be created, the cardinality of the intervals, and other specifications) are grouped under one heading. Groups can be hierarchically arranged (nested). For the top level, the groups from chapter 4 that associate the operators to important high-level preparation tasks could be used.

Finally, M4 includes the type Assertion that can be used to specify some assertions that operators can make about their output. In particular, this type is used to declare the *separation* and *specialisation* links (section 3.2.2) between input and output of an operator. Section 7.1.1 explains how this information is used in the MiningMart system to instantiate these links when the operator is instantiated. More details about constraints

and assertions can be found in (Scholz, 2002) and (Scholz, 2007).

By adding specifications to the list of operators, M4 is easily extensible by new operators. The MiningMart project has filled the meta model with a large number of operators. All operators from chapter 4 and some further operators, for several mining tasks and some administrative data processing tasks (such as materialisation in the database, see section 7.3), are modelled. A slightly outdated list of all MiningMart operators and their parameters is given in (Euler, 2002c), the latest list is to be found in the MiningMart user guide. However, the functionality, or actual processing behaviour of the operator, is not declaratively specified in the meta model, but procedurally in the system that interprets it. This is the *compiler* functionality of the system; see section 6.4. Whenever a new operator is added declaratively, a new procedural module for the compiler must be added, too. A Java API is available for this; for details, see (Euler, 2002b).

So far in this section, the static part of M4 has been described, which stores the *specification* of an operator. For the *instantiation* of an operator in a concrete KDD process model, the M4 type Step is offered. The KDD process is a directed acyclic graph whose nodes can be Steps or Chains; the latter correspond to the chunks introduced in section 4.4, and subsume one or more Steps. Each Step employs exactly one Operator and uses the type Parameter to define the input and output of an operator. All parameters are instantiated at the conceptual level. Therefore, the complete KDD process is modelled using the conceptual level and can be applied to new data by changing the mapping to the technical level; see section 6.6.

Figure 6.6 demonstrates the use of the discretisation operator in a concrete Step. The parameters specify the input and output, the number of intervals, etc., following the specification in figure 6.5. For instance, the input concept for this step is the Concept with M4-ID 42, which is shown in figure 6.4. The parameter `TargetAttrib` is instantiated by the Feature `Salary` with the M4-ID 48 (see figure 6.4). The parameter `Labels` is not instantiated, which is allowed because it is an optional parameter. The parameter `NoOfIntervals` refers to an M4 object (with ID 55) of the type Value, which is used to store numeric or discrete values. The parameter `OutputAttrib` refers to a Feature that is created automatically by the system, as soon as the step is created; this is explained in section 7.1.

If the user instantiates a parameter in a way that violates one of the constraints mentioned above, the Step object is invalid and cannot be compiled. This would happen, for example, if the target attribute of this discretisation operator is already *discrete*, since there is a constraint (shown in figure 6.5) that requires it to be *continuous*. The reason for the invalidity is displayed in a message to the user whenever a constraint is violated.

6.4. Executing KDD models

In this section the MiningMart compiler (see figure 6.1) that produces the technical-level SQL code from the conceptual-level description of KDD models is briefly explained, for the sake of completeness. The MiningMart compiler was developed by Martin Scholz and is described in detail in (Scholz, 2007). Several operators were also implemented by the author of the present work, the more interesting of which are explained in section 7.2.

The compiler creates SQL code that can be used to execute the given KDD application.

Table Step:

StepID	Name	OpID
40	DiscretiseSalary	96

Table Parameter:

StepID	Name	M4ObjectID
40	InputConcept	42
40	TargetAttrib	48
40	NoOfIntervals	55
40	OutputAttrib	51

Figure 6.6.: M4 process model, operator instantiation (step) example.

Some examples for compiler-created SQL code can be found in appendix D. A possible future extension to MiningMart would be a system component that can process flat file data, which would require no changes to M4 because the lower level of the data model can model arbitrary data tables. However, to enable the management of large data sets, the current version of the system requires all data to be in a relational database. Another advantage of this is that many intermediate results (data sets) during the process can be realised as database views, which consume virtually no extra storage. Issues about materialising such views are discussed in section 7.3.

Continuing the example from figures 6.2, 6.4, 6.5 and 6.6, the compiler executes the discretisation of the `Salary` attribute into, say, three intervals (of same width). To this end the user has created the step `DiscretiseSalary` using the graphical editor of the system, and has set its parameters. The system creates an additional Feature in the input concept that represents the column with the discretised values, which as yet has not been created. The name for the new Feature is given by the user as the value of the parameter `OutputAttrib`. Details about this automatic creation of Features are given in section 7.1.

The user can now start the compiler, which will read the information about the step and create SQL code for the output parameter. In this case, the higher-level output is a Feature that the compiler connects to a virtual column. That is, the compiler creates an M4 object of type Column which does not represent an existing database column, but contains SQL code that computes its values based on an existing column, in this case the `Salary` column. In succeeding steps, this virtual column plays the same role as any other column; in particular, it can be used in database views on the original table `EmplData`. In this example, the SQL code might be: `(CASE WHEN EmplData.Salary < 1000 THEN 'Label1' WHEN EmplData.Salary < 2000 THEN 'Label2' ELSE 'Label3' END)`.

In order to be independent from the underlying database management system (DBMS), the compiler creates only standard SQL code. In fact, however, different DBMS offer slightly deviating SQL dialects, especially where metadata is concerned. For example, the numeric SQL data type is called `NUMBER` in Oracle systems, but `NUMERIC` in Postgres databases. Therefore, all queries to the database that concern database metadata have to be implemented for each DBMS separately. A Java API was created for this purpose by the author of this work, and implemented for Postgres and MySql database systems (while an implementation for Oracle was joint work with Martin Scholz).

The compiler is the system module that is responsible for administrating more than one Columnset for a single Concept. This is needed to realise the pseudo-parallel processing of different data sets with the same schema. All columnsets for the same concept

(representing database tables or views) must have the same columns and types. Usually, a concept has only one columnset, but the operator SEGMENTATION (section A.6.1) may split a columnset into several ones which are all attached to the same concept. When compiling a Step whose input concept has several columnsets, the compiler simply repeats its compilation on each of them, and produces a columnset for the output concept for each. Thus the number of columnsets attached to the output is the same as that in the input. In this way, the preparation of several data sets of the same kind can be hidden behind one conceptual model. This has been motivated in section 1.1.1, and is shown to be very useful in (Euler, 2005d). Two particular operators are available in MiningMart to control this, see section A.6. More details can be found in (Scholz, 2007).

6.5. A public repository of KDD models

This section describes the knowledge portal, called Case Base[5], that serves to distribute successful KDD models (cases) publicly without publishing the data they are based on. Section 6.5.1 elaborates on the motivation for setting up the portal, while section 6.5.2 presents the technical realisation. Then section 6.5.4 presents an algorithm for detecting common subprocesses in the case base, while 6.5.5 summarises issues related to the retrieval of cases from the case base.

6.5.1. Motivation

The creation of the web platform, or the case base, is motivated in the beginning of this chapter by the opportunities for knowledge flow that it offers. KDD experts can document and publish their expensively developed solutions for later reuse by themselves or by less experienced users, without having to publish the data sets their solutions are based on. But in fact, once such a platform is available, further advantages can be expected. Section 6.6 discusses reusable elements of KDD processes. One detailed example there concerns the reuse of subgraphs (chunks) of KDD steps which together solve a particular subtask. This example leads to the idea that many KDD processes exhibit a number of common patterns that can be collected as recipes for new KDD applications in a central "cookbook". In typical KDD applications, there is a number of reoccurring subtasks whose solutions are often similar. One might also expect to recognise further patterns for subtask solutions once a larger collection of successful KDD processes – the cookbook – is available. A pattern, then, is a part of some KDD process that occurs more than once among the processes in the case base, and is defined by a number of connected data preparation operators and the data flow between them. The problem of automatically finding such patterns is that of *frequent subgraph discovery*. Thus the case base is seen as a collection of graphs, where the nodes of the graphs are the steps (operators) of the case while the edges represent the data flow. As said in section 4.4, the resulting structure is called a DAG (directed acyclic graph). The graph nodes are *labelled* in this application, each label corresponding to one operator. Section 6.5.4 presents a frequent subgraph discovery algorithm for finding common KDD subprocesses in the case base. It identifies candidates for reoccurring KDD subtasks. Such candidates can be added to the

[5] `http://mmart.cs.uni-dortmund.de`

web repository as blueprints for solving specific subtasks, thus extending the collection of complete process models by a collection of sub-models for specific purposes (compare the example in section 6.6). Here the explicit representation of *chunks* of processing steps, as introduced in section 4.4, provides a useful tool to model subtasks.

Obviously such patterns, or solutions for frequently occurring subtasks, can also be created by hand, and can be published as a collection of chunks with extensive documentation. This idea has been realised for this work, as described in section 6.5.3. The manually created solutions can be seen as (part of a) tutorial on data preparation for KDD. Due to the public nature of the web platform, new solutions can be easily added by anyone, so that the tutorial can be assembled by collaborative efforts.

The technical framework described in the following (section 6.5.2) allows to download KDD models from the case base, and use them as blueprints for local KDD solutions. For example, the model application described in chapter 5 is available in the case base in all its details, under "ModelCaseTelecom". The framework supports KDD developers by offering solutions to data preparation and algorithm selection problems which have worked before in similar settings. Even if only parts of a case are reused, they can serve as starting points for a new application. In this way, a more collaborative style of work can be achieved once the collection of good case models reaches a critical size. More on reusing cases is said in section 6.6.

A limitation to this idea might be that the coupling between the conceptual data model and the actual data schema is rather close, since a concept corresponds directly to a data table or a database view. Thus it is possible to guess what the data schema looks like from a publicly available data model. Since some institutions would consider not only their data, but also their data schema as sensitive information, such institutions are not likely to publish complete data models in the public web repository. One remedy to this problem is to publish only a part of a case, leaving out the initial steps that deal with the original data tables, but still presenting the later steps which are more interesting in terms of the KDD methods applied.

6.5.2. Realisation

In the following, the MiningMart Case Base is described in some detail. It consists of a collection of HTML files that each represent an M4 object, and one top-level HTML page that points to the different cases. By following the HTML links from a case to its steps, concepts, and so on, the structure of the case can be explored.

Only the M4 types of the conceptual level are published, as the lower data level is considered confidential in many institutions. Figure 6.7 shows a UML model of those M4 types that occur in the case base, and how they are linked (these links are the ones that are realised at the object level by HTML links).

The first technical realisation of the case base is described in (Haustein, 2002). That version directly accessed a database with an M4 schema to publish *all* cases in that schema as HTML files, delivered on demand to a web server. This was possible using a software called Infolayer (Haustein & Pleumann, 2002; Haustein, 2006). The advantage was that once a case had been imported into that central M4 schema, it was publicly available immediately, and any changes were immediately reflected in public. A disadvantage was that the central schema had to be administrated separately, so that all cases

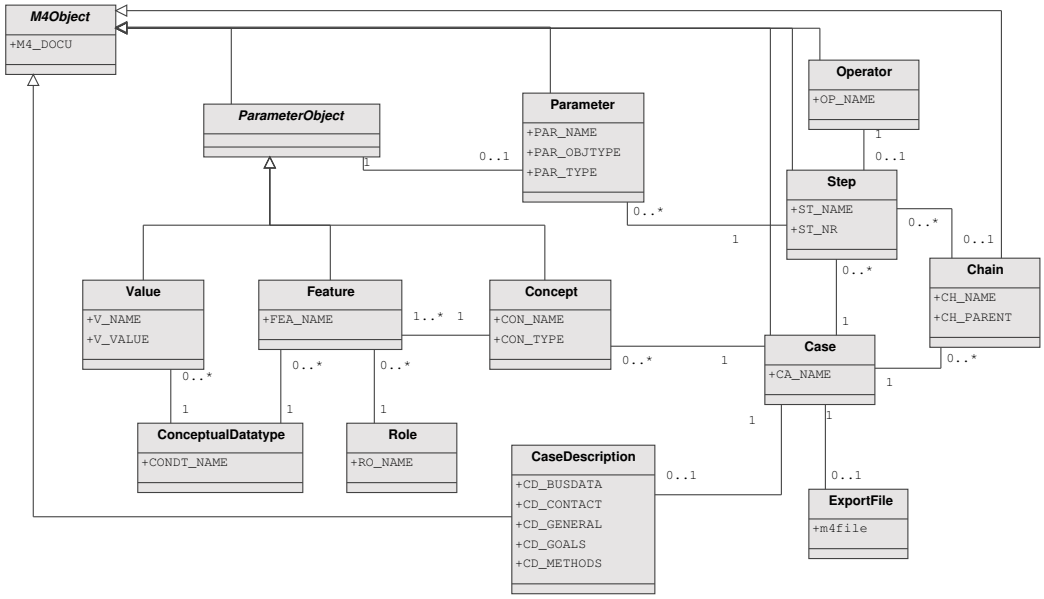

Figure 6.7.: A UML model showing the parts of M4 that are published in the Case Base.

to be published had to be exported into that schema. As another disadvantage, querying the M4 database online before creating the corresponding HTML page turned out to consume more time, before the page could be delivered, than users of web sites are accustomed to. Thus a second version was realised by the author of this work.

In the second version, the HTML files are created offline, either simply by exporting from MiningMart, or by a special program. A user selects the cases to be published from their M4 schema. The HTML files for these cases are created by the program, and can be stored directly in the public directory of the web server, so that they are immediately available like standard web pages.

The HTML-creating program exploits a recurring structure in the M4 schema: certain M4 types act as *containers* for others. For example, a Chain contains Steps, a Step contains Parameters, and a Concept contains Features. The program treats all container types alike, and creates a HTML file with links to the contained objects for each instance of a container type. HTML links are realised through file names that identify each M4 object uniquely; these file names are easily found by using the unique M4 id that is stored with every M4 object in the database (for more details on the administration of M4 objects, refer to (Scholz, 2007)). To ease navigation through a case, the path to the current M4 object is displayed at the top of the HTML page.

Figure 6.8 provides a screenshot of the case base as it displays the start page of a case, together with links to the case's properties, in particular to the concepts that represent the data input to the case, and to the chains (chunks) of steps that model the process.

When setting up a case with the MiningMart system, every object from the case itself to operators, parameters, concepts and features can be documented using free text. These

Your Navigation Path

Case Base
MiningMart

Case: ModelCaseTelecom

Has Chains DataMining
HandleCallsData
HandleCustomerData
HandleRevenueData
HandleServicesData
Select Client Ids For Mining

- **Exported into M4 file:**
  ModelCaseTelecom.xml.zip
- **Concepts used as input:**
  InputCallDetails
  InputCustomers
  InputRevenues
  InputServices

M4 Documentation:

A case with complex data preparation that demonstrates many useful operators and the two dual views on KDD processes. The data is from a telecommunications company, providing mainly detailed information about telephoning behaviour of customers. After bringing this information into a suitable representation, decision trees are learned to predict churning behaviour of customers.

Additional documentation for the Case Base:

- **Overview**

  This case demonstrates a complex application based on churn prediction in telecommunications. It exemplifies

Your Navigation Path

Case Base	Case	Chain
MiningMart	ModelCaseTelecom	HandleCustomerData

Step: ComputeAge

Name ComputeAge
Belongs to Case ModelCaseTelecom
Uses Operator GenericFeatureConstruction
Belongs to Chain HandleCustomerData
Concepts of resulting Data Model InputCallDetails
InputRevenues
CustomerDataToMine
Has Parameters SQL_String
TheInputConcept
TheOutputAttribute0
TheTargetAttribute0

M4 Documentation:

Computes the age of each customer as for 2001.

Figure 6.8.: Two screenshots of the Case Base as displayed in a web browser. Left: the start page for a case. Right: a step.

comments serve users for their own orientation in complex models. They are stored in M4 and appear on the web pages when a case is published (see the field `M4_DOCU` in the class *M4Object* in figure 6.7), so that other users browsing the case have a good orientation as to the purpose of each step in the KDD model and the use of its parameters. If such comments are missing, they can be added by the operators of the case base.

However, users who search for a case which they might use as an inspiration for their own KDD problem, or even as a blueprint of a solution, need some additional, more general information about each case. The most important types of information, following the discussion in section 2.1.1, are (i) the business domain, (ii) the business problem that was attempted to solve, (iii) the kind of data that was used, (iv) the mining task and other KDD methods that were employed, and (v) the results, both in terms of KDD (e.g. the quality of the model) and the original business problem (e.g. saved costs). Hence, exactly this information is provided together with every case that is presented in the case base. The program that creates the HTML pages asks the user to provide this information. It is organised in five fields for free text, corresponding to the five types of information above (a sixth field with contact information enables further inquiries by interested users). The filled fields are displayed in the case base as the first page of information about each case (see the left part of figure 6.8, under "Additional Documentation"). From there users who

are motivated by the descriptions can start to browse the case model, beginning with the chains of operators or the input data. In this way, the case model is linked with essential information relating the case to the context in which it was set up, which allows potential re-users of the case to judge its suitability for their own business problem.

Finally, each case is linked to the file into which the case was exported. It can be downloaded using the web browser, and be imported into a MiningMart client. There the conceptual level of the data model is available and can be mapped to local data sets as described in section 6.3.1. All parts of the case can be edited and adjusted if necessary; section 6.6 discusses the nature of likely adaptations. Once the adaptations are made, the case can be directly executed in the MiningMart system.

6.5.3. Templates

Section 6.5.1 motivates the idea that some subtasks in KDD tend to reoccur. Section 6.5.4 discusses the automatic discovery of such subtasks. In fact a certain number of small but general preparation problems, or subtasks, have been identified by the author of this work and by Pyle (1999) and Rem and Trautwein (2002).[6] These problems are usually very simple and have a straightforward solution. They are particularly interesting for inexperienced KDD users. The solutions are only informally described in the cited literature; however, the framework of this chapter allows their formalisation and direct publication for the first time. A collection of such formalisations can be seen as a (useful element of a) tutorial for KDD. Using MiningMart, contributions to such a collection can be made by anyone.

For this work the term *template* is used to refer to pairs (P, S) of problem descriptions P (using free text) and solutions S which are chunks in a MiningMart case. A special MiningMart case in which each chunk realises such a template has been set up by the author, in order to provide a public collection of templates. Formally there is no difference between a template and a complete KDD application together with a description of the problems it solves, but the term "template" will be used here for small solutions which typically involve one to three steps only. Templates are designed to be straightforwardly reused even by inexperienced users. In this way they also help professional users in saving development time. What is crucial for this purpose is an extensive documentation of the steps and concepts involved in a template.

The templates are designed to capture the essence of solutions to a number of typical preparation tasks, similar to the way design patterns are used in software engineering to describe abstractions of approved solutions to typical problems (Gamma et al., 1995). Like in software engineering, making profitable use of the templates requires their adaptation to concrete preparation problems by the user.

Since the templates are published in the case base, only their conceptual level is available. In particular no actual data comes with a template, in spite of its illustrative potential for how a template "works". Instead, the conceptual data is well-documented, down to the level of single attributes, so that small matching data sets can be easily constructed

[6] In addition, Knobbe et al. (2000) have proposed a few design patterns for typical mining tasks like cross validation or boosting. However these patterns concern the subprocess of experiments within the mining phase, and they are not developed to a formal description.

artificially using external tools. One might consider to modify the case base slightly, such that small illustrating data sets can optionally be published together with every case. However, this is not considered necessary to make successful use of the templates.

Appendix B lists short problem and solution descriptions under headings which are equal to the name of the chunks in the special template case. To retrieve formal solutions, please refer to the case "TemplateCase" in the MiningMart case base, and look up the structure and documentation of the chunk corresponding to the solution. The design of the templates followed a modular approach, attempting to isolate solutions to small preparation problems within a chunk rather than combining the solutions of several problems in larger chunks. For example, to prepare data before a support vector machine (SVM) can be applied, discrete values must be changed to a numeric representation by applying VALUE MAPPING or DICHOTOMISATION, and then all numeric values should be normalised to the real interval between 0 and 1. Rather than exemplifying these preparation approaches together under an SVM heading, they are demonstrated separately in different templates. This eases their retrieval in the case base. Larger preparation chunks can easily be created by combining the various modules. For a KDD tutorial, the direct availability of larger chunks is preferable, but the present work concentrates on the identification of small preparation modules to demonstrate solutions for basic tasks.

The list of templates given here covers the major preparation problems and solution methods to be found in the literature (see sections 4.1 and 6.1). Not all minor variants could be included, but the list of templates is of course open and can be extended by collaborative efforts. As it stands, it provides the first public collection of operational solutions to all major preparation tasks, a major step towards a public KDD "cookbook" or tutorial.

Each template description is structured as follows. The name refers to the chunk in the template case in the case base. A brief description of the problem and its solution is given. Then there is a list of ideas from the present work that are illustrated by the template. Finally, any MiningMart-specific notions that the template illustrates in addition to those general ideas are also listed. Pointers to other sections of this work are given where possible.

6.5.4. Finding common subprocesses

As explained in section 6.5.1, the problem of finding reoccurring KDD sub-processes in the case base automatically is that of finding frequent subgraphs in a database of graphs. The problem can be restricted, for this application, to directed acyclic graphs. A graph is a subgraph of another graph if all its nodes and edges can be embedded into the other graph. A subgraph is called frequent if and only if it occurs in the case base more than *minsup* times. *minsup* is called the *minimum support* threshold. Note that a subgraph might occur several times in the same graph (here, the same case). Only connected subgraphs are considered.

For general graphs, the problem is very hard because it involves solving at least the subgraph isomorphism problem, which is NP-complete (the general graph isomorphism problem is probably neither in P nor NP-complete) (Fortin, 1996). An additional constraint that makes the problem tractable for many practical applications is to require *labelled* graphs, which assign a label to each node and edge. In the case base, a node

label is easily found to be the type of operator the step employs. Edge labels are not available (using the concept whose data flow is represented by the edge would result in almost as many labels as edges), but edges are directed.

By using operator groups instead of operators as labels, a more abstract or approximate matching of subgraphs to graphs is possible, which may allow to identify higher-level substructures in the case base.

Frequent subgraph discovery algorithms are presented by Inokuchi et al. (2000), Kuramochi and Karypis (2001) and Borgelt and Berthold (2002). They are based on the Apriori paradigm (Agrawal et al., 1993) of level-wise bottom-up candidate generation and support thresholding, exploiting the monotonicity of the minimum support constraint, whereby a subgraph can only be frequent in the graph collection if all of its subgraphs, in turn, are frequent. A major problem in frequent subgraph discovery is to store candidates for frequent subgraphs efficiently, so that no redundant new candidates are created. To tackle this problem, the first two approaches cited above expensively compute *canonical forms* of graphs to uniquely identify subgraphs with the same structure. In contrast, Borgelt and Berthold (2002) allow somewhat redundant storage of candidates to avoid the computation of canonical forms. However, this requires to scan the output of the algorithm for identical structures. Central to their approach is the use of data structures that point from each candidate subgraph into positions in the graph database where it can be embedded. These pointers are called embeddings.

For the problem at hand, a frequent subgraph discovery algorithm was developed that is based on ideas from the work cited above, but incorporates some simplifications which are possible because the graphs in the case base are directed and acyclic. Since these two features reduce the search space, and since the number of graphs in the case base is not high, efficiency issues are not as pressing as in other applications.

Similar to Apriori, the algorithm proceeds from candidate subgraphs of size 1 (the size is the number of nodes) to larger subgraphs, filtering infrequent candidates at every level. For every candidate subgraph a canonical representation is computed that is the same for every graph with the same structure, regardless of the permutation of nodes. This representation is explained below. It allows to index the set of candidates uniquely, so that no two identically structured subgraphs are kept in the candidate set, which could happen because the same subgraph can be grown from different candidates of lower size. Together with every candidate subgraph, a list of all embeddings of this subgraph into the graph database is maintained. Growing a candidate of size k to size $k + 1$ means to explore all possible extensions of the subgraph by one node, in every embedding. That is, every such possible extension from every embedding yields a new candidate. By using the embeddings, candidate extension (or candidate generation) is constrained by the structures that actually occur in the graph database. Filtering infrequent candidates then means to count the number of embeddings that allow a given extension, and to reject candidates that do not meet the *minsup* threshold. These two steps, growing and filtering, are done at the same time in the presented algorithm because the maintained embeddings allow this easily. Note that growing a candidate may mean to extend it by an edge against the direction of this edge. This is necessary because some nodes (namely those labelled with the operators JOIN and UNION) can have multiple incoming edges. In other words, the DAGs are not trees.

Algorithm: Frequent Subgraph Discovery

Input: A collection of labelled graphs G and an integer threshold *minsup*
Output: A collection of subgraphs S that occur at least *minsup* times in G

```
1.  Create all frequent subgraphs of size 1 and their embeddings;
2.  S_1 = canonically sorted list of the subgraphs of size 1;
3.  k = 1;
4.  While S_k ≠ ∅:
5.      For every subgraph g from S_k:
6.          For every embedding e associated with g:
7.              For every node n in e:
8.                  For every neighbour n' of n that is not in e:
9.                      sup = number of embeddings of g that allow an extension
                              by an edge between n and n' (respecting labels);
10.                     If sup ⩾ minsup Then insert extended g into S_{k+1};
11.     k = k + 1;
12. return S = ⋃_k S_k;
```

Figure 6.9.: An algorithm to find frequent subgraphs in a collection of directed acyclic graphs with labelled nodes.

Figure 6.9 gives an overview of the algorithm. The first step is to create the frequent subgraphs of size 1 and their embeddings into the cases (line 1). This step requires linear runtime in the number of all nodes in the graph database. Note that there are atmost as many subgraphs of size 1 as there are labels (here, each type of operator corresponds to a label).

Secondly (line 2), these subgraphs are sorted canonically into the collection S_1 as explained below; S_k collects the candidate subgraphs of size k. Thirdly, the main loop of the algorithm starts (line 4). It extends every node in every embedding of every candidate subgraph, by following one of its edges that lead out of the embedding; then it counts the number of embeddings in which this extension is also possible (line 9). If the minimum support threshold is met (line 10), the newly extended candidate subgraph is frequent, so it is sorted into the collection of subgraphs of the current level (line 10). This sorting step requires to compute the canonical representation and to insert the representation into an already sorted list (line 10). The algorithm terminates when no embedding of any candidate subgraph allows an extension that meets the *minsup* threshold (line 4).

The computation of the canonical representation for graphs basically follows (Kuramochi & Karypis, 2001). The basic idea is to use a string representation of the adjacency matrix. However, two graphs with the same structure can have different adjacency matrices, because there is no global method by which one can sort the nodes of the graph canonically. The problem exists even for labelled graphs because several nodes can have the same label. To achieve a canonical representation one can compute all possible adjacency matrices for a graph, using all possible permutations of the nodes, and then choose

Algorithm: Canonical Form

Input: A labelled graph g
Output: A string that is the same for all labelled graphs with the same structure as g

```
1.  For every node n of g:
2.      Sort n into a bucket that corresponds to its
            label, its input and its output degree;
3.  Create all permutations of nodes within each bucket;
4.  For every combination c of the permutations from each bucket:
5.      Let l be the list of nodes sorted according to c;
6.      Compute the adjacency matrix m of g using node order l;
7.      Compute the string representation of m;
8.      insert m into a lexicographically sorted list s;
9.  return the first element of s;
```

Figure 6.10.: An algorithm to compute a canonical representation of a labelled graph.

the lexicographically first string representation. This is computationally expensive.

To reduce the computational demand, one can exploit the invariant properties of the nodes: their label, their input degree (number of incoming edges) and their output degree. To this end, all nodes of the graph are partitioned into sets representing the labels. Within each set, nodes are further partitioned according to different input degrees $(0, 1, \ldots)$, and then again according to output degrees. Within these last sets, all permutations of nodes are computed. Then every combination of local permutations gives an adjacency matrix, and the lexicographically first string representation of them is a canonical representation of the given graph. This method requires fewer adjacency matrices to be computed than without the partitions. Figure 6.10 shows the algorithm that computes the canonical representation.

6.5.5. Retrieval of public KDD process models

In this subsection, the case retrieval scenario is examined in some detail. Recall that a case is a complete KDD process model in MiningMart. But as mentioned in section 6.5.1 and further discussed in section 6.6, *parts* of a case can also be interesting for other users and thus worth publishing. Therefore the term *case* is used in this subsection to refer to both complete and partial process models. Some ideas from this subsection have not been implemented yet, but are suggested for future work.

The scenario for case retrieval is that a user in some institution has a number of data sets, called *local data* hereafter, that they want to examine using data mining. How can they get advice if no KDD expert is available? A suitable starting point is the additional documentation published in the case base for every case. Assuming a small case base (a low number of published cases), this information can be searched manually, but as the case base grows, automatic search methods are needed. By restricting publicly

available search engines to the MiningMart web domain `mmart.cs.uni-dortmund.de`, a service that at least Google[7] offers for free, the complete case base can be searched for keywords, including the documentation annotations and the names of M4 objects. Another useful way of approaching the case base can be offered by sorting the cases according to various topics extracted from the additional case documentation. The five slots of the documentation template provide five useful topics for indexing the case base. Further topics (such as type of business/institution where the application was realised) can be added by extracting this information from the free text descriptions in the slot. While automatic keyword extraction methods from the area of text mining (like (Euler, 2002d)) might be used for this, the size of the case base will probably grow moderately enough to allow a manual administration of such indexes. Other indexing methods are described in the following.

The business-related information will often not be enough to determine whether a published solution is suitable for adaptation to own data sets. A second method of approaching the case base is by looking for data models in it, called *target models* hereafter, that are similar to the local data sets. Section 6.6 explains how an automatic schema matcher may support this task. Currently, using this matcher requires downloading a number of candidate data models and determining the degree to which they match the local data, by applying the schema matcher, and comparing the results. In the future, the case base could be extended by an online schema matcher that allows to upload a local data schema and search among the target data models in the case base for similar data models. (A simple schema matching approach that searches among several available target schemas for the best match is described in (Shah & Syeda-Mahmood, 2004).)

This online matching scenario has an important advantage. All cases use a particular data model as input, then preparation operations are applied to the data. Each preparation operation produces intermediate data collections. These intermediate models can be included into the search for target models, so that the most suitable *entry point* into a case can be found. Since preparation is actually a method to adapt data representations, it would make no sense to restrict the search for target data models to the initial data that the original KDD processes started out on. The entry point (in the blueprint case's preparation graph) is the data model that the user's local data is mapped to; the data transformations in the blueprint before the entry point are irrelevant for the local data. The most suitable entry point, then, is the (intermediate) data representation that can be mapped best to the local data schema. This "entry point" approach can be parameterised by considering various degrees of the exactness of matching (the schema matching algorithms use a distance measure that can be used to rate the quality of the suggested matching). Another possible extension is based on exploiting the possible connections between a target data model and the operator that uses it as input: for example, some operators require certain input attributes to have a specific conceptual data type. Section 6.3.2 explains how M4 includes constraints to model this. Such constraints could be involved in a decision on suitable entry points, by excluding entry points whose constraints are not fulfilled by the local data.

However, an important restriction of the schema matching approach is that it relies on syntactic clues to discover similarities between the data sets. In contrast, similarities

[7] `http://www.google.com`

in the *way* the data is prepared in the process to be reused, and the way the local data *should be* prepared, cannot be captured. This is discussed further in section 6.6.

The search for suitable cases might also benefit from inspecting the *last* data model in every case: in predictive learning, it includes the learned information (the predicted label) that was decisive for the success of the case, while in both descriptive and predictive settings it represents the form of the data that must be reached in order to be able to benefit from learning. Having found a suitable data format to aim at (not disregarding the business-related information, of course, see above), users can start backwards from there to find suitable preparation methods that can also be applied to their own, local data. The browsable case model on the web platform is very adequate for this type of backward searching, as it displays chunks of steps in the order of the data flow.

A third possibility for indexing the case base is based on the mining algorithms applied in each case, so that all cases that use a particular type of mining algorithm can easily be found. The argumentation is similar to the one for looking at the last data model in a case, as the type of mining algorithm determines technical requirements that the data representation must meet. Currently an index of all operators used in any case is given on the start page of the case base; it allows to find all applications of any operator by pointing from an operator to the list of MiningMart steps that use it. This includes the mining operators, of course.

A fourth index can be set up by applying the subgraph detection algorithm from section 6.5.4, having the extracted common subgraphs inspected and commented on by the administrator(s) of the case base, and using the resulting list of typical KDD tasks as an index. Similarly, the manually created templates (section 6.5.3) can be linked to cases in which they are employed. See also section 6.6.2. The list might be ranked by the number of occurrences of each subgraph or template. Both the subgraphs themselves as well as the cases in which they occur can be of interest. The explicit representation of chunks (chains) in MiningMart helps to administrate subgraphs in the case base.

To sum up, several indexes or views of the case base are possible that reflect different approaches to looking for suitable KDD solutions. Some indexes focus on the application background while others focus on the technical details of a solution. These indexes provide a flexible and powerful tool for case retrieval, which is one of the most important tasks that the case base has been set up for. Once a case is retrieved, it can be reused as discussed in the following section.

6.6. Reuse and adaptation of KDD processes

Section 2.2 has discussed several advantages of the availability of executable models of KDD processes. Applications in distributed and grid or web services based data mining were mentioned in section 6.1.2. In simpler settings, the documentation, storage and retrieval of such processes is no less important, however. Considering that successful mining projects are often integrated into other business processes, for example deployed on a regular basis by nonexpert staff on updated data sets (see also section 2.1.6), documentation and ease of execution are prerequisites to value-adding deployment of KDD within an institution. Wirth et al. (1997) describe wasted efforts in a situation without them. Thus reusability of KDD processes is important even within an organisation or

for the same data miner. The knowledge about successful KDD projects should not only flow from experts to non-experts, or from experts in one domain to those in a different domain, but also from the past to the present, or from experienced staff to new staff in the same organisation.

Section 6.5 has presented a technological framework to publish and reuse KDD models. But which aspects of a KDD process model are reusable, and when? Referring to the six phases of a KDD process defined in the CRISP-DM standard (chapter 2), the more "technical" phases data preparation, mining, and deployment can be modelled in a standard way, as the previous sections have argued, which leverages their reusability (see below). Business understanding seems less amenable to transfers from one KDD project to the next. Yet, the general case information added to the MiningMart web platform (section 6.5.2) serves the purpose of documenting examples of business problems and their solution. Clearly experts of the business at hand will be indispensable, however, when planning a new KDD project. Further, data understanding, as an important CRISP phase, can not be reused across different data sets, but the documentation of the data model in M4 stores all relevant metadata at least for future applications on the same data set. This is very useful even if the new applications are quite different from the original one, since M4 includes expensively computed data characteristics (compare section 3.3.3) and expensively created human comments.

In the remainder of this section, mainly the reusability and adaptability of a preparation process is considered. If the process is applied regularly, for example as a basis of monthly reports, one can expect the underlying data schema not to change from one run to the next, so that the complete process is usable without adaptations. If the process is applied in a new domain or a different institution, adaptations are likely to be necessary. To examine the different types of adaptations, the model case described in chapter 5, which had been implemented in a number of KDD tools for experiments described in chapter 8, has been adapted to a similar but smaller application for this work (in more than one tool). Since the new case was smaller, adaptation consisted a lot of cancelling superfluous attributes and operators in the model case. However, there were also some operators, concepts and attributes that had to be added to the model case.

This adaptation has been made towards a similar case, meaning that its input data came from the same domain as in the reused application, and represented similar things, though it was organised in a slightly different way. However, sometimes one would like to reuse the "essence" of a preparation process on rather different data, for example on data from a different application domain. In such cases, the data sets themselves are not similar, but the way they have been/should be prepared for mining is similar. In other words, certain elements from the local data and the data model to be reused may be recognised, by experienced KDD analysts, as playing or having to play a similar "role" for mining. For example, Morik and Köpcke (2005) describe a knowledge discovery application on insurance data, in which an encoding of features that had until then only been used on text data turned out to enable successful learning. Thus they have transferred the role played by documents, in preparation for text analysis, to insurance contract data. A preparation process from text analysis that computes this encoding can easily be reused on this different kind of data, using the framework of this work, provided that the conceptual data model is connected to the new data in the correct way, so that

the right data columns are encoded. The main problem in this scenario is that the reused data model is likely to use names for attributes and concepts that are based on the original data, so that its names are misleading after the connection to different data has been made. This is not a technical problem, but one that might confuse a user. Thus, a third important operation for reuse is consistent renaming at the conceptual level. Finally, also conceptual data types of attributes may have to be consistently changed.

So the main operations needed for adaptations, or reuse, are the deletion, addition, renaming and retyping of elements of the conceptual level, under the constraint that the models remain consistent. Consistent means, in particular, to change all dependent copies of a changed element. For example, if an attribute of some concept is renamed or deleted, and the original process has applied a SAMPLING operator to this concept, then the output concept of the operator will have the same attribute, which therefore also has to be renamed or deleted. This *propagation* of changes has to be done throughout the graph that models the preparation process. For more details, see below and section 7.1.2.

The remainder of this section discusses more issues in reusing the data model (section 6.6.1) and the process model (section 6.6.2) separately.

6.6.1. Reuse of the data model

Concerning the given data model, only the conceptual part (the higher level) is intended to be reused. Note that this excludes the data characteristics (section 3.3.3), which must be obtained anew from the local data. The conceptual data types used in the given model, on the other hand, are to be reused, since the syntactic validity of the reused process may depend on them. As noted in section 3.3.1, the mapping of conceptual to technical data types can be rather flexible. Only a few combinations, like mapping a continuous attribute to a string-based column, must be excluded.

The terms *target model* for a conceptual data model from the case base (to be reused), and *local model* for the new data schema, are used here like in section 6.5.5. Recall from that section that MiningMart can employ schema matching algorithms that attempt to map the local model to the target, in order to support the reuse of the target model. The schema matching uses the similarities of names and data types. Therefore, schema matching is not useful if the data models represent different application domains, as in the scenario sketched above. In such cases the user must provide an adequate mapping. Whether given by schema matching or by the user, such a mapping may be incomplete in general. Two cases are distinguished.

First. The target model may use attributes or concepts that are not present in the local model. Then it has to be decided whether the role of these elements in the case to be reused has been paramount to the success of the case. This should be easy to find out from the documentation of the case. If the elements can be removed from the given model, this can easily be done provided that the deletions are *propagated* through the process model. That is, all conceptual outputs of processing operators that depend on the reduced parts of the target data model have to be automatically updated to exclude the additional attributes. If, instead, these elements are indispensable due to the important role for mining they have played in the original application, one may be able to generate

attributes that can contribute similar information for mining as the missing attributes. Adding attributes or concepts to the case is discussed below. Only if one cannot extract similar information from the given data, the adaptation is impossible.

Second. The target model may lack attributes or concepts that are present in the local model. In this case, it has to be decided whether the additional local data should be added to the blueprint KDD process, for example as additional attributes in the representation that is used for learning. If yes, additions should be made to the target data model and these additions have to be propagated through the model again. So all conceptual outputs of processing operators that depend on the enhanced parts of the target data model are updated to include the additional attributes. To do this, the operator-specific ways how input changes determine output changes must be known. In M4, the constraints on parameters of an operator (section 6.3.1) model this. A constraint can specify, for example, that the output concept must have the same attributes as the input concept. When a new attribute is added to the input concept, the propagation algorithm can infer that a corresponding attribute must be added to the output concept. The propagation algorithm in MiningMart was implemented by the author of this work, see section 7.1.2.

At this point a crucial advantage of conceptual modelling as discussed in this work shows up again: the model can be updated to represent the new process without the need to execute the process. The syntactic validity of the process can be checked ahead of execution time, which saves the developers a lot of work. This situation can be compared to its extreme opposite: programming all data processing in a language like Perl or SQL, where adaptation to changed circumstances is a lot harder, especially by someone who has not created the original code. The more of the high-level concepts discussed in previous chapters a KDD tool supports explicitly, the easier the adaptation of a process model to changed local data sets or new KDD tasks.

If the local model offers additional tables not represented in the target model, concepts for these additional tables can be created. Then they can be joined to or unified with the mining table (the input for data mining) in the given case, if there are key links between the tables. Perhaps some additional preparation of each concept is necessary, thus the availability of KDD operators can be exploited for the adaptation of the data model (Euler & Scholz, 2004). See also below. Any new attributes that a join introduces into the target data model can again be propagated to later steps. At last, the updated model is executed as usual.

The discussion up to here assumes that the relationship between the additional local attributes or tables and the target data model is semantically transparent to the user. For example, the user must be able to decide whether a local concept and a target should be linked by a separation, a specialisation, a relationship, or not at all. Some research exists that attempts to support users in such tasks by providing more expressive ontological formalisms that describe the data. This is called *ontology integration*, see (Wache et al., 2001; Mena et al., 2000; Akahani et al., 2002; Tan et al., 2003) or (Kalfoglou & Schorlemmer, 2003) and the systems cited there. But these methods also rely on manually or semi-automatically built mappings between different ontologies (Fiedler et al., 2005), even when the ontologies are built using the same formalism and model the same

application domain. Though such manual mappings may not be too difficult to set up with appropriate tools, there are arguably no less user efforts involved than in manual concept matching. Section 3.2.1 explains other reasons why a rich ontology formalism was not considered for MiningMart.

The use of ontologies is often advocated for in data integration in distributed or federated databases, see section 4.1.1. In such databases, a global schema exists that provides a reconciled view of the individual databases (the sources). There are two distinct realisations, the global-as-view scenario in which the global schema is expressed as views on the sources, and the local-as-view scenario in which the sources are formulated as views over the global schema (Lenzerini, 2002). The former approach is much more common in federated databases. The methods developed in this research might be applied here as follows. In MiningMart, one could see the target data model (from a case to be reused) as the global, common data model. This data schema is expressed in M4. The local model corresponds to a source database which has to mapped to the global model when it is to be reused. As section 4.1.1 points out, the creation of such mappings remains a manual, time-consuming task that does not appear to be suitable for the quick reuse of data models.

The local-as-view scenario, applied to the reusing of cases, would mean that the local data sets must be formulated as views over the given M4 target data model. For example, the model case from chapter 5 includes two tables with information about each customer of the telecommunications company. These two tables have been modelled as Concepts `Customers` and `Services` in M4. Assume that another company wants to reuse that case, but has all information about each of their customers in only one table called `CustData`. The aim would then be, in this approach, to express `CustData` as a view on `Customers` and `Services`. In the simplest case this could be done as follows (in SQL): `Create View CustData_View As (Select * from customers_table, services_table Where customers_table.caller = services_table.caller)`. This re-formulation of data sets as views on the mediated schema has to be done by hand. The problem of answering a query on the mediated schema then becomes the problem of answering a query given only a number of views on the original sources; for this problem, a number of approaches (query rewriting algorithms) exist (Halevy, 2001).

In sum, in both data integration scenarios, the mapping from local to global schemas is crucial, but is difficult to construct, and cannot be found automatically (Fiedler et al., 2005). However, if it is at all possible to re-express the local data sets as views on the mediated schema (the target data model), then it is also possible to transform the local data sets so that they match the target schema exactly, using only standard data preparation operators. In other words, in a KDD tool such as MiningMart, this reformulation of data sources can be done at the conceptual level, rather than the technical level as usual in the data integration approaches. In other words, the intelligent but costly (in terms of human efforts) methods such as ontology integration or schema mediation can be circumvented by an intelligent user interface.

In the case of adaptation of a process to similar data (representing the same application domain), this idea can be extended to a partial matching of the local and target model, which is the likeliest outcome of an automated schema matcher. Recall the "entry point" approach suggested in the section on case retrieval (6.5.5). This approach can be

enhanced by some weak reasoning based on the available transformation operators. To illustrate this idea, consider that the schema matcher might rate a possible entry point P lower than another one, P', based on its similarity to the local data, although a simple transformation of the local data would make P the better entry point. Knowledge about how the preparation operators affect their input is stored declaratively anyway in M4 (in the constraints, see section 6.3.2). This knowledge can be exploited by an algorithm that examines a number of possible entry points that the schema matcher delivers, and suggests data transformations to better match the local data to one or more entry points. The reasoning is too weak to consider automatic application of these data transformations, but suggestions to the user can help to quickly adapt the local data. For details, assume that the schema matcher has matched a local concept L to a target concept T, and they have a number of similar attributes (as measured by the matcher) in common. Based on the list of preparation operators in chapter 3, and depending on L's and T's attributes as compared by the schema matcher, the following suggestions could be made to the user:

- If L contains attributes not present in T, the operator Attribute selection (section A.1.1) can be suggested to remove them.
- If L lacks an attribute that T offers, say t, an Attribute derivation (A.5.4) might be suggested to the user. This should, however, only be done if an attribute l with a suitable conceptual data type and a name similar to the name of t is present in L (so that the new attribute l' can be derived from l to match t). Otherwise it is unlikely that the derivation is possible. Name similarity is given by the schema matcher.
- If two attributes l and t of L and T, respectively, match, but l is continuous while t is discrete, a Discretisation (A.5.1) of l can be suggested.
- If two attributes l and t of L and T, respectively, match, but l is discrete while t is continuous, a Value mapping (A.5.3) can be suggested to transform the categorical values into numbers (recall that the technical data type is adjusted automatically).
- If some concepts $L', L'', \ldots$ exist whose features match those of L exactly, a Union (A.2.3) of all these concepts can be suggested. Note that such unions do not help if none of the involved concepts is matched to a concept from the target model.
- Based on foreign key-relationships in the local data, joins can be suggested if their result matches a target concept better than any single local concept.

In sum, because only the concept level information is available in the case reusage scenario, approaches based on richer ontologies or on mediation are waived in favour of user-given mappings, or simple schema matching, extended with a recommendation module that helps the user to find suitable adaptation operators. The main reason is that in many reusage scenarios, the mapping between the target and local model will have to be done based on abstract, mining-related principles that are not reflected directly in the data model. This discussion extends the one in (Euler & Scholz, 2004).

CallerNumber	CalledNumber	Length	Date	Tariff	...
7222277	2777722	194	12-02-2002:18:04:56	11	...
1881181	8118818	82	24-12-2002:11:44:23	2	...
...	...	...	...	...	...

Table 6.1.: A Call Details Table.

6.6.2. Reuse of the process model

Concerning the process model, reusable items range from attribute derivation formulas or parameter settings of algorithms to complete processing chains. (The templates from section 6.5.3 are of course short processing chains that were set up in the first place only to be reused by others.) Again, adding and removing elements from the given case, like single steps or chains of steps (chunks), is not a problem if the supporting system propagates the conceptual changes through the dependent parts of the process model. An important aspect is the robustness of the system that realises the propagation, because propagation of such changes can invalidate the process model. For example, attributes may be deleted that other attributes are derived from. Invalid states and the reasons for them must be highlighted to the user, so that appropriate remedies can be undertaken.

Importantly, operators that "promote" data items to metadata, like the operators that change the representation of the data (section A.3), must be handled: the "signature" of their output, i.e. the list of attributes in their output concept, depends on their input data, which is likely to change now that the process is reused. In MiningMart this means that the parameters of the steps that employ such operators have to be set to new values. This can be done by the system, based on the estimates of data characteristics, in order to support reusing the process model. This is described in more detail in section 7.2.2. The adaptions may entail a change in the number of output attributes, requiring propagation to later steps like in section 6.6.1.

As argued in section 6.5.3, the reuse of *chunks* of operators that solve a particular, typical data processing task is important. For illustration, consider the following example, which demonstrates the use of AGGREGATION (see also the corresponding template in section 6.5.3). In a telecommunications company, a database stores each telephone call individually; see table 6.1. For each call, the column `CallerNumber` contains the caller's telephone number, `CalledNumber` is the telephone number that was called, `Length` is the number of seconds the call took, `Date` gives the exact date and time of the call and `Tariff` gives a code for the tariff used for billing the call.

Assume that for mining, a new attribute is introduced that aggregates the information about individual phone calls into the amount of time *per month* that each client spends calling somebody. One way to compute the desired result is to construct a new attribute `Month` from `Date`, that contains a different value for each month of the time period under consideration (using the operator ATTRIBUTE DERIVATION, section A.5.4). Then the records can be grouped by the values of `Month` and `CallerNumber`, and finally the sum can be computed and inserted into a new table that contains only one record for each caller and month. This is done using AGGREGATION (section A.1.4).

This way of computing a monthly sum is a common subtask that can be reused in

other domains. Assume that a supermarket chain is interested in the monthly sales of their products. In the above table, replace `CallerNumber` by `Product` and `Length` by `Sales`; if the supermarket stores its data in this way, the same procedure as above can be used to compute the monthly sales. Or assume that a road maintenance institution, which maintains a number of weather sensors along their roads, is interested in the amount of rain per month at different sites. Replace `CallerNumber` by `Sensor` and `Length` by `Rain` to get exactly the same problem with the same solution. Appendix B describes many more such solution patterns. As described in section 6.5.4, the recognition of such common subtasks in a collection of process models can be partly automated.

To sum up the discussion on adaptation and reuse, adding or deleting elements to/from a given KDD model, and renaming or retyping, are the central operations for this, and they are easily realised if the system supports propagation of the conceptual changes, for which in turn the conceptual level must be explicitly represented. A second way of support for reuse is the option to use automatically estimated data characteristics to update the operators whose output signature (attributes in the output concept) depends on the input data characteristics. See section 7.2.2. With this kind of support, substantial work efforts during reuse can be saved, even if only some parts of a given KDD process are reused. However, obviously there are situations when adaptation of a given case is not a suitable option. This can be true when a change of the mining task would be needed (e.g. from classification to concept description), as this would usually require rather different data representations as input for mining. The case descriptions explained in section 6.5.2 help users to avoid attempting such difficult adaptations.

6.7. Summary

The need for public environments that enable the modelling and distribution of KDD processes has long been recognised in the literature. MiningMart is based on a public meta model that allows to model the conceptual level of KDD processes, as discussed in chapters 3 and 4, as well as the necessary technical level notions. Section 6.3 has described this meta model. It is the basis for the web repository which has been presented in section 6.5. Providing this environment has enabled the formalisation of approved solutions to common preparation problems; these solutions are listed in appendix B. Reusing previously developed process models (solutions) on different data has been examined in detail in section 6.6. Schema matching and propagation of changes have been identified as two important supportive features that the environment should offer. Chapter 7 describes technical solutions used in the MiningMart system for these and other issues.

7. Implementing the Conceptual Level

This chapter continues the discussion of the MiningMart framework and system as introduced in chapter 6. While the focus in that chapter was on the meta model, the web repository, and aspects of reuse, in the following a more dynamic view of the system is given. In particular, the implementations contributed by the author of this thesis are explained. MiningMart's implementation follows a metadata-driven approach: declarative, static aspects of the meta model determine how the system instantiates and uses elements of the conceptual data and process model. Compare figure 6.3 on page 101. Thus the implementation is *generic*, in the sense that changing or extending the meta model appropriately automatically changes or extends the behaviour of the system, without a need to change the implementation.

Section 7.1 explains how the output of an operator (at the conceptual level) can be created based on the specification of the operator, and how changes to such elements by the user are propagated through the whole application model. It also discusses the implementation of the estimations of characteristics that are given for each operator in appendix A, and explains how schema matching was realised to help in connecting conceptual data elements to actual data sets.

Section 7.2 then presents details on the implementation of some MiningMart operators created by this author. The importance of these operators had been recognised when developing the system of preparation operators described in chapter 4 and when implementing the model application from chapter 5.

Section 7.3 discusses some issues of data storage and caching that complement the view-based approach taken by the MiningMart compiler. Finally, section 7.4 briefly introduces the main aspects of the implementation of the user interface.

7.1. The concept editor

Recall from section 4.6 the duality of the data- and process-oriented views on the KDD process, which was illustrated in chapter 5. MiningMart is the first KDD tool that implements the data-oriented view. Whenever a step (employing a processing operator) is added to the preparation graph in the process view, the user must specify its input and output parameters. From these parameters, elements of the data-oriented view (the concepts and links between them) are automatically generated, as explained in section 7.1.1. If the output of a step exists already but the parameters are changed by the user, parts of the output may have to be changed, too, and this may have effects on later, dependent steps. Propagation of such changes was already discussed in section 6.6.1; section 7.1.2 discusses its implementation in MiningMart. Section 7.1.3 explains the implementation of the metadata inferences, or data characteristics estimations, as introduced in section 3.3.3. Finally section 7.1.4 presents the use of schema matching in MiningMart.

7.1.1. Automatic creation of conceptual-level data elements

Overview

As was explained in section 6.3, MiningMart is largely based on a declarative meta model called M4. M4 provides means to store KDD process and data models. Recall that M4 can be divided into a static part, which stores information about the available operators, and a dynamic part, which stores elements of the KDD models that users edit. The MiningMart system is designed such as to allow the extension of the static part easily without having to change the implementation. In order to add an operator to MiningMart, its input and output parameter specifications, as well as the way output is created from given input parameters, have to be described using the expressions available in the static part of M4. Then the system can automatically instantiate the operator at runtime.

In the following, these mechanisms are explained in detail, in particular the M4 elements that allow to create output concepts of steps automatically, and thus link the two dual views discussed in section 4.6. Recall that an instance of the M4 type Step represents the application of an operator at a particular point in the preparation graph. Which parameters an operator has is given in the static M4 type Op_Param (compare figure 6.5 on page 103). For example, the fact that a certain discretisation operator expects a single, non-optional, input parameter (giving the number of intervals into which the input attribute is to discretise) is stored as one M4 object of the type Op_Param. The input attribute itself, and the name of the output attribute that the operator will create, are two further parameters of this operator. As already discussed in section 6.3.2, the M4 type Constraint is used to further specify such parameters, for example to indicate their data type, or that they cannot be negative, and similar constraints. These constraints help to develop a process model without executing it, since they can be used to check the syntactic validity of all user-specified parameters.

The M4 Constraints are also crucial for the automatic creation of output elements of the data model (at the conceptual level), because they are used to specify how to do this for each operator. Thus they serve the double purpose of specifying possible instances for input and output parameters, and dependencies between them, for both the user and the system. Table 7.1 on page 162 lists all types of constraints used for the latter purpose, the creation of output data model-elements by the system. Each constraint has two "slots", or parameters, that specify to which step parameters it applies. Most of the constraints from table 7.1 were added by the author when implementing the mechanism for output creation; the M4 constraint formalism itself has been developed by Scholz (2002). The constraints from table 7.1 will be explained in the following, including examples. Afterwards, the creation of semantic links between concepts is explained.

To understand the output creation, a design decision that was taken for MiningMart must be mentioned. Chapter 4 explains the growing web of concepts in the data-oriented view, as each operator's output concept is added to the representation of the initial data. In MiningMart there is one exception to the rule that each operator produces an output concept: the operators based on ATTRIBUTE DERIVATION (the feature construction operators of section A.5) only produce an output *attribute* that is added to the input concept. This has the advantage that the resulting concept web is less complex, since there are

fewer concepts. The disadvantage is that it may change the semantics of concept links. For example, if a concept C is the separation of another concept D because, say, a ROW SELECTION created it from D, then adding an attribute to C changes the separation to a specialisation. One might choose to automatically update these links in the concept web; but it may also make sense to leave the original links in place, since they reflect the creation of the concepts. MiningMart currently takes the latter approach.

The constraints for creating output elements

Input parameters of a Step refer to M4 objects that exist already at the time of creating the step. For example, the input concept of the step represents the data set that it is applied to, and thus it must exist already in the data model. Thus the user should only be able to use existing objects for input parameters. In fact, the sets of available objects for an input parameter can be further constrained:

For input *concepts*, only concepts representing initial data sets, plus all concepts created by steps that precede the current step in the preparation graph, can be used. When the user wants to specify an input concept, the GUI therefore calls a method that provides this set.

Input *attributes* must always belong to (one of) the input concept(s); a Constraint specifies which input concept it is for each operator and input attribute. It can also be a concept that is not an input parameter itself, but is attached to an input relationship. The constraints IN_RELFROM and IN_RELTO serve this purpose. The Java class for Steps provides a method that tells the GUI which input concepts an attribute may be selected from. If the input concept(s) of the step have not been set yet at the time of calling this method, an error message is produced.

Output parameters are treated rather differently. Their objects do not exist yet at the time the user creates a step, but are created when the user tells the GUI to save the current step parameter settings. The user only provides the names for the output objects.

The following paragraphs explain how objects for output parameters are created, based on the constraints. The creation of the output *concept*, if there is any, is done first. In the simplest case, its attributes can be simply copied from the single input concept; this is the case if a Constraint of type SAME_FEAT holds for the operator of the current step (see table 7.1). An example for an operator that uses this constraint is ROW SELECTION. Similarly, the constraints FEAT_RFR and FEAT_RTO specify that the attributes of the output concept should be copied from one of the two concepts attached to an input relationship. If, instead, there is a constraint of type ALL_EXCEPT, then all attributes of the input concept, except those specified by an input attribute parameter given by the constraint, are copied to the output concept. The constraint types SAME_FEAT, FEAT_RFR, FEAT_RTO and ALL_EXCEPT are mutually exclusive.

In many cases, additional constraints specify output attributes to be added to those created based on an ALL_EXCEPT constraint. Such output attributes may be based on input attributes, as is the case if any of the constraints RENAME_OUT, CR_SUFFIX, or CREATE_BY hold. For example, a constraint CR_SUFFIX holds for the MiningMart version of AGGREGATION, and specifies that for those attributes of the input concept whose minimum value will be available in the output concept, an attribute in the output

concept must be created that takes the same name but with a suffix "_MIN" (the suffix is specified by the constraint). A similar constraint holds for output attributes with the maximum, average, and so on, values. The CREATE_BY constraint is used for PIVOTISATION, for example, and allows to create output attributes based on the name of an input attribute (the index attribute, see section A.3.2), with added suffixes based on values given by another parameter. Note that there can be more than one such input attribute, in which case combinations of single values given by the other parameter are used to create suffixes and thus output attributes; this is used to implement the generalised, n-fold pivotisation, and is described in more detail in section 7.2.2. For output attributes constructed in this way, the constraint OUT_TYPE may be used to specify a fixed conceptual data type; if no such constraint exists, the data type from the input attribute is copied. The RENAME_OUT constraint is used to specify that certain input attributes should be used as a template for output attributes, but the output attributes should have different names; the names are given by a parameter that the constraint specifies. This constraint is used in the JOIN operator to provide a way of dealing with like-named attributes in the concepts that are joined. (An additional constraint of type NO_COMMON, not shown in table 7.1, ensures that an exception is thrown if concepts to be joined have like-name attributes and the step does not use the provided parameters to resolve the conflict.)

The MATCHBYCON constraint is also used by JOIN in order to copy only one set of the joining key attributes into the output concept (these attributes are specified by a parameter of JOIN, and the constraint refers to this parameter).

As mentioned above, in MiningMart not all operators create an output concept, but some only add an output *attribute* to their input concept. This is the case if the constraint type IN is used for an input concept and an output attribute. The name of the output attribute is given as the parameter, and its conceptual data type is specified by a TYPE or SAME_TYPE constraint.

While most MiningMart operators create a concept or an attribute, some create a relationship as their only conceptual level output (this is a difference to creating relationships as a by-product, namely as semantic links between the output and input concepts of an operator). Such operators can be used to restore relationships between intermediate concepts, since valid foreign key links between the initial data sets may not be valid after applying data transformations. For example, when a ROW SELECTION has been applied to the many-side of a many-to-one relationship, the output concept is still in a many-to-one relationship to the original one-side, but a corresponding link is not automatically created in the data model since too many links would clutter the web of concepts then. But if the link is desired, then a MiningMart operator that creates a one-to-many relationship between its two input concepts can be used to create it. The operator produces an error at execution time if the relationship is not valid in the actual data (e.g. if there are entities on the many-side for which there is no correspondent on the one-side). The last set of constraints in table 7.1 is used to specify which parameters of such operators give the concepts and keys involved in the relationship (the two concepts involved in a relationship are called the From-concept and the To-concept in MiningMart). Like other elements of the conceptual-level data model, relationships (created by operators directly or as a link between an operator's output and the other data sets) are realised in

SQL statements on the technical level by the MiningMart compiler when the preparation process is executed. For relationships, these SQL statements create the corresponding primary and foreign key constraints in the underlying database.

The assertions for creating links between concepts

Finally, the automatic creation of semantic links between input and output concepts of an operator is described in the following. Recall that there are three types of links that show how concepts are created from each other, providing a structured view on the concept web. The type and direction of each link is specified by objects of the M4 type Assertion for each operator: the presence and the validity of the given link are a post-condition of applying the operator. Like constraints, assertions have two slots or parameters that determine what step parameters they refer to. Both for *separations* and *specialisations*, a specific type of assertion exists. The order in which the input and output concept are specified in the assertion determines the direction of the link. The creation of a one-to-many *relationship* is more complex since the key attributes that constitute the link must be specified. Therefore a particular type of assertion, REL_N_1, indicates the presence and direction of the relationship, and two other assertions, REL_N_K and REL_1_K, determine which input or output attribute parameter gives the attributes that function as the keys on the many-side or the one-side, respectively, of the relationship. Many-to-many relationships are never created in this assertion-based way, since no operator from appendix A produces such links.

An example

As a simple example, consider the MiningMart operator RemoveFeatures, which realises the variant of Attribute selection in which the user selects a number of attributes (features) to be removed from the input (in a different variant, the attributes to retain are selected). The following items of information are each given by one object of the type Constraint in the static part of M4:

- There must be exactly one input concept.
- There must be exactly one output concept.
- There must be a parameter "FeaturesToRemove" which refers to a non-empty list of objects of the M4 type Feature.
- The list of features given by the previous parameter must be present in the input concept. This is specified by an IN constraint.
- All features from the input concept, except those given by "FeaturesToRemove", must be created as a copy in the output concept. This is specified by an ALL_EXCEPT constraint.
- The input concept is a specialisation of the output concept.

When creating an object of type Step that uses this operator, users select an input concept from those that are available at this point in the preparation graph. Afterwards

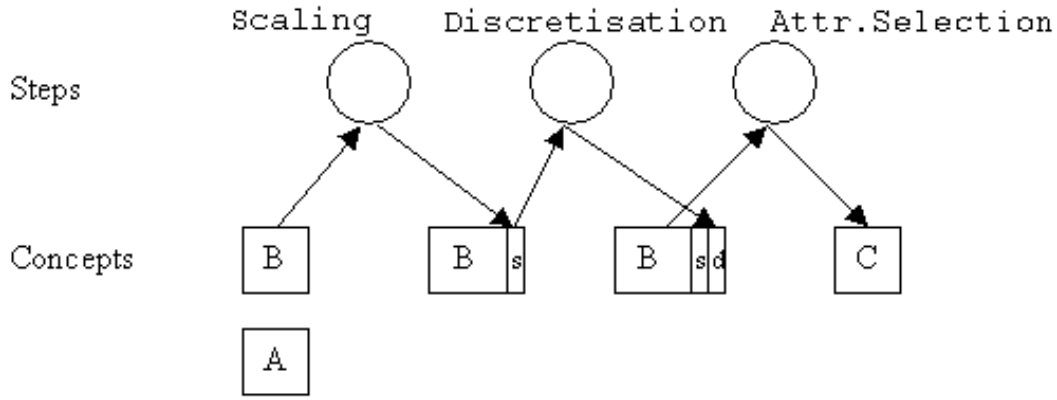

Figure 7.1.: Illustration of the preparation scenario discussed in the text. Arrows pointing upwards mean "used as input by". Arrows pointing downwards mean "creates". The three steps must be executed from left to right, since each step's input is created by the previous step.

they select the "FeaturesToRemove" from that input concept; there must be at least one feature. Finally they give a name for the new output concept. The system then uses the last two constraints mentioned above to create the output concept, including its attributes, and link it to the input concept.

7.1.2. Propagation of data model changes

Motivation

The parameter settings of any step can be edited by the user at any time. If the step has successor steps in the preparation graph, changes to its output may affect one or more of the successors, too. It is central to supporting the conceptual level that these changes to its elements are performed automatically by the system. To see what is involved in the problem, a motivating example is discussed.

Consider a preparation graph in which the first step is a SCALING application, the second step applies DISCRETISATION to the scaled attribute, and the third step involves an ATTRIBUTE SELECTION. The scenario is depicted in figure 7.1. Each step takes the output of the previous step as input, but recall that SCALING and DISCRETISATION only add a new attribute to their input concept (B), so that the third step is the only one that produces a new output concept (C). The first step creates the attribute s while the second creates d; both steps add their output attribute to B.

Now suppose the user changes the input concept of the first step from B to A, for example because they decided to insert an additional preparation step (which creates A). The system now has to delete s from B (thus "cleaning" the old input object), and must create it instead in A. The second step, however, used s as one of its input parameters (the scaled attribute was the one to be discretised). It is clear in this case that the second step is now also supposed to be applied to the new input concept A, otherwise one of its input parameters would not be available. This change of parameters of the second step is

determined by the changes to the first step, and can be done automatically. If the second step had been used to discretise not the scaled attribute s, but some other attribute, then changing the input concept of the first step (from B to A) would not necessarily mean that the input concept of the second step should also be changed. However, any of these changes can lead to a change of C, since its attributes are copied from B (respectively, A). This change can again be performed automatically. If there are further steps that depend on the third step (i.e. take its output concept C as input), their output objects may also require adaptation.

It may happen that a step becomes invalid during the course of these adaptations, meaning that one or more of its input objects is nonexistent. The simplest scenario is that the user wants to delete an attribute from a concept, but the attribute (or a copy of it in a dependent concept) is used as an input parameter of some step. Another example is that the selection of attributes made by an ATTRIBUTE SELECTION operator changes, and some attribute that is no longer in the selection is supposed to be used by a later step.

Scenarios such as these lead to the following requirements that the system must meet in order to allow safe edits by the user:

- deletion of attributes from a concept must be propagated to (modified) copies of that concept;
- adding new attributes, and renaming of attributes, must be propagated in the same way;
- when input parameters are changed, the objects previously used for them may have to be "cleaned";
- input parameters of following steps may have to be adapted;
- before steps become invalid, the user should confirm the action;
- when the input concept of a step that does not create an output concept changes, and following steps use the same input concept, the user must decide whether the following steps should also change their input concept (which is usually what is desired, but this cannot be presupposed safely).

Overview of the propagation scheme

Propagation is done based on the process-oriented view, that is, the graph of preparation steps; the reason why the data-oriented view cannot be used, despite the fact that it models dependencies between concepts directly, is that the local changes depend on the parameters of the operators.

The propagation involves a graph traversal. The two classic schemes of traversal, depth-first (DFS) and breadth-first (BFS), cannot be used here because of the requirement that any node can only be processed after all its predecessors have been processed. Figure 7.2 illustrates this. Fulfilling this requirement ensures that all updates of previous steps are accounted for when the current step is updated. Breadth-first search can be organised using a queue of nodes which is filled level-wise with the nodes of each search level.

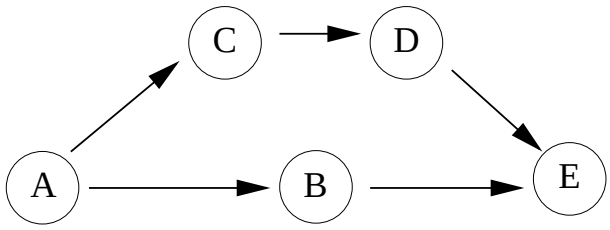

Figure 7.2.: Both standard graph traversal schemes, breadth-first and depth-first search, could result in node E being processed before D has been processed. This is desired, for the propagation task in this section, if the traversal starts at B (since D should not be visited at all then) but not if it starts at A.

The following modification is needed to fulfil the requirement. By noting for each node whether it has been processed, the predecessors of any current node can be checked as to whether they have all been processed; if not, the node can simply be ignored, since it will be reached again on a different path. In figure 7.2, assuming that propagation starts at node A, node E can be ignored when it is visited on the path from B but has not been visited yet from D, as the visit from D will follow later. In this way the node is only processed when it is visited for the last time. However, if E is visited from B when B was the starting point of the propagation, then E should be processed without visiting D at all. Yet at node E, no local information is available to decide between the two possibilities. To solve this, only those predecessors of a node are considered (when deciding whether to ignore it) that are "relevant" for the current traversal, meaning they can be reached from the starting point of the traversal. The set of nodes reachable from the starting point can be found previous to propagation, by either of the two standard schemes.

Despite these modifications, the propagation of changes through the preparation graph should proceed mainly in a breadth-first manner, so that it can stop at the first level where no changes to the output concepts of any step are performed. The levels of this modified breadth-first search are to be defined such that each level consists only of concept-creating steps; any intermediate steps that only add output attributes to their input concepts are also dealt with in passing. The reason for this modification of the search scheme is that the concept-creating steps have a property that the "attribute-creating" steps do not have, which is that changes to later steps will not occur if changes to the output of the current step have not occurred. For example, suppose a concept is modified by deleting one of its attributes. If the concept is used as input by a step that adds some other *attribute* to it, no changes to this step's output are done, but a later step that creates a copy of the concept is affected (the copy of the deleted attribute must be deleted). In contrast, if an output *concept* of some step remains unchanged during the propagation, it is safe to conclude that following steps and parameters also remain unchanged, so the propagation can stop on this path.

Realisation in MiningMart

In MiningMart, whenever the user performs parameter changes in the GUI, first the output of that particular step is updated if required. Second, previous input of that step is cleaned if necessary, and third, propagation of changes to successors is started. The method `updateOutput(Collection names)` of the class representing Step objects realises the first task; it is called by the GUI with the names of the new output objects. It shares most of its code with the constraint-based method for output creation explained in section 7.1.1. The big advantage of this is that the propagation algorithm does not have to be changed when any operators are added to the system, or change their specification. Secondly, the method `handleOldInput()` of the same class is called. It only applies to input concepts that have become replaced by a different concept. It has two tasks: one is removing output attributes that have become invalid from such replaced input concepts. The other is to check if any successors of the current step also use the replaced input concept. For this, any path starting from the current step is only followed until the first output concept is created; steps that occur yet later can still use the replaced input concept without problems. If there are any successor steps with the replaced input concept, the user is asked if these steps should change their input concept, too, like the current step. If yes, the replaced input concept is also cleaned from output by these steps, and the output of these steps is instead added to the new input concept.

Thirdly, propagation can be started. The method `propagateOutputChanges()` of the class representing steps does this. It uses the graph traversal scheme explained above. The method `adaptOutputToChangedInput()`, which again shares code with the output creation as explained in the previous section, is called for every step on the current BFS level; it returns a boolean flag indicating whether any output has changed. This flag can be used to stop the search if no changes have occurred on the current level.

The level-wise search is necessary since the preparation graph cannot be assumed to be a tree. However, this search scheme is slightly complicated by the fact that changes to one level can affect not only the next level but also arbitrary higher levels, since *any* successor step might explicitly access the output of the current step through one of its input parameters. For example, renaming an attribute during a propagation may entail updates of some step that happens to use this attribute as an input parameter, even if that step is located elsewhere in the graph. The old attribute must be known when the later step that uses it is dealt with. Such dependencies are *long* dependencies in the process view, with any number of intermediate steps. Though some such dependencies may appear as *direct* dependencies in the data view, the direct dependencies cannot be exploited by the algorithm, since the intermediate steps in the long dependency may affect the data representations, too. To understand how the propagation algorithm deals with such long dependencies, recall from section 6.3.2 that the M4 type Parameter stores the links between a step and the M4 objects that it uses as parameters. The M4 interface module allows to retrieve all such links an M4 object is involved in from that object. Thus, whenever the algorithm modifies an object used by the current step, it can follow the links from that object to any later steps that also use it. However, those later steps should not be modified before they are visited in the breadth-first scheme, because the need for further changes might arise during the search (after the current step has been dealt with). Therefore the algorithm stores the links in a global data structure that maps

each link to the previous and the new name of the modified parameter object. When the later steps are visited during the search, the global map can be used to find out which of their input objects has been modified, and how.

As an example consider the scenario discussed above, depicted in figure 7.1, in which a SCALING step creates an output attribute that is used as input parameter by a following DISCRETISATION step, but at some point in time the first step changes its input concept. The attribute s that the first step creates must be deleted from concept B. Instead, a new attribute object is created in the new input concept A, but its name remains the same. Thus when the algorithm handles the second step, it can use the old name, s, stored in the global map, to find out which of the attributes in the new input concept (A) of that step is the one that the step had used for discretisation. This information would be lost without the global map. Similarly, had the user not changed the input concept of the first step from B to A, but just decided to change the name of the attribute it creates, say from s to t, then the new name t would also be stored in the global map, allowing the algorithm to update the second step accordingly. This becomes relevant when there are other steps between the first and second one, so that the second step operates on a *copy* of the attribute created by the first one.

This algorithm is important for making the conceptual level easily and robustly reusable on changed data schemas, as discussed in section 6.6.1. Figure 7.3 shows the algorithm in pseudo code, as realised in the above-mentioned method `propagateOutputChanges()`. The algorithm uses a queue to organise a traversal that is similar to the breadth-first scheme. As motivated above, a step is ignored when one of its relevant predecessors has not been processed yet (lines 10 to 12; the set of "relevant" steps is computed in line 3). The boolean variable f is used to indicate whether any step of the current BFS level l has changed its output; if not, the algorithm stops (line 16). The changes to the current step are made in line 20, so that f can be set to `true` (in line 22) if any changes have occurred. Note that "concept successors" in line 23 includes all steps on any path that (i) starts from the current step, (ii) ends in a step that produces an output concept, and (iii) has no output concept-producing steps other than this last step. This has been explained above. Restoring inputs of the current step from the global map (line 19) only concerns those inputs for which there is an entry in the global map that matches the current step; such entries are added to the map in line 20: if any long dependencies from the output of the current step to later steps, as explained above, are discovered, the corresponding links to the dependent steps are stored in the map. When the propagation ends, the global map is emptied (line 24) so that the next call to the algorithm starts with a cleaned map.

7.1.3. Estimation of data characteristics

This section presents the MiningMart implementation of the estimation mechanisms that section 3.3.3 introduces and chapter 4 specifies for each preparation operator. Why it is useful to provide (estimated) data characteristics at the time of creating a KDD process model is explained in section 3.3.3. The list of which characteristics to estimate is also given there. The estimations are based on actual characteristics computed from the initial data sets.

Algorithm: Propagation of conceptual data changes

Input: A preparation graph modelled in M4, and an object S_0 of type Step whose parameters have changed
Output: A possibly modified version of the preparation graph

Initialisations:
1. Set the BFS level of all steps in the preparation graph to 0;
2. Initialise Q to be an empty queue;
3. Find R, the set of "relevant" steps that are reachable from S_0, by a DFS from S_0;
4. Set the BFS level of the start step S_0 to 1;
5. Enqueue the start step S_0 into Q;
6. Set the boolean flag f to `false`;
7. Set the current search level l to 1;

Graph traversal:
8. While Q is not empty:
9. Dequeue Q into step S;
Check if all predecessors of current step S have been processed:
10. If any predecessor of S is in R, and has BFS level 0 or is in Q:
11. Set the BFS level of S to 0;
12. Continue at line 8;
Check if new BFS level has been reached:
13. If BFS level of S is strictly greater than l:
14. Set l to BFS level of S;
15. If f is `false` and global map is empty:
16. return;
17. Else: set f to `false`;
Update the input parameters of S:
18. If global map contains entries for S:
19. restore input of S from global map;
Update the output parameters of S:
20. Adapt output of S to (possibly) changed input,
adding any links to dependent steps to the map;
Check if any changes are made at current BFS level:
21. If output of S has changed:
22. Set f to `true`;
Continue the search with the successors of S:
23. Enqueue all "concept successors" of S whose BFS level is 0 into Q,
setting their new BFS level to $l + 1$;

Final clean-up:
24. Empty the global map;

Figure 7.3.: An algorithm to propagate changes to the current parameter settings of a Step object to all dependent Step objects. *Comments* are in italics. See text for further explanations.

Overview

In MiningMart, (estimated) characteristics (also called statistics) can be displayed in the concept editor for any concept, but also in the process editor when the input of an operator is specified, because some operator parameters are based on such characteristics (compare criterion 20 on page 232 and section 7.2.2). The estimations and inferences are done on the fly, whenever the user wants to display the characteristics. MiningMart can store *actual* data characteristics of any data set, to avoid their re-computation, but M4 currently does not provide means to store the inferred or estimated characteristics; since the latter can be computed in linear time in the number of operators, concepts and attributes of a process model, on the fly computation suffices. If the actual statistics have already been computed, or stored in M4, for a concept whose estimated statistics a user wants to display, these actual statistics are shown. MiningMart provides the option of declaring the inferred/estimated statistics as actual statistics, so that they can be stored. In this way users can avoid expensive computations if their background knowledge tells them that the inferred or estimated statistics are accurate enough for the current purpose. Also, any estimated values of data characteristics can be *edited* by the user, and are then kept in main memory as long as the user's MiningMart session lasts (and as long as the concept for which they hold does not change). This enables to integrate background knowledge when the inferences or estimations do not provide enough information, and conforms to criterion 19 (see page 232).

While estimated statistics are presented to the user in a way that is clearly different from actual statistics, the quality of the estimations is not easy to judge for users. Future work could enhance the estimation framework presented here by methods that differentiate results of inferences, or safe knowledge, from estimations or unsafe knowledge. However, it should be noted that even using the rather optimistic estimation methods presented here, only little information about the characteristics can be expected at the end of long chains of preparation operators. Safe inferences would render even less information, so that a distinction between these types of knowledge quality may not be very useful in practice.

To get a feeling for the usefulness of MiningMart's estimations, the model application from chapter 5 can be considered. This application is characterised by early joins of detailed information with selected customers in every chunk. The output size of these joins cannot be estimated since they are not based on valid foreign key links (the keys are not created before execution time). However, for all attributes that are not used as joining keys, the list of distinct values is retained. Thus there are rather useful estimated characteristics available in every chain – until the first aggregation operator is applied, which occurs several times in the main chunks, but not early on in each chunk (pivotisation usually involves aggregation, too). Aggregation does not allow to estimate much information about its output data characteristics. However, this still means that useful (if not always accurate) data characteristics of the input of a step can be displayed for 51 of the less than 100 steps in this application, so a substantial part of the application could have been developed easily without actually performing the time-consuming data processing. To perform at least partial data processing when developing such a large application, for testing purposes, is unavoidable, though.

Operators in a system that uses characteristics estimation must be robust against input

specifications that do not accurately reflect the technical level, because misestimations may occur at the conceptual level. The MiningMart operators are robust in this sense simply by the fact that they are implemented using standard SQL views: SQL queries may be invalid if they refer to non-existing tables or columns, but the invalidities that can arise from misestimations only concern values of a column. For example, the operator VALUE MAPPING maps input values of a particular attribute to new values in the output. Estimations of value lists help the user to specify such a mapping. At the technical level the mapping is realised by the SQL `CASE WHEN ... THEN` statement. Thus, if the specified mapping uses input values that actually do not exist on the technical level, the SQL statement simply does not apply. Conversely, if there are input values at the technical level that do not occur in the user-specified mapping, they are ignored or mapped to a default value, since it may well be that the user only wanted a few values mapped. See also section 7.2 where the robustness of other operators is explained.

Inferences and estimations of the data characteristics are based on the operator specifications. Therefore, following the general MiningMart approach of using declaratively specified knowledge to drive the system, the kinds of inferences and estimations that are possible for each operator are stored in the static part of M4, and are interpreted by the system at runtime. Again this enables the simple extension of the system by new operators without changing the implementation. The M4 type Assertion is used here. Table 7.2 on page 163 lists the various types of assertions that can be specified for an operator; they are briefly explained below. A given operator uses a combination of these types.

Estimated statistics always concern a particular concept. When they are to be displayed, the system decides whether the concept represents an initial data set for which actual statistics are available or can be computed. If yes, they are displayed. If no, the step that creates this concept is found, and estimations are done by modifying the input estimation(s) that hold for the input concept(s) of this step according to this step's assertions; the input estimation(s) are computed recursively in the same way. Some input statistics can simply be copied. When a step only adds an attribute to a concept, the estimations for that attribute are simply added to the statistics. When a step has more than one input concept (this concerns the JOIN and UNION operators), the estimations are merged according to particular assertions that apply to several inputs. There is one global characteristic per concept, the number of entities in it; the other estimations concern single attributes. To create its output characteristics, a step copies its input estimations (for those attributes that are present in the output concept), and then modifies them according to the assertions, so that assertions are only needed where copying input estimations is not appropriate.

Assertions for estimating data characteristics

The remainder of this subsection briefly explains each assertion type related to statistics estimation and gives examples for their use in operator specifications, so that the MiningMart implementation of this functionality becomes more transparent. Refer to table 7.2 on page 163 for an overview of these assertions. Recall that an assertion comes with two "slots" or parameters that determine the step parameters it applies to. These slots will be dealt with implicitly in the explanations below.

The simplest assertion is NO_CHANGE, which says that for any attribute in the out-

put concept, its estimated statistics can be copied from its corresponding input attribute, and the number of entities can also be copied. This assertion is used in the attribute selection and materialisation operators, and also some operators whose output concept is largely a copy of the input, but has additional attributes; additional assertions are used for these extra attributes.

A number of assertions concern the estimation of the output size. SZ_ADD is an assertion that is used for UNION, specifying that the sizes of the input concepts are added to get the output estimation. SZ_BY_VAL gives the output size directly, by referring to a parameter whose value in an instantiated step gives either the size or the fraction of the input size. This is used for the sampling operator. SZ_BY_VL can be used by aggregation operators to compute the output size from the numbers of distinct values in the grouping attributes. SZ_MIN_MV subtracts the number of missing values of a particular input attribute from the input size, useful for the operator that deletes entities that have missing values in that attribute. SZ_DIV_BY divides the input size by the value of some parameter; this assertion holds for the WINDOWING (divide by window width) and SEGMENTATION (divide by number of segments) operators. SZ_MULT_NO multiplies the input size by some factor given as the number of attributes in the parameter specified by this assertion. It is only used for reverse pivotisation: the number of attributes to be "folded" into one determines the integer factor by which the data set grows (see section A.3.2). Finally, SZ_BY_REL is used for JOIN; see section 3.3.3 for an explanation of how the output size of joins can be determined if the input concepts are linked by a relationship. Note also that MiningMart provides operators that allow the creation of a relationship, on both levels, between two data sets, so that relationships can be made available to join operations whenever they are needed (and valid).

The other estimations concern the minimum and maximum bounds, list of distinct values, number of missing values, and value frequencies, of attributes. For continuous attributes, the list of distinct values gives interval means instead; the interval bounds are chosen so that there are 10 bounds (thus 11 intervals) in the range of values the attribute takes. The frequency of a value is then the number of values within that interval.

Some assertions simply state that these attribute-specific characteristics can be copied from a certain input attribute (MM_FROM, VL_FROM, VF_FROM and MV_FROM); they are mainly used for attribute-creating operators that do not create their own output concept. A similar assertion states to copy these properties from the input for all attributes in the output concept (MM_UNCH, VL_UNCH). Some operators can directly specify minimum or maximum bounds of their output attribute, like SCALING, so they can use MIN_FROM or MAX_FROM to make the system exploit this. The VL_COMB and MM_COMB assertions are used for UNION, where output attributes have combined, or merged, value lists and bounds from their corresponding input attributes.

Adding a value to an input value list can be done with VL_ADD: for example, the operator that replaces missing values by a default value uses this assertion. For the value mapping operators, the new value list is known from the specified mapping; the VL_BY_PAR assertion can be used here. The VL_BY_LIST assertion is very similar, but is used when there are several entries with output values in a given value parameter object (see section 7.2.2 for an example). Finally, the VL_BY_SYM assertion states that a particular symbol (which is globally fixed in MiningMart) is used, with number suffixes,

for the output values. Such default symbols are used for value mapping or discretisation operators when the user does not specify the new discrete output values. The second "slot" of this assertion provides the number of output symbols (or the step parameter that gives this number); by convention the number suffixes start with 1, so this determines the output value list.

Some operators even allow estimations of the frequencies of the values in the output, provided that these frequencies are given in the input. The VF_ADD assertion is used for UNION, and says to add the frequencies of a particular value from all corresponding attributes in the input concepts. The VF_REPL_MV says to add the number of missing values from the input attribute to the frequency of the value determined by the second slot of this assertion. This is used for the operator that replaces the missing values in the input by a default value: obviously the frequency of the default value increases by the previous number of missing values.

Sometimes the selectivity of the operator application, meaning the ratio of output size to input size, can be optimistically assumed to apply to value frequencies and numbers of missing values. Such operators can use the VF_BY_SEL and MV_BY_SEL assertions to state this. In MiningMart this concerns the ROW SELECTION and SEGMENTATION operators.

The VF_BY_AGG assertion is used for the grouping attributes of aggregation operators; their frequencies are determined by the possible combinations of grouping values. For example, if there is only one grouping attribute, each of its values occurs with frequency 1 in the output. Finally, the VF_MULT_NO assertion is based on similar reasoning as SZ_MULT_NO, and is applied by the same operator (reverse pivotisation).

The missing values estimation assertions parallel those already explained.

The last estimation assertion is ES_SELECT. It tells MiningMart to apply some operator-specific reasoning to estimate output characteristics of ROW SELECTION. As discussed in section 3.3.3, rather complex reasoning can be applied for selection operators. Currently MiningMart supports a simple histogram-based method, since the input value list and value frequencies together provide the needed histogram. More complex methods can be added here at any time, but this will not change the system of estimation assertions. This particular assertion is obviously of a different nature than the others, as it does not specify directly how to do inferences or estimations based on input characteristics, but is operator-specific. While this is a slight violation of MiningMart's design principles, following these principles here would have meant to design rather complex assertions that specify the histogram-based method of selectivity estimation, which would be no less operator-specific but probably somewhat over-engineered.

7.1.4. Schema matching between the two levels

When developing a KDD application from scratch in MiningMart, the first step is to model the initial data sets. The system can create concepts directly from database tables or views. The results can and should be edited by the user by giving explanatory names to the concept and its attributes, or by removing superfluous attributes from the concept. This functionality, creating concepts directly from database tables, depends on the information about the table or view that is stored in the system tables of the underlying DBMS; this is one point where MiningMart cannot rely on standard SQL statements,

since such meta queries are not standardised in SQL. Within the MiningMart code, such meta queries are done through an abstract Java class which is implemented differently for different DBMS systems; the author has implemented it for the PostgreSQL[1] and MySQL[2] database systems.

In contrast, when an existing KDD model is to be reused, an existing conceptual data model has to be mapped to existing technical-level data sets, as discussed in section 6.6.1. MiningMart can support finding this mapping by employing simple schema matching algorithms, which is documented in this section. A preliminary study by Wagner (2005) has identified and implemented suitable matching algorithms, on which the schema matching approach implemented by the author of this work is based. It should be noted that schema matching relies on syntactic clues to judge the similarity of two data schemas, and thus is only useful if the application domains that the two schemas model are similar. Where this is not the case, the user can attempt to find some mapping that takes the role that the various schema elements play in the KDD process into account, as discussed in section 6.6. In this scenario, schema matching should not be used.

Task description

Schema matching attempts to find mappings between elements of two given data schemas (compare (Rahm & Bernstein, 2001)). The elements are concepts, attributes or relationships (in this application). Mappings can be $1:1$, $1:n$ or $n:m$; each mapping comes with a similarity value from the real interval [0..1]. The mappings are simple pairs of elements, without further structure. In this respect, schema matching can be differentiated from ontology mapping or alignment, where mappings are sought that also provide the precise translations between expressions in each ontology; see (Kalfoglou & Schorlemmer, 2003) for an overview. In general, a mapping can be suggested between different types of elements, for example between an attribute and a concept, since different data schemas can represent the universe of discourse (which is assumed to be the same in the two schemas) in different ways. As noted by Madhavan et al. (2001), schema matching is an inherently subjective task, since there may be several plausible mappings between elements of two schemas. Thus it makes sense to suggest the mappings whose similarity value exceeds a certain threshold to users as candidate matches, but to enable them to choose other mappings or to edit the given ones.

The specific matching task in this section is to suggest a connection from a MiningMart data model to local data sets. Note that in order to get valid connections, mappings between different types of elements (like mapping a concept to a column rather than a table) are not allowed. The MiningMart data model to be mapped will usually consist of the *initial* concepts of a previously modelled KDD application, i.e. the concepts that represent the raw data, and the relationships between them. Such concepts are marked by a `DB` flag in M4. However, in the "entry point" approach explained in section 6.5.5, any *intermediate* data model of a modelled preparation process is also examined for its similarity to the technical data sets. The set of intermediate data models is defined by the set of "current data views" of each step in the preparation process. For each step, the

[1] `http://www.postgresql.org`
[2] `http://www.mysql.com`

Algorithm: Computation of Resulting Data Model for a Step

Input: An object S of type Step (possibly connected to a preparation graph)
Output: A collection of objects of type Concept that represents the data view created by S and its predecessors

1. Let I be the collection of all initial concepts (type DB) used as input of the preparation graph attached to S;
2. Initialise global collections *conceptsToBeReplaced* and *visitedSteps* to be empty;
3. Let Concept r := *getReplacingConcept*(S);
4. Remove all concepts in *conceptsToBeReplaced* from I;
5. If r is not *null* then add r to I;
6. return I;

Function *getReplacingConcept*(Step S) returns a Concept:

1. Add S to *visitedSteps*;
2. Set Concept r to *null*;
3. For all predecessors P of S:
4. If P is not contained in *visitedSteps*:
5. Let r := *getReplacingConcept*(R);
6. Add all input concepts of S that are of type DB to *conceptsToBeReplaced*;
7. If S has an output concept o:
8. return o;
9. Else: return r;

Figure 7.4.: An algorithm to compute the data view created by a step and its predecessors.

current data view consists of all initial concepts, but replaces those that were used in the graph preceding the step by the step's output concept. In other words the current data view shows the *results of the current path* of data preparation. Displaying the current data view is a MiningMart feature implemented by this author to help the user keep track of the current preparation path. Figure 7.4 shows the simple algorithm used to compute the current data view of a given step. The algorithm starts with the input data model. A step can "consume" one or more concepts and replace them with its output concept, so the algorithm follows the path to the current step and collects all concepts of the input data model that must be replaced (in the collection *conceptsToBeReplaced*). They are removed from the input data model (line 4 of the main algorithm), and instead the output of the current step is added to it (line 5). The result is the current data view. In the schema matching task, any intermediate data view is a possible entry point for starting data preparation, if the local data sets to which the preparation is to be applied are similar enough. In sum, the MiningMart schema matcher is able to find the best matching of the initial concepts to new data sets, or to find the intermediate data view that achieves the best matching among all intermediate data views. The user can choose to execute either of these two tasks.

Basic elements of the schema matching algorithm

The only types of information that the matching algorithms can use for the envisioned task are (i) the names and (ii) data types of attributes and concepts, or columns and tables, respectively, (iii) which attribute/column belongs to which concept/table, and (iv) the relationship links between concepts (one-to-many or many-to-many relationships; separation and specialisation links are only available in the conceptual model, and thus cannot be used for matching). This level of information is called the schema level by Rahm and Bernstein (2001), who give a survey on schema matching approaches. As noted by these authors and others, schema matching approaches exist that use further information, such as the data (at the instance level), but when the conceptual level is to be mapped to actual data sets, information about data contents is only available on one side, so that it cannot help in the task at hand.

To match names, the system must be able to map a pair of strings to a real value between 0 and 1 that reflects the similarity of the two strings. There are four methods available to do so: a simple one that uses boolean full match of the strings (ignoring case); one that is based on the edit distance between the strings; one that compares the "soundex" representation of the strings; and one that compares all n-grams of the two strings. The last method seems to work best for the task here. These methods are described in (Wagner, 2005). They result in a *name similarity* value between 0 and 1.

To match data types of attributes and columns, the same mechanism that is used elsewhere in MiningMart to "guess" the conceptual data type from the technical type is employed. It simply maps string-based technical data types to *discrete* and numeric types to *continuous*; key columns that are declared as such on the technical level are also recognised. When the conceptual data types of two attributes match, their *type similarity* is 1, otherwise it is 0.

Structural information, such as relationships between concepts, can be available at both levels and is therefore also used. Some schema matching approaches express the structural properties locally, as features of the schema elements to be matched, like (Euzenat & Valtchev, 2004); this allows the representation of the elements by feature vectors, with standard metrics as similarity measures. More advanced methods consider the given schemas as graphs, and incorporate the similarity of neighboured nodes when finding the similarity of two nodes in the respective schemas. By representing relationships, concepts and attributes as nodes in the graph, such methods allow flexible mappings between different types of nodes. One example for this approach is Cupid (Madhavan et al., 2001). Cupid also exploits relationships (foreign key links) between data sets in a second way: such links indicate possible valid joins of data sets, and the result of a join might match a given element of the other schema more closely than any original element. Another graph-based approach is presented in (Melnik et al., 2002).

Since mappings between different types should be excluded in the present task, a simpler matching scheme was developed which is explained in the following. It also adds the results of joins as possible elements to be matched to the local data schema, but not to the conceptual model which is to be reused, because adding a join there would mean to modify the conceptual model during the process of matching, which appears to make the task of editing the suggested mappings rather complex for users. So the following method for schema matching is tailored towards the specific task outlined above, in that

it respects element types and allows joins only in one data schema. It proceeds in a top-down fashion, attempting to match relationships before concepts, but it does not preclude the matching of concepts if their relationships do not match. Thus it does not introduce a top-down bias (Madhavan et al., 2001).

To simplify the implementation, the local data sets are internally represented as concepts with attributes and relationships, like the conceptual data model of the given MiningMart Case. Thus in the following, it suffices to speak about comparisons between these types of elements. All possible results of valid join operations on the local data sets, indicated by foreign key constraints in the database, are also represented by concepts in this schema. This allows to map a concept of the given data model to a join result if the similarity is higher than for the original database objects.

A recurring subtask in this schema matching scheme is to find the best mapping between two sets of elements of the same type. This is needed for matching the attributes of two concepts, or matching the relationships of two data models, or matching the concepts of two data models. The solution taken here is a simple greedy method. A matrix of similarity values is computed. The highest similarity value in the matrix, if it exceeds a similarity threshold which is a global parameter of the whole scheme, gives the first pair of elements to be mapped. Then the corresponding row and column are deleted from the matrix and the procedure is repeated, until no columns or rows remain or until no similarity value exceeds the threshold.

There is also the recurring subtask of computing a global similarity value from such a matrix, which gives the attribute-based similarity of two concepts, for example. This is done by finding the mappings that exceed the threshold in the same greedy fashion. Obviously there cannot be more mappings than the smaller number of elements in the two sets to be compared indicates. The latter number is the maximum number of possible mappings. Therefore the sum of similarity values in the mappings is divided by this number to get the global similarity. However, this means that a concept C with one attribute matches another concept D with a larger number of attributes perfectly, if only that single attribute matches any one attribute of D perfectly. But another concept C' with more attributes might also match D perfectly, in which case the mapping of C' to D should be preferred over the mapping of C to D. The analogous problem holds for other element types. Therefore the global similarity value is decreased with the differing number of elements in the two given sets to be matched. If this number is d then the penalty factor is 0.95^d. This allows matchings that map more elements to reach a higher similarity[3].

The computation of similarity values between elements of the same type is as follows. Attribute names are compared using the name matching methods described by Wagner (2005). The best results are provided by an n-gram matcher, which again uses the greedy, similarity matrix-based method above, where the elements compared in the matrix are the n-grams of the two names to be compared. The default value of n is 3. The conceptual data types of the attributes are used to decrease the name-based similarity by a certain penalty factor (currently 0.75) if they do not match. Note that the conceptual data types

[3] The fact that the penalty factor is the same if two of the attributes of a concept have not been matched, regardless of whether the concept has 4 or 20 attributes, is irrelevant because such concepts are never compared to each other, but only to match candidates from the other schema.

of the local data sets are automatically inferred from their technical data types, which may give inappropriate results.

Concepts are compared by computing their attribute-based similarity using the greedy, matrix-based method outlined above. If the name similarity of the concepts' names does not exceed the global similarity threshold, the attribute-based similarity is reduced by the penalty factor.

Relationships are compared by computing the mean of the similarities of the two concepts of each relationship. For one-to-many relationships, the direction of the relationship is respected; for many-to-many relationships, the better result of comparing the first (second) concept of one relation with the first (second) of the other, or comparing the first of one with the second of the other and vice versa, is taken.

Overall schema matching algorithm

Now that the matching of individual elements, the method for finding the best matchings among several candidates, and the method for computing a global similarity from individual similarities have been explained, the process of matching two data models can be presented. It starts by examining the "stars" of each data model, which are concepts involved in more than one relationship. This heuristic of considering the stars first is chosen in order to take the global structures of the two schemas, which are given by the relationships, into account. Two stars of the two schemas are compared by applying the greedy method from above to all concepts involved in each star. This gives the similarity values for the cells of the matrix that compares the two sets of stars. From this matrix, again using the greedy method and the global threshold, all matching stars are found.

In the second step of matching two data models, the remaining relationships that have not been matched based on the stars are matched, using the greedy method.

Thirdly, all concepts that have not been matched in any previous step are matched.

In each step the result is a set of pairs of concepts of the two schemas, such that the similarity of the two concepts in each pair exceeds the global similarity threshold; only the method for finding the pairs is different in the three steps. The three sets are disjunct by construction. Their union gives all mappings from a given data model to local data sets that can be suggested to the user. If the task was to match the initial data model of a Case, or to match the resulting data model of a particular step, a solution has been found. If the task was to find the best intermediate data model in a Case, then the above method for finding a global similarity of the two current data models is applied, and the search continues with the next intermediate data model. The intermediate data model with the highest global similarity to the target data sets finally gives the mappings suggested to the user.

The user then has the option of modifying the suggested mappings of concepts as desired. Additional mappings can be specified for concepts that could not be matched automatically, and suggested mappings can be changed. Where a suggested mapping involves the result of a join on the local data sets, this is indicated to the user; if such a mapping is confirmed, a view that realises the join is added to the database.

Finally, the individual concepts are connected to the local data sets as specified in the mapping after possible user modifications. For matching attributes to columns, again the greedy approach is used, but without using the global similarity threshold in order to

match as many attributes as possible. Again, the user has the option of modifying the attribute connections; this is part of the main functionality of the MiningMart concept editor. Of course, the concept editor can also be used to match a particular single concept, instead of a complete data model, to the best-matching local data set; the matching methods for this are the same as above.

7.2. New operators in MiningMart

This section briefly describes the implementation of some operators that have been added to MiningMart by the author, in reaction to the analyses from previous chapters. Creating the conceptual-level output for these operators is explained in section 7.1.1; the implementations below concern the MiningMart compiler modules for these operators. An overview of the compiler is given in section 6.4, while details can be found in (Scholz, 2007).

7.2.1. Attribute derivation

This general operator (see section A.5.4) must support an open part, to be programmed by the user, which returns the values of the new attribute. Since MiningMart is implemented in Java, a Java interface was set up for this purpose. It prescribes to implement a certain method which is given a data set and returns values to be added as a new column to that data set; see figure 7.5. Users can create Java classes that implement this interface, and add a Java archive file with their classes to the class path when starting MiningMart. Then, for any step that employs the MiningMart operator AttributeDerivation, a string parameter specifies the name of the class that is to be used for this step.

The operator reads the data from the data set represented by its input concept, and provides it as a two-dimensional string array when calling the user-implemented method `deriveAttribute(...)`. It also provides the names of the columns of the data set. The operator has an optional parameter called *TheTargetAttributes*, which can be used to specify some particular columns of the input data set for whatever purpose. For example, if the operator is supposed to compute the product of two attributes, for each entity, these two attributes can be specified here. The names of the columns that are represented by these attributes are then provided in the string array `namesOfTargetColumns` when the method is called. The method must return the values for the new attribute in the order that matches the order of rows in the given data set, so that the operator can create the correct new data set with the new attribute added.

The operator then creates a table in the database, which is filled with the new data set. It is connected to the output concept of this operator. The output concept is a copy of the input concept, but with one attribute added. The conceptual data type of the new attribute is given by a parameter of this step (i.e. it is specified by the user).

It can be seen that this operator is exceptional in the MiningMart framework, in that it does not process the data inside the database. Also the operator is executed immediately when the compiler runs it (most other operators only create SQL views, so that actual data processing can be done later). Both issues could be resolved by having the user

```
package edu.udo.cs.miningmart.operator;

public interface AttrDerivInterface {

   /**
    * The method expected by the MiningMart operator 'AttributeDerivation'.
    *
    * @param columnNames Names of the columns of the input data set
    * @param namesOfTargetColumns names of target columns, can be NULL
    * @param dataset the input data set (columns in the first dimension,
    * rows in the second dimension)
    *
    * @return a String[] with the values of the newly derived attribute
    */
   public String[] deriveAttribute( String[] columnNames,
                                    String[] namesOfTargetColumns,
                                    String[][] dataset);

}
```

Figure 7.5.: The Java Interface that all classes to be used by the MiningMart operator AttributeDerivation must implement.

create *stored procedures* instead of Java code. Such procedures are programmed in a proprietary language like PL/SQL, which is provided by database system vendors. They can be used (called) in view definitions. As a simple example, one might implement a function that returns the square of its single argument. If that function is called `SQ`, it can be used in a database view definition as follows: `CREATE VIEW example AS (SELECT a, b, SQ(c) AS d FROM some_table)`. When reading data from the view `example`, its column `d` appears as any other column to the caller, but provides the squared values of `c`. However, the language needed to encode them differs between various database systems, and these languages are less well-known than Java. The current implementation of this operator serves as a proof of concept, but can easily be changed to use stored procedures. Automatically (rather than manually) created stored procedures are used by the operator discussed in section 7.2.5.

7.2.2. Pivotisation and reverse pivotisation

These two operators are explained in section A.3.2. They are among the operators that change the status from data to metadata and back, as discussed in section 4.1.1: pivotisation is an operation that transforms the distinct values of a certain attribute into new attributes, while reverse pivotisation transforms a set of attributes into one attribute whose values reflect the original attributes. Such changes between data and metadata are necessary to allow transformations between different representations of the same data, but unfortunately they conflict with the aim of allowing to set up a KDD process model

without executing it (compare section 3.3.3), because the data is not available before execution. In other words, the shape or signature of the conceptual output depends on the actual data contents of the input, which are unknown before executing the operator. Clearly, this property also undermines the reusability of preparation models involving such operators. The solution used in MiningMart is to let the user specify the necessary parts of the data as input parameters. The data characteristics estimations explained in section 7.1.3 can be used for this directly, so that the user does not need to type in distinct data items by hand. For example, the list of values that occur in a certain input attribute is needed for DICHOTOMISATION (section A.3.1), since this operator creates a new attribute for each such value. Since the list of these values may be available through estimation, the corresponding parameter of DICHOTOMISATION can be instantiated automatically. Since the estimated characteristics are available without executing the process, the sketched conflict is avoided to the extent that the estimations are accurate. There is also the option, of course, to execute the process up to the point where the data is needed; then the estimated characteristics can be made accurate by computing them from the actual data. In any case, the user has the option to edit the parameters manually, too. This approach means that the operators must be robust against clashes between the input parameters, which may be estimated or manually given, and the actually used input data. The robustness of the two operators from this section is discussed below.

In order to signal the possibility of using estimated input values as parameters to the MiningMart system, a new M4 constraint (compare section 7.1.1) called USE_VALUES has been introduced. Its two "slots" are the attribute parameter that provides the list of values, and the value parameter where they have to be listed. This allows the system to provide its estimated values automatically to the user for all operators that use this constraint.

To realise n-fold pivotisation, the MiningMart operator PIVOTIZE takes a *list* of index attributes as input parameter. For each index attributes, its distinct data values must be specified in a second parameter; the MiningMart system can insert the estimated value lists of the index attributes automatically here. A third parameter specifies the pivot attribute, whose values are to be distributed into new attributes based on the index values. The new attributes are created automatically at the conceptual level, as section 7.1.1 explains; note that there is one new attribute for each combination of index values from different index attributes. For example, if there are two index attributes `Colour` and `Size`, with distinct values `red`, `green` and `big`, `small` respectively, then there are four new attributes in the output concept for the combinations `red-big`, `red-small`, `green-big`, and `green-small`. Each of the four new attributes takes the value of the pivot attribute, say `Weight`, for those entities that take the combination of index values corresponding to the new attribute, and 0 or the empty value for the other entities. The operator also allows to specify an optional aggregation operator like SUM or MAX, and attributes to group by.

Technically, when the operator is executed, the compiler creates a database view that is represented by the output concept. Continuing the above example, and assuming for ease of reading that the database columns have the same names as the attributes, the compiler would create an SQL statement like the following:

```
CREATE VIEW output AS
```

```
SELECT
  id,
  SUM(CASE WHEN colour = 'green' AND size = 'big' THEN weight ELSE 0 END)
          AS weight_green_big,
  SUM(CASE WHEN colour = 'green' AND size = 'small' THEN weight ELSE 0 END)
          AS weight_green_small,
  SUM(CASE WHEN colour = 'red' AND size = 'big' THEN weight ELSE 0 END)
          AS weight_red_big,
  SUM(CASE WHEN colour = 'red' AND size = 'big' THEN weight ELSE 0 END)
          AS weight_red_small
FROM input
GROUP BY id;
```

This example includes aggregation by summation and grouping by some key attribute `id`.

As mentioned above, PIVOTIZE must be robust against actual data that is different from its specification, for example because the KDD model is reused on different data. There might be additional index values, say `blue`, in its actual input data. This only means that entities that take this value are not represented in the output data set, but the operator does not produce invalid output. On the other hand, a value like `green` which is specified as a parameter might not be present in the actual input data. Then the corresponding output attributes always take the value 0, or the empty value. In both cases the operator's output is syntactically valid, but it might not represent what was originally intended by the designer of the KDD model. Therefore the compiler issues a warning to the user whenever it encounters such mismatches between specified and actual data.

The MiningMart operator REVERSEPIVOTIZE has the following parameters. It takes a list of attributes to be "folded" into one. For each attribute it takes a value or a combination of data values that holds for all values in that attribute. As an example, consider a data set in which car prices are stored in several attributes, depending on the type of car. Assume that the prices of the basic variants of each car are stored in an attribute `BasicPrice`, and the prices of the luxury variants are stored in the attribute `LuxuryPrice`. These two attributes together with the values `basic` and `luxury` are input to the operator. Now the operator creates two output attributes whose names are given as input parameters; one of the attribute takes the pivot values, here the prices, and the other takes the index values, here the variants (`luxury` or `basic`). When PIVOTIZE is applied without aggregation, then REVERSEPIVOTIZE exactly reverses the transformation performed by PIVOTIZE.

Technically, this operator is a little more complex because it creates temporary views which it then unifies. Continuing the car prices example, there would be two temporary views:

```
CREATE VIEW temp1 AS
  SELECT
    car,
    colour,
    BasicPrice AS price,
```

```
    'basic' AS variant
  FROM input;
CREATE VIEW temp2 AS
  SELECT
    car,
    colour,
    LuxuryPrice AS price,
    'luxury' AS variant
  FROM input;
```

Thus each temporary view holds the entities of one variant, with a constant value for the variant in that view. Then the views are unified. It would be possible to integrate all this into one SQL statement, but with temporary views it is easier to read:

```
CREATE VIEW output AS
  SELECT car, colour, price, variant
  FROM
     (SELECT car, colour, price, variant FROM temp1
      UNION
      SELECT car, colour, price, variant FROM temp2);
```

Unlike Pivotize, ReversePivotize is not dependent on actual input data, but creates data from its input metadata.

7.2.3. Aggregate by relationship

This operator is described in section A.2.2. It adds a new attribute to a concept. The new attribute takes aggregated values from a different concept which is linked to the first one by a relationship. Each entity in the first concept is linked, via the relationship, to several entities in the second; the aggregation is done over those entities, and the aggregated value is added as the value of the new attribute to the entity of the first concept. As a further restriction, the aggregation is only done over those entities of the second concept that take the value that is most frequent in the relationship.

To illustrate the technical realisation of this operator, the example from section A.2.2 is used again. There are two concepts, one with customer data and one with product data; they are linked by a relationship that stores which product has been bought by which customer.

The first step in the execution of this operator is to find the product that has been bought most often by customers. Assuming that the relationship is stored in the database cross table `bought`, this can be done as follows:

```
SELECT
  product.pid,
  COUNT(product.pid)
FROM     product, bought
WHERE    product.pid = bought.pid
GROUP BY product.name;
```

The result returned by this query is searched for the most frequent product. Assume that its `pid` value is `1004`. In the second step, a view doing the actual aggregation can be created. The number of times a customer has bought the most frequent product is calculated for each customer in this view. Note that this information comes from the relationship:

```
CREATE VIEW temp AS
  SELECT
    customer.cid,
    COUNT (CASE WHEN product.pid = 1004 THEN product.pid ELSE NULL END)
         AS tempcol
  FROM customers, bought, product
  WHERE customers.cid = bought.cid AND bought.pid = product.pid
  GROUP BY customer.cid;
```

Finally, to attach the information stored in `tempcol` (how often the product `1004` was bought) to the concept representing the customer data, the above view is joined to it:

```
CREATE VIEW output AS
  SELECT
    customer.cid,
    customer.name,
    customer.address,
    tempcol
  FROM customer, temp
  WHERE customer.cid = temp.cid;
```

The operator uses the information about the relationship, which is stored in M4 and which includes the primary and foreign key columns that make up the relationship (here `pid` and `cid`), to create these views.

7.2.4. Dichotomisation

This operator is described in section A.3.1; it creates a binary indicator attribute for each value of a particular input attribute. The realisation of this operator in MiningMart is faced with the same problem as the pivotisation operators (compare 7.2.2, also section 4.3): the shape or signature of the conceptual output depends on the actual data contents of the input, which are unknown before executing the operator. The same solution as for pivotisation is used here. Thus the user specifies one particular input attribute (say `Colour`), its values (like `red`, `green`, `blue`) and for each of these values the name of the output attribute to be created (perhaps `isRed`, `isGreen` and `isBlue`). The user can decide to directly use the values of the input attribute that are estimated to be present by the methods explained in section 7.1.3. This eases the parameter specification when there are many different values in the input attribute. The constraint USE_VALUES explained in section 7.2.2 is used by this operator, too. Since these estimations can be made to reflect the actual data, the user has the two options of using the estimated

values, without the need to execute the preparation graph up to the point where this operator is used, or of using the actual values, after an execution of the graph so far. See below for an explanation why the operator is robust against misestimated values. Names for the new output attributes are suggested automatically when the estimated values are used directly, but can also be specified manually. The output attributes are added to the input concept when the parameter specification is saved (compare section 7.1.1).

When the operator is executed by the compiler, a simple SQL statement creates a virtual column for each output attribute. In the example, three SQL statements would be created as follows:

```
(CASE WHEN colour = 'red' THEN 1 ELSE 0 END)
(CASE WHEN colour = 'green' THEN 1 ELSE 0 END)
(CASE WHEN colour = 'blue' THEN 1 ELSE 0 END)
```

The names of these virtual columns (`isRed` etc.) are stored in M4. Then such statements can be used by following operators to read the binary indicators, like in the following example:

```
CREATE VIEW new_data AS
SELECT ..., (CASE WHEN colour = 'red' THEN 1 ELSE 0 END) AS isRed, ...
FROM ...;
```

When the input data changes because the case is reused on new or updated data, and the parameters of this step are not adjusted, then still valid SQL is created. For example, if the actually occurring values of `Colour` are now `red` and `yellow`, then the output attribute `isGreen` still indicates the absence of the value `green` by only taking the value 0. To create an indicator attribute for `yellow` the user would have to update the conceptual parameters of the step. To make the user aware of such a situation when it arises, the compiler issues a warning if the actual input values of the input attribute differ from the specified parameters.

7.2.5. Results of mining as new attributes

The idea of integrating the results of applying a machine learning algorithm with the data on which it was applied was discussed in section 4.1.2. In MiningMart a few machine learning operators have been included to demonstrate the capability of modelling the whole KDD process in one framework. At the same time MiningMart lacks some important operators that allow to model the experiments around machine learning, in the way exemplified by the YALE system (Mierswa et al., 2006). The reason is that MiningMart puts its focus on data processing inside the underlying database system, but mining algorithms with their superlinear runtime are usually too slow to process the large data sets for which databases are used. Even for smaller data sets, running mining algorithms inside the database is usually inefficient due to the complex ways in which the same data is accessed repeatedly during mining; see the report by Rüping (2002), for example.

An example for a compromise are the support vector machine (SVM) operators in MiningMart. An external implementation of an SVM algorithm is called on data extracted from the database for training. The operator includes a sampling parameter that allows to trim the input data to a size that fits into the client's main memory, which is where the algorithm runs. The result of training the SVM is a prediction function that can be applied to new data. The MiningMart operator translates this prediction function to a

database function that can be called on new data. In this way the *deployment* of the SVM results can be performed on large data sets inside the database. This demonstrates the capability of the developed framework to include both the mining and the deployment phase (see sections 2.1.4 and 2.1.6) in its models. Although mining is technically not done inside the database, at the conceptual level an integrated view of all phases is available. By integrating the results of mining as an attribute, this also holds for the data-centred view.

This section documents the SVM operators in MiningMart, as they were implemented by the author of this work, using a previously available external implementation of the training algorithm, but translating the application of the learned function to a database function. In order to understand how the learned function is realised, a little background on SVMs is given.

As usual in machine learning, a training set S with N examples is represented by N vectors from $X = \mathbb{R}^n$ together with their *label* from a set Y:

$$S = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$$

For classification tasks, the binary case $Y = \{-1, 1\}$ is considered here. For regression (see section 2.1.4), $Y = \mathbb{R}$. The training set is drawn from an unknown distribution $\Pr(\vec{x}, y)$ which determines the learning task: a function $h : X \to Y$ (the *hypothesis*) is sought which assigns an element from Y to any element from X and minimises the error rate, which is the probability of making a wrong prediction on an example drawn randomly according to $\Pr(\vec{x}_1, y_1)$:

$$Err(h) = \Pr(h(\vec{x}) \neq y \,|\, h) = \int L(h(\vec{x}), y)\, d\Pr(\vec{x}, y)$$

where L is a *loss* function $L : Y \times Y \to \mathbb{R}$ that compares the predicted and the actual label.

Since the hypothesis h is unknown and must be found, the space H from which it is taken must be defined. One guideline for defining H is its complexity, because it can be used to bound $Err(h)$, based on $Err_{tr}(h)$, which is the error rate of h on S. The complexity of H is given by its *Vapnik-Chervonenkis* (VC) dimension d, defined as the maximum number of examples that a function from H can separate, if the examples are labelled arbitrarily. As an illustration, consider the real plane $\mathbb{R}^2$ and three linearly independent points in it, and consider H to be the class of straight lines. It is easy to see that for any binary classification (or partition into two sets) of the three points, a straight line exists that separates the points in one class from those of the other. Since this is not possible for four points, the VC dimension is 3 in this case. In general, the VC dimension of hyperplanes in $\mathbb{R}^n$ is $n + 1$.

The bound on $Err(h)$ that is based on d is as follows (Vapnik, 1998; Joachims, 2001), where $1 - \eta$ is the probability that the bound holds:

$$Err(h) \leqslant Err_{tr}(h) + O\left(\frac{d \ln\left(\frac{N}{d}\right) - \ln(\eta)}{N}\right) \tag{7.1}$$

Thus the true error $Err(h)$ is dependent on the training error and on the complexity of the hypotheses. Intuitively, simple functions would not typically give low training errors,

since they often cannot separate the examples. On the other hand, very complex functions can give low training error, but also a high value for the right part of equation (7.1). This can be interpreted as a low generalisation capacity of the learned function, a situation denoted by the term *overfitting*. In both cases the bound is loose. Thus the choice of an appropriate hypothesis space H is crucial.

Support vector machines (Cortes & Vapnik, 1995; Burges, 1998; Joachims, 2001) are based on the principle of *structural risk minimisation*. The general idea of this principle is to choose nested hypothesis spaces of increasing complexity:

$$H_1 \subset H_2 \subset \ldots \subset H_i \subset \ldots \quad \text{with} \quad \forall i : d_i \leqslant d_{i+1}$$

Then the task is to find the index i such that equation (7.1) is minimised. In the case of support vector machines, the risk minimisation works slightly differently. SVMs attempt to find a hyperplane in X that separates the positive ($y_i = 1$) from the negative ($y_i = -1$) examples. The separating hyperplane has the form $\vec{w} \cdot \vec{x} + b = 0$ with norm vector $\vec{w}$ and distance to the origin $b/\|\vec{w}\|$. A separating hyperplane is called optimal if it has the maximum distance to all examples. Intuitively, a bigger distance of the hyperplane to all examples, the so-called *margin*, corresponds to a better generalisation; and in fact it has been shown that a bigger margin corresponds to a lower VC dimension (see (Vapnik, 1982) or (Joachims, 2001)). Thus by finding an optimal hyperplane, the margin is maximised and the right part of equation (7.1) is minimised, and if the hyperplane separates the examples, the training error is also minimised.

It can be shown that finding an optimal hyperplane is equivalent to finding a vector $\vec{w}$ and a constant b_0, such that $\|\vec{w}\|$ is minimal and $y_i(\vec{w} \cdot \vec{x}_i + b_0) \geqslant 1$ holds for all $1 \leqslant i \leqslant N$. Minimising $\|\vec{w}\|$ or, equivalently, $\frac{1}{2}w^2$, under the given constraints, is the problem solved by SVM algorithms.

In general, it may not be the case that a separating hyperplane exists. For such cases, errors are allowed by introducing slack variables ξ_i for each training example, which correspond to the classification error, i.e. they are positive if the example is wrongly classified, and measure the distance to the hyperplane. Now to minimise the global error, the function to be minimised is no longer $\frac{1}{2}w^2$ but $\frac{1}{2}w^2 + C\sum_i^N \xi_i$, under the same constraints and additionally $\xi_i \geqslant 0$ for all $1 \leqslant i \leqslant N$, and with a parameter C used to balance the influence of wrongly classified examples.

To solve this optimisation problem, the saddle point of its Lagrange functional must be found; without going into details, the Wolfe dual form of the equation to be minimised can be given as:

$$W(\vec{\alpha}) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j + \sum_{i=1}^{N}\alpha_i$$

The scalar product is represented by "$\cdot$" here. This form must be maximised under the constraints $\sum_{i=1}^{N}\alpha_i y_i = 0$ and $0 \leqslant \alpha_i \leqslant C$. It depends only on $\vec{\alpha}$. When $\vec{\alpha}$ has been found during training, it can be used to predict the label of an unlabelled example $\vec{x}$ by computing

$$F(\vec{x}) = \text{sign}\left(\sum_{i=1}^{N}\alpha_i y_i \vec{x}_i \cdot \vec{x} - b\right).$$

The constant b can be computed from the training examples. Note that the prediction function F depends on the training examples for which $\alpha_i \neq 0$. These examples are called *support vectors*, they are the closest training points to the found hyperplane, and the only points that determine the position of the hyperplane. The SVM-based MiningMart operators must implement this function F in the database, which means that a table with the support vectors must be available in the database.

In the case of regression, the real values to be predicted are approximated by a linear function, and the SVM minimises the sum of errors made by this approximating function. Both for classification and regression, an extension to non-linear functions is possible by transforming the input space X into some other space $\mathcal{X}$, by a non-linear transformation $\Phi : X \to \mathcal{X}$. The training equation and F can then be restated as follows:

$$W(\vec{\alpha}) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \Phi(\vec{x}_i)\cdot\Phi(\vec{x}_j) + \sum_{i=1}^{N}\alpha_i$$

and

$$F(\vec{x}) = \text{sign}\left(\sum_{i=1}^{N}\alpha_i y_i \Phi(\vec{x}_i)\cdot\Phi(\vec{x}) - b\right).$$

In other words, only the scalar product in $\mathcal{X}$ is needed to solve the problem as before. This allows to employ the "kernel trick": the transformation function Φ is chosen such that a kernel function $K : X \times X \to \mathbb{R}$ exists with $K(\vec{x}_1, \vec{x}_2) = \Phi(\vec{x}_1)\cdot\Phi(\vec{x}_2)$ for all $\vec{x}_1, \vec{x}_2 \in X$. Then all scalar products involving $\Phi(\cdot)$ above can be replaced by $K(\cdot,\cdot)$. Some known suitable kernel functions, which are also used by the MiningMart SVM operators, are:

- The linear kernel (corresponding to $\Phi = id$): $K(\vec{x}_1, \vec{x}_2) = \vec{x}_1 \cdot \vec{x}_2$
- Polynomial kernels: $K(\vec{x}_1, \vec{x}_2) = (\vec{x}_1 \cdot \vec{x}_2 + 1)^p$ for $p \in \mathbb{N}$
- Radial basis kernels: $K(\vec{x}_1, \vec{x}_2) = \exp(-\gamma|\vec{x}_1 - \vec{x}_2|^2)$ with $\gamma \in \mathbb{R}^{\geqslant 0}$
- Sigmoid kernels: $K(\vec{x}_1, \vec{x}_2) = tanh(s(\vec{x}_1 \cdot \vec{x}_2) + c)$ for certain $s, c \in \mathbb{R}$.

The decision function F can then be written as

$$F(\vec{x}) = \text{sign}\left(\sum_{i=1}^{N}\alpha_i y_i K(\vec{x}_i, \vec{x}) - b\right). \tag{7.2}$$

A final aspect of SVMs that is needed in this section is their ability to estimate their generalisation error without using a test set. Usually, after training a predictive learner, its performance can only be determined on a separate set of examples that were not used for training, but whose labels are known. By comparing the known labels to the predicted ones, an empirical error is found and taken as an approximation of the true error of the learned model. The closest approximation possible is obtained by using all labelled examples for training except for one, then testing on this one example, and repeating this for all examples. Averaging over the single errors renders the so-called leave-one-out error. The problem is that for N training examples, this requires N learning runs which is usually not feasible. In practice, the number of examples held out for testing is often

increased to N/j, and this is repeated j times with disjunct test sets. So the number of learning runs is reduced to j, where often $j = 10$ is chosen. This process is called cross validation.

This scenario is applicable to any predictive learner. However, SVMs provide a different, unique method for estimating the empirical error. The method is called $\xi\alpha$-estimation because its inputs are the two vectors $\vec{\xi}$ and $\vec{\alpha}$ described above. It was introduced by Joachims (2000). Let $\vec{\xi}$ and $\vec{\alpha}$ be the vectors computed during a training run of an SVM as described above. The *$\xi\alpha$-estimator of the error rate*[4] for a hyperplane h is

$$Err_{\xi\alpha}(h) = \frac{d}{N} \text{ with } d = |\{i \mid (\alpha_i R^2 + \xi_i) \geqslant 1\}| \tag{7.3}$$

where N is the number of training examples and R^2 is an upper bound on the kernel function evaluated on any pair of examples.

The key measure in definition (7.3) is obviously d. It counts the number of examples for which the inequality $(\alpha_i R^2 + \xi_i) \geqslant 1$ holds. There is a connection between this inequality and those examples that can produce a leave-one-out error if they are not used for training, but for testing. More precisely, if an example $(\vec{x}_i, y_i)$ is not classified correctly by an SVM trained on a sample *without* it, then for this example the inequality must hold for an SVM trained on the sample *with* it. Therefore, all examples for which the inequality does not hold cannot produce a leave-one-out error. So the $\xi\alpha$-estimator is an approximation to the leave-one-out error which is never too low, i.e. never too optimistic. It can be computed during the training run of an SVM at virtually no extra cost. Empirical tests have shown that the estimator is often, but not always, tight enough to be useful in practical applications. In particular for text data it works well (Joachims, 2001).

To sum up this discussion with respect to the MiningMart SVM operators, they must be able to implement the decision function F in the database, using a certain kernel function and its parameters, and incorporating the support vectors, their labels y, their α and ξ coefficients, and the constant b. The last five are the output of the external SVM training algorithm that the operators call, while the kernel function and its parameters are input parameters of the operators which they pass to the training algorithm and also use for implementing the decision function F. An additional input parameter is the error-bounding constant C.

There are four MiningMart operators that involve the support vector machine: one for classification, one for regression, one for replacing missing values by predicting the missing values using a regression SVM (trained on the data rows where the value of the attribute in question is *not* missing), and one for automatic feature selection based on the $\xi\alpha$-estimation method. All these operators use the SVM *wrapper* that controls the external algorithm and provides the learned decision function as a database function. The external algorithm that MiningMart uses is MYSVM[5], implemented by Stefan Rüping for his thesis (Rüping, 1999). The tasks of the wrapper are to read the input data from the database table or view that is represented by the input concept of the MiningMart operator, to read C and the kernel parameters from the operator, to run the external

[4] The definition here is slightly simplified.

[5] http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/index.html

algorithm on this data with those parameter settings, to create a temporary table in the database that stores the support vectors and their α values, and to implement the decision function in the database (except for the feature selection operator). The α values that MYSVM provides for its support vectors have already been multiplied by their label y, so there is no need to store the label in the temporary table as well.

As explained in section 7.2.1, today's major database systems provide the option to include calls to *stored procedures*, which are functions and procedures programmed in a proprietary language, in database views. An SVM decision function implemented in this way must access the temporary table of support vectors internally; compare equation 7.2. Alternatively, the support vector values could be hard-coded in the function, but since there can be rather many support vectors for large data sets, the solution with a temporary table is more elegant. Both the temporary table and the decision function remain in the database until the step with the SVM operator that created them is deleted, or is compiled again (the MiningMart compiler keeps a list of such temporary objects that have been created during compilation; see (Scholz, 2007) for details).

The feature selection operator that involves the SVM does not require to use the decision function, as it only uses the $\xi\alpha$-estimator after training to guide the search for a set of features (attributes) of the input data set on which the SVM achieves the best result, or a similar result as with all features but in less time. This operator is described in more detail in (Euler, 2002a). The computation of the $\xi\alpha$ result is done by the external algorithm automatically, and is read by the operator. The operator provides two simple feature selection strategies, but uses a simple interface to the SVM wrapper so that other strategies can easily be realised.

Figure 7.6 shows an example of a decision function as created by the MiningMart operator that employs an SVM for classification. The version shown compiles on Oracle database systems; under Postgres there are some slight differences in the syntax, but the operator can also create Postgres versions. The name of the function reflects the internal identifier of the step that applies the operator. The input parameters of the function are the database columns with the data row on which the function is applied; the four input parameters are named after the four columns that have been used for training, although the function can of course also be applied to four different columns. In this example the four training columns represent a time window of width four, to which a scaling operator has been applied. These column names are also used in the model table with the support vectors, called `CS_100110056_MODEL`, to identify the entries of each support vector. There is a declaration part that is used to declare all internal variables used by the function. The "cursor" variable `supportvectors` provides the contents of `CS_100110056_MODEL`. The "row type" variable `currentrow` iterates through these contents. The variable `inner` contains the scalar product of one support vector and the incoming example. The variable `kernel` evaluates the kernel function, in this example a polynomial kernel of degree 2, and multiplies the result with the α value of the support vector. The variable `retValue` computes the sum over the support vectors, to which the constant b is added. The sign of the final value of this variable is returned by the function. Compare equation (7.2) above.

```
CREATE OR REPLACE function CS_100110056_F (
   IN_SCALED_WINDOW1 IN NUMBER,
   IN_SCALED_WINDOW2 IN NUMBER,
   IN_SCALED_WINDOW3 IN NUMBER,
   IN_SCALED_WINDOW4 IN NUMBER
)
RETURN NUMBER
AS
BEGIN
  DECLARE
    retValue NUMBER;
    CURSOR supportvectors IS
        SELECT SCALED_WINDOW1,
               SCALED_WINDOW2,
               SCALED_WINDOW3,
               SCALED_WINDOW4,
               Alpha
        FROM CS_100110056_MODEL;
    currentrow supportvectors\%ROWTYPE;
    kernel NUMBER;
    inner NUMBER;

  BEGIN

    retValue := 0;
    FOR currentrow IN supportvectors
    LOOP
      inner := (currentrow.SCALED_WINDOW1 * IN_SCALED_WINDOW1)
               + (currentrow.SCALED_WINDOW2 * IN_SCALED_WINDOW2)
               + (currentrow.SCALED_WINDOW3 * IN_SCALED_WINDOW3)
               + (currentrow.SCALED_WINDOW4 * IN_SCALED_WINDOW4);
      kernel := POWER(inner + 1, 2) * currentrow.Alpha;
      retValue := retValue + kernel;
    END LOOP;
    retValue := retValue + (-0.2233839308663433);

    IF (retValue >= 0)
      THEN RETURN 1;
    ENDIF;
    RETURN -1;
  END;
END;
```

Figure 7.6.: A stored function in PL/SQL (Oracle), automatically created by a MiningMart operator, that realises the decision function of an SVM trained with a polynomial kernel.

7.2.6. ReverseFeatureConstruction

This operator supports the deployment phase of the KDD process. It reverses certain transformations that have been applied to an attribute. As explained in section 2.1.6, a prediction function learned by a mining algorithm predicts values of the kind that have been used as labels during training. However, if the label attribute had been transformed before training, then the predicted values have to be transformed back in order to get predictions in the original domain of the attribute. This has been referred to as post processing in this work. Criterion 52 (appendix C) therefore requires that a reversing operator be automatically available whenever an attribute is transformed in a reversible way. The MiningMart operator "ReverseFeatureConstruction" has been provided for this purpose. It can reverse any application of SCALING and VALUE MAPPING, since these are the only reversible transformations currently provided by other MiningMart operators (the mappings performed by an application of VALUE MAPPING may also be non-reversible if several values have been mapped to one).

Because this operator is not useful if there is no step whose transformations can be reversed, a step that employs this operator cannot be created in the usual way in MiningMart, but has to be created using a "wizard" that requires the user to select an existing step to be reversed. If the selected step does not employ a reversible operator, the wizard prevents the creation of the new step.

When compiled, an instance of this operator must read the parameters of the original transformation in order to be able to reverse it. Therefore the step to be reversed must be linked to the reversing step (which employs this operator). This type of link between steps is stored by an additional M4 type, whose objects simply refer to the two steps involved. The link is created by the wizard. One of the parameters of the reversing step refers to the originally transformed output attribute of the step to be reversed. When the reversing step (with this operator) is compiled, the compiler module thus knows which transformation to reverse. The other scaling or value mapping parameters of the step to be reversed provide the information needed to set up the reverse transformation; it is encoded in SQL and used for the output of the reversing step.

7.3. Materialisation recommendations

As explained in section 6.4, the MiningMart compiler uses database views (at the technical level) to create the new representations of the data resulting from operator applications. A chain of operators, when compiled, thus leads to a stack of views, each of which depends on the previous view. More generally, the view dependencies parallel the structure of the DAG of MiningMart steps given at the conceptual level. In larger applications, such as the one described in chapter 5, the nesting of views can become rather complex. At the technical level, the problem arises that reading data from a view that depends on other, deeply nested views can be rather inefficient, because every tuple in the original data table(s) has to be accessed and possibly re-represented by each intermediate view.

An obvious solution is to *materialise* some of the intermediate views, so that they become tables. Now the question is which views should be materialised. Considering data preparation for KDD, which usually leads to a single final data set to be used for mining,

reading data from this final set must be efficient, as it is the interface to data mining algorithms, so this final set ought to exist as a table after preparation. Clearly, then, the final view of a data preparation process has to be materialised, and intermediate views should also be materialised if this can reduce the overall time needed for *all* materialisations. This section discusses when this might be the case. The ideas discussed below lead to an automatic method of identifying suitable places for materialisation in the preparation graph, which is needed for hiding the technical level. Although materialisation can be done automatically, in MiningMart the adopted solution is to *recommend* places for materialisation to the user, and to include a materialisation operator at the respective place in the preparation graph only if confirmed by the user. While this weakens the separation of the two levels slightly, it gives more control of the system's storage behaviour to the user.

The issue of selecting views to materialise is known from data warehousing, but with a somewhat different problem setting. The scenario is that there are a number of base tables in an operational database system, and a set of views on these base tables that make up the data warehouse. To enable efficient retrieval in the warehouse, the views in it are materialised. The problem of selecting the views to materialise thus arises in the design phase of the warehouse (Gupta, 1997; Gupta & Mumick, 2005), and involves considering average querying and update costs. The latter occur whenever the contents of the base tables change so that the views have to be updated (though often, updates are done in regular intervals, rather than being triggered by any change to the base tables). The usual approach to this problem considers a set of given queries, together with expected query frequencies, that the warehouse will have to answer. Equalling queries with views, the set of views to materialise can be chosen from this set, although approaches that consider additional views have also been proposed; see for example (Ross et al., 1996; Theodoratos & Xu, 2004). Typically, the set of given queries is examined for common subexpressions which might be worth materialising; this is called multiple query optimisation (Sellis, 1988; Mistry et al., 2001). But this alone does not take updates into account. Update frequencies are usually also modelled for each given query, reflecting how often a materialised view that realises this query would have to be updated. The view selection problem is then to minimise the sum of querying and updating costs, under a global maximum space constraint. Querying costs are minimal when all views are materialised, updating costs are minimal when no views are materialised. The problem is NP-hard (Gupta, 1997).

Fortunately, for the present purposes the issue is less complex. There is no question of optimising response time over a set of queries; rather, there is only a single query (the final data set for mining), which should be materialised anyway. As said above, what is to minimise here is the overall time needed for materialising the final data set and any intermediate sets. The costs for materialising the latter are analogous to the update costs in the warehousing scenario. In spite of this analogy, the optimal solution is not to materialise no intermediate view, because the unavoidable "update" cost of materialising the final data set could be too high (it is unavoidable because otherwise the querying cost for querying this data set, at the interface to data mining, would be very high). Materialising all views, on the other hand, consumes a lot of space, and is unnecessary because the cost of reading from views that are not deeply nested is not high. In other

words, it would certainly be enough, for example, to materialise every third or fourth view on any path through the preparation graph. But can the number of materialisations be reduced further?

To answer this question, the costs of reading data from a view are examined more closely. If the view depends on a single base table, then even if there are intermediate views it is justified to approximate the processing costs for reading from the view by the number of tuples in the base table (this is done, for example, by Harinarayan et al. (1996), who consider the materialisation of nested aggregation views). Suppose there is a base table T and a sequence of k views $V_1, \ldots, V_k$ such that V_1 is based on T, and V_i is expressed over V_{i-1} for $2 \leqslant i \leqslant k$. If V_k is to be materialised, every tuple from T must be processed, even if not many tuples belong to V_k due to some selectivity in the sequence. Materialising one or more of $V_i, \ldots, V_{k-1}$ does not change this situation and thus will not reduce the overall materialisation costs. It is easy to confirm this experimentally. However, there is one exception if the preparation operators from chapter 4 are considered, rather than only standard relational operators: since ATTRIBUTE DERIVATION (section A.5.4) may use its complete input in rather arbitrary ways to create the values of its new attribute, it might read its input several times, possibly resulting in tuples from T being processed more than once. This exception is discussed again below.

A view can be dependent on more than one base table, of course, if it represents the output of a JOIN or UNION operation, which are the only operators in chapter 4 that apply to more than one input data set. For joins, the processing time for reading from the output view can only be bounded by the product of the sizes of the base tables. Nevertheless, the output views of these operators are not more suitable places for materialisation than other views, since the number of base table tuples to be processed would not change if materialisation were used.

However, what does change the number of base table tuples to be processed is any view over which more than one other view is expressed. Suppose the views V_2 and V_3 are both expressed in terms of V_1. It can be assumed that both V_2 and V_3 will be read from when the final mining table is materialised, since otherwise one of V_2 or V_3 or both would be useless for the preparation. Reading from V_2 means processing the base table that V_1 is based on, and the same holds for V_3. So the tuples from this base table are processed twice. If V_1 or its predecessors involve some selectivity, the overall processing can be made more efficient if V_1 is materialised.

This leads to the idea that all steps in a preparation graph whose output is consumed by more than one other step should materialise their output (these are the nodes with outgoing degree bigger than 1). Note that this method is independent of given data contents, and can thus be applied at the conceptual level alone. What is avoided by this method is reading tuples from a base table more than once. Returning to the exception mentioned above, namely the possibility that ATTRIBUTE DERIVATION processes its input more than once, one can argue by the same token to materialise all inputs of steps that involve this operator.

These ideas were experimentally validated using different materialisation schemes in the model application described in chapter 5. As noted there, this application involves more than 90 steps, not counting the materialisation operators. The total time for compiling this application in MiningMart, which includes materialisation if any operator uses

it, has been measured on artificially created data sets with 100000 tuples representing customers, and more than five million tuples with call details for these customers. Due to aggregation and some selectivity, the final mining table (with one tuple per customer), which is materialised in all experiments, contains 97052 tuples.

Using no intermediate materialisation at all, the materialisation of the final table was stopped without having finished after more than 24 hours. Using materialisation of the outputs of the steps with outgoing degree higher than 1, the total execution time was 1 hour and 44 minutes. Four such steps exist in the application; an experiment with four materialisations inserted at random places also was stopped without a result after 24 hours.

While no steps in that application involve a complex attribute derivation in the sense discussed above, there are a few operators that are special cases of ATTRIBUTE DERIVATION, and that must read their input indeed more than once. An important example is DISCRETISATION with an automatic generation of discretisation intervals: the minimum and maximum values of the attribute to be discretised must be read before the output column can be defined; then reading from the output inevitably involves the second or third scan of the input data. After adding materialisation of the input of such operators, the total execution time fell to 1 hour and 12 minutes. Adding still more materialisations did not lower the total execution time, which confirms the approach discussed above. It should be noted that for technical reasons, some MiningMart operators always materialise their output, of which one operator (the MiningMart version of AGGREGATION) is employed twice in the application used for the experiments.

Although materialisation is a technical concept, recommending suitable places for it is then based solely on information from the conceptual level, and can be done without having processed any data. This property supports the reusability of conceptual models on new data, as discussed in section 6.6. A MiningMart module that performs such recommendations, and inserts materialisation operators automatically when confirmed by the user, was therefore added to the system by the author. Conforming to criterion 11 from appendix C, it is automatically checked if any recommendations should be given whenever the user compiles a complete application on large input data (using a configurable threshold for input data size).

7.4. The user interface

This section briefly introduces a few aspects of the implementation of MiningMart's graphical user interface (GUI). The GUI provides the two dual views on the KDD process, and allows to edit and annotate elements of it. Compare figure 1.4, or the figures in chapter 5. The implementation of the GUI is based on LiMo, a modelling framework developed at the University of Dortmund by Pleumann (2007). While the framework was intended to support the graphical representation of (models of) software architectures, it turned out to be useful for the graphical representation of KDD models as well. MiningMart has thus been one of the applications that confirmed the usefulness and validity of LiMo (Pleumann, 2007).

LiMo is used to represent the M4 model elements graphically. In LiMo, a "core meta model" is available that provides abstract Java classes for models and model elements.

There are two types of model elements, those for figures and those for connections. Model elements for figures can be nested. For the implementation of the MiningMart GUI, classes that represent the M4 types have been made to inherit from classes of LiMo's core meta model. Figure model elements were used for steps, chunks, concepts and attributes; connection model elements were used for semantic links and step dependencies (the latter represent the data flow in the process view). Nesting of figure model elements was useful for the chunks of preparation graphs, which can be nested, too (compare section 4.4).

The graphical representation of the M4 objects (or in LiMo terms, of the model that is specified by extending the core meta model) is then realised by drawing elements for figures and connections that "observe" the model elements: as soon as the latter change, the former are updated, too, effectively updating the graphical display. The observation mechanism is a well-known design pattern from object-oriented programming (Gamma et al., 1995). LiMo's drawing elements provide almost the full graphical interface, including the observation and update mechanism, leaving only small specifications about what the figures and connections should look like to the developer. The main part of the GUI implementation thus concerns threading and specific dialogs with the user.

LiMo also allows to annotate any model element using HTML text. These annotations could easily be mapped to the annotations that M4 provides.

In sum, LiMo has been a very suitable graphical framework for the MiningMart system, thanks to the fact that MiningMart uses an explicit model of the KDD process, which LiMo's graphical tools can directly represent. This is another advantage of the declarative modelling approach used in MiningMart.

7.5. Summary

This chapter has provided a more dynamic view of the MiningMart system than chapter 6. Section 7.1 has explained how elements of the data view are created automatically and in a generic way as soon as elements of the process view are created. The propagation of changes in both views, the estimation of data characteristics, and the schema matching algorithm have also been presented. Section 7.2 has explained the realisation of some important operators, including the deployment of the function learned by a mining algorithm inside a database. Section 7.3 has extended the view-based compiler approach by a strategy for materialisation, in order to speed up the execution of complex preparation graphs. Finally, section 7.4 has taken a short look at the graphical user interface and its implementation based on an existing framework for the graphical representation of structured models.

Name	Applies to	Meaning
Constraints indicating data type of output attribute		
TYPE	Output attribute	Use given type
SAME_TYPE	An input and an output attribute	Copy type to output
OUT_TYPE	Input attribute	Use given type for output created from the attribute
Constraints indicating how to create output attributes		
SAME_FEAT	Input and output concept	Copy features (attributes) to output
ALL_EXCEPT	Output concept and input attribute(s)	Copy all features (attributes) from input to output except given ones
RENAME_OUT	An input attribute and an output name	Copy input attribute to output but use given name
MATCHBYCON	Input attribs from different input concepts	Copy only one of the given input attributes to output
CREATE_BY	Input attribute and input values	Create one output attribute per given value, based on given input attribute
CR_SUFFIX	Input attributes	Copy to output but add suffix to name
FEAT_RFR	Input relationship and output concept	Use attributes of From-concept of given relationship for output
FEAT_RTO	Input relationship and output concept	Use attributes of To-concept of given relationship for output
Constraints indicating where to find input attributes		
IN	Attributes and concepts	Given attribute must belong to given concept
IN_RELFROM	Input attribute and input relationship	Attribute must be in From-concept of given relationship
IN_RELTO	Input attribute and input relationship	Attribute must be in To-concept of given relationship
Constraints used for creating output relationships		
FROMCON	Output relationship and input concept	Use given concept as From-concept of output relationship
TOCON	Output relationship and input concept	Use given concept as To-concept of output relationship
CROSSCON	Output relationship and input concept	Use given concept as cross table concept of output relationship
FROMKEY	Output relationship and input attributes	Use given attributes as keys of From-concept
TOKEY	Output relationship and input attributes	Use given attributes as keys of To-concept
CR_FROMKEY	Output relationship and input attributes	Use given attributes as keys of cross concept to From-concept
CR_TOKEY	Output relationship and input attributes	Use given attributes as keys of cross concept to To-concept

Table 7.1.: List of M4 constraints that can be used to specify how conceptual-level data output can be automatically generated from given input parameters of an operator.

Name	Meaning
Assertions related to size of output concept:	
SZ_BY_REL	Compute size for Join based on relationship between inputs
SZ_BY_VAL	Size is given by a specified constant
SZ_MIN_MV	Output size is input size minus no of MVs of specified attribute
SZ_DIV_BY	Output size is input size divided by value of specified parameter
SZ_BY_VL	Get size from combinations of distinct values of specified attributes
SZ_ADD	Input sizes are added to give output size
SZ_MULT_NO	Output size is input size times no of attribs in specified parameter
Assertions related to the list of values of an output attribute:	
VL_FROM	Take value list from specified attribute
VL_UNCH	Copy value list from corresponding input attribute
VL_ADD	Add value given by specified parameter to input value list
VL_BY_PAR	Take value list from specified parameter
VL_BY_SYM	Value list is given by particular symbols
VL_COMB	Combine (merge) value lists of corresponding input attributes
VL_BY_LIST	Value list is given directly or by specified parameter
Assertions related to the minimum and maximum bounds of an output attribute:	
MM_FROM	Take bounds from specified attribute
MM_UNCH	Copy bounds from corresponding input attribute
MIN_FROM	Take minimum from specified value or parameter
MAX_FROM	Take maximum from specified value or parameter
MM_COMB	Combine (merge) bounds of corresponding input attributes
Assertions related to the value frequencies (VF) of an output attribute:	
VF_FROM	Take VFs from specified attribute
VF_ADD	Add VFs from corresponding input attributes
VF_REPL_MV	Take VF of specified value from no of MV
VF_BY_SEL	Multiply VFs of input attribute by selectivity factor
VF_BY_AGG	Get VFs from combinations of distinct values of specified attributes
VF_MULT_NO	Multiply VFs of input attrib by no of attribs in specified parameter
Assertions related to the number of missing values:	
MV_BY_SEL	Multiply no of MVs of input attribute by selectivity factor
MV_FROM	Take no of MVs from specified attribute
MV_ADD	Add no of MVs from corresponding input attributes
General assertions:	
NO_CHANGE	Copy all relevant estimations from input to output
ES_SELECT	Apply special selectivity estimation for ROW SELECTION

Table 7.2.: List of M4 assertions that can be used to specify which inferences and estimations of data characteristics are possible for an operator. MV = missing value, no = number, VF = value frequency.

8. Evaluating KDD Tools

The previous chapters have set the background to understand many important issues during data preparation and other phases of the KDD process. This chapter applies this background to develop detailed criteria which serve to evaluate software packages that support KDD. Generally, this work argues that KDD software should be evaluated according to the extent to which it supports the conceptual description level discussed in previous chapters. Measures for this extent are given in the shape of concrete, objective and quantifiable criteria in section 8.3 and appendix C, as a slightly extended version of (Euler, 2005a). But first some related work is reviewed (section 8.1) and the methodology used is discussed in section 8.2. A small test case that can be easily used in practice to determine the degree to which a given tool fulfils each criterion is presented in section 8.4. Section 8.5 describes a number of software packages which have been evaluated under the criteria from section 8.3; the results are presented in section 8.6.

8.1. Related work

8.1.1. General software evaluation

There are many aspects of software which can be evaluated. A useful distinction is that between the development of a software and its actual use as a *product*. The main evaluations concerning the development of software assess the quality and correctness of the source code; this is usually called *testing*. Testing is a complex issue, but this work does not involve testing a software. A good overview of software testing methods is given in (Riedemann, 1997). A higher-level type of evaluation assesses the development *process* in an institution, to see whether it follows certain standards that make the process controllable and repeatable. The software capability and maturity model (CAMM) is a major evaluation framework for development processes (Paulk et al., 1995).

The present work is concerned with software product evaluation, which addresses the central notion of software *quality*, and is defined as the assessment of software quality characteristics according to specified procedures (Punter et al., 1997). The characteristics of software quality are defined in an international standard, ISO/IEC 9126, entitled "Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for their Use", developed in 1991 and slightly modified several times afterwards. It defines six main characteristics of software quality, each with several subcharacteristics, as listed in figure 8.1. These characteristics can be the subject of an evaluation of a software product. The standard is a result of a decade of research that is mainly based on Boehm et al. (1978) and Cavano and McCall (1978). While the ISO standard 9126 aims at comprehensiveness, Kusters et al. (1997) and others have pointed out that different users of a product may have rather different quality requirements, and that it

may be difficult for an organisation to determine the level and type of quality required in a specific situation.

Most of the ISO 9126 characteristics refer to external quality attributes, that is, such characteristics as can be examined when the software's source code is not available. However, at least the maintainability characteristic concerns internal aspects which are related to the code. This work considers only external characteristics; this view of software is often subsumed under the notion *COTS* (commercial off-the-shelf) software (Maiden et al., 1997; Colombo & Guerra, 2002).

Importantly, the evaluation itself should also follow a standard procedure in order to be as objective as possible, and in particular to be reproducible. To this end another standard was published in 1999, the ISO 14598 standard, entitled "Information Technology – Software Product Evaluation". It introduces four phases that make up the evaluation process:

1. Establish evaluation requirements: The purpose of the evaluation, and the types of products to be evaluated, must be identified in this phase. Most importantly, a *quality model* is set up, which lists the characteristics that are agreed to bear an influence on the quality. The ISO 9126 quality characteristics provide a useful guide, or a checklist, for the identification of quality-related issues in a particular evaluation, but the ISO 14598 standard also allows other categorisations of quality that are more appropriate under the given circumstances. ISO 14598 explicitly states that there are no established methods for producing software quality specifications.

2. Specification of the evaluation: Since the ISO 9126 characteristics are not directly quantifiable, metrics that are correlated with them have to be established. The term "metric" is used in ISO 14598 not in the usual mathematical sense, but refers to a quantitative scale and a method which can be used for measurement. The word "measure" is used to refer to the result of a measurement (the term "score" is also used in this work). According to ISO 14598, every quantifiable feature of software that correlates with a characteristic from the quality model can be used as a metric. For every metric, a written procedure is needed that prescribes the assignment of measured values to it, to achieve objectivity.

3. Design of the evaluation process: An evaluation plan is produced that specifies the required resources, e.g. people, techniques or costs, and assigns them to the activities to be performed in the last phase.

4. Execution of the evaluation: Measurements are taken and scores computed as fixed in the evaluation plan.

In (Punter et al., 2004) a critical review and some refinements of this process can be found. In particular, the importance of establishing and prioritising the goals of an evaluation, and of involving all stakeholders of the evaluation in this, are stressed. Since the present work involves only one evaluator and has a clear, simple objective (see section 8.2), these refinements are not used here. Instead, section 8.2 describes the instantiation of the above process in the present work. Other ideas from the literature below are also used.

- Functionality – the capability of the software to provide functions which meet stated and implied needs when the software is used under specified conditions
  - Suitability – the capability of the software to provide an appropriate set of functions for specified tasks and user objectives
  - Accuracy – the capability of the software to provide right or agreed results or effects
  - Interoperability – the capability of the software to interact with one or more specified systems
  - Security – the capability of the software to prevent unintended access and resist deliberate attacks intended to gain unauthorised access to confidential information, or make unauthorised modifications to information or to the program so as to provide the attacker with some advantage or as to deny service to legitimate users
- Reliability – the capability of the software to maintain the level of performance of the system when used under specified conditions
  - Maturity – the capability of the software to avoid failure as a result of faults in the software
  - Fault tolerance – the capability of the software to maintain a specified level of performance in cases of software faults or of infringement of its specified interface
  - Recoverability – the capability of the software to re-establish is level of performance and recover the data directly affected in the case of a failure
- Usability – the capability of the software to be understood, learned, used and liked by the user, when used under specified conditions
  - Understandability – the capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use
  - Learnability – the capability of the software product to enable the user to learn its application
  - Operability – the capability of the software product to enable the user to operate and control it
  - Attractiveness – the capability of the software product to be liked by the user
- Efficiency – the capability of the software to provide the required performance, relative to the amount of resources used, under stated conditions
  - Time behaviour – the capability of the software to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions
  - Resource utilisation – the capability of the software to use appropriate resources in an appropriate time when the software performs its function under stated conditions
- Maintainability – the capability of the software to be modified
  - Analysability – the capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified
  - Changeability – the capability of the software product to enable a specified modification to be implemented
  - Stability – the capability of the software to minimise unexpected effects from modifications of the software
  - Testability – the capability of the software product to enable modified software to be validated
- Portability – the capability of the software to be transferred from one environment to another
  - Adaptability – the capability of the software to be modified for different specified environments without applying actions or means other than those provided for this purpose for the software considered
  - Installability – the capability of the software to be installed in a specified environment
  - Co-existence – the capability of the software to co-exist with other independent software in a common environment sharing common resources
  - Replaceability – the capability of the software to be used in place of other specified software in the environment of that software

Figure 8.1.: The ISO 9126 software quality characteristics and subcharacteristics, taken from (Punter et al., 1997).

A new standard, ISO 25000, entitled "SQuaRE – Software Product Quality Requirements and Evaluation" is currently being developed to combine ISO 9126 and ISO 14598 (Suryn et al., 2003).

Regarding evaluation techniques, Punter (1997) argues for the use of weighted checklists, where the presence or absence of a number of agreed features is indicated and integrated into an overall score. Checklists are easy to customise and are a transparent, reproducible method of evaluation. A problem is the choice of items on the list, that is, the identification of the quality model. Punter argues that the only way to make this choice less subjective is to document and justify it extensively. In particular, each item on the list must be clearly related to the characteristic or aspect of the software whose quality it is supposed to indicate.

For COTS software, Carvallo et al. (2004a) and Botella et al. (2002) have suggested a process to refine the ISO 9126 standard characteristics, to arrive at a quality model for evaluation. Even a tool has been developed which supports this process and provides a formal model of the resulting quality attributes (Carvallo et al., 2004b). However, the actual identification of basic attributes is still left to the evaluator in this process.

A more empirical approach of how to arrive at a quality model (thus at items on a checklist, or at evaluation criteria) is given by Brown and Wallnau (1996). These authors suggest to identify those features of a technology that distinguish it from existing technologies. The authors call such distinctive features "technology deltas". Thus they stress that a product should be evaluated with respect to competitive products. This method ensures that no quality attributes are overlooked by the evaluators. It is particularly useful for functional criteria. Secondly, Brown and Wallnau (1996) stress that the technology deltas should be evaluated in well-defined, sharply focused usage contexts, because then the extent to which a technology delta supports a given context can be evaluated. The importance of distinctive features is supported by Maiden et al. (1997), who found that they costly evaluated some requirements which were, in the end, met by all candidate products among which they had to select. These authors also point to the usefulness of test cases, in terms of which the requirements can be stated. The present work includes a test case that can be used for that purpose, see section 8.4.

The distinction made at the beginning of this subsection, between the development of a software and its use as a product, serves the clarity of description but does not imply that there are no connections between these aspects. Obviously the quality of the source code and the development process influences the quality of the finished product; hence, some research exists that addresses these connections. For example, Punter (1997) stresses that the results of a software product evaluation are interesting for the developers of the software as well as for the potential buyers. Mayrand and Coallier (1996) and others relate the internal design of software to some external quality attributes. Similarly, April and Al-Shurougi (2000) map features that are based on the source code of a software to the ISO 9126 characteristics.

As regards metrics (see the second phase of the standard evaluation process above), obviously no internal, source-code related metrics can be used for COTS products (Colombo & Guerra, 2002). Previous research on product metrics has mainly concentrated on such internal metrics (e.g. (Mayrand & Coallier, 1996; Cartwright & Shepperd, 2000)). Research on COTS evaluations has concentrated on process-oriented aspects (Maiden et al.,

1997; Carvallo et al., 2004a) but has not established quantitative metrics, except for Rangarajan et al. (2001) and Colombo and Guerra (2002). In (Rangarajan et al., 2001), rather general metrics are given, only some of which are external, but require much effort to measure (such as the percentage of design goals met by the finished product). In contrast, Colombo and Guerra (2002) seem to use a metric similar to the one developed in the present work (section 8.2.2), but no details are given, nor any examples from a concrete evaluation project.

It can be seen that the goal in software product evaluation is not to arrive at one single metric that indicates the quality of a software, as the notion of quality is too complex for this; rather, the derivation of a detailed picture involving different aspects of quality, some of which can be in conflict with each other (Barbacci et al., 1995), is recommended. Though scores from a checklist can be integrated into a single value if desired, usually this is not the goal of an evaluation. Instead, the complete checklist scores are needed to arrive at an informed opinion about a product. Section 8.2 explains how the evaluation of KDD software products was performed for the present work, in the light of the guides cited above.

8.1.2. KDD product evaluations

The earliest comparison of KDD systems known to this author can be found in (Matheus et al., 1993). It is a study that compares three systems with respect to an early model of major components a KDD system should have. The components are: the interface to a database, a domain knowledge base, a "focusing" component used for data selection (the predecessor of data preparation), a pattern extraction component (providing the mining algorithms), an evaluation component and a controller module for interaction with the user. The three systems are analysed with respect to the extent to which they include (the functionality of) these components, so only rather coarse criteria are used. The evaluation is done by textual description.

One of the first attempts to evaluate KDD tools more systematically is (Abbott et al., 1998). This evaluation is based on a given application purpose (fraud detection). In order to handle the large number of tools then already available, the authors applied a three-stage approach. In the first stage all tools were evaluated under rather broad and simple criteria, such as support for the intended system environment, or range of algorithms provided. This stage left 10 products for the second stage, which filtered 5 products for the final examination using the additional criteria quality of technical support, and exportability of models, for example to source code. The last criterion relates to the deployment phase in KDD (compare chapter 2).

In the final stage, five tools remain and are examined under five well-discussed criteria. These are: (i) support for client server settings, which the authors deem related to scalability; (ii) automation of parameter search and documentation of experiments; (iii) range of algorithms and options offered for each algorithm; (iv) ease of use in data manipulation, mining, visualisation and technical support; and (v) accuracy of neural nets and decision trees on a dataset from the authors' application. Mainly points (ii) and (iv) are of relevance for this work. Concerning data preparation, they distinguish between loading the data and manipulating it. During data load, automatic recognition of data types and naming of attributes is an issue. This criterion is taken up in section

8.3 (criteria 14 and 19). Data manipulation is not discussed to a great extent, only the availability of built-in functions for attribute derivation is briefly discussed.

Among their lessons learned is the requirement to define what a tool is going to be used for, in order to focus the evaluation. Reasonable though this is, it is not applicable in this work, which attempts to find application-independent criteria. More relevant is their suggestion to test a tool in the environment where it is going to be used; as all the criteria listed in section 8.3 are based on the experiences made with the different tools when implementing the model application described in chapter 5, this requirement is fulfilled in the present work.

Another early attempt to give a systematic overview of different KDD software tools is (Gaul & Sauberlich, 1999). They consider the whole KDD process insofar as they examine only tools that offer some data preparation and deployment facilities, not only mining features. They list 16 tools and give the following features for them: manufacturer, available mining algorithms, system platform, price, year of first version, support for parallel environments, and limitations on data set size. For 12 out of the 16 tools, they give some further information in a second table with boolean entries indicating presence or absence of certain features. Concerning data preparation, they only consider the presence or absence of the operators Missing value replacement, Attribute derivation, Attribute selection and Scaling (see chapter 3), plus some unexplained operator Standardisation. Concerning deployment, they consider exportability and visualisability of models. The closest they come to conceptual aspects is the presence or absence of graphical user interfaces.

A more extensive list of classification features is provided by Goebel and Gruenwald (1999). These authors discuss three groups of features: general product characteristics, database connectivity, and data mining characteristics; they give tables with information for each feature for 43 tools. Of certain interest for this work is their stress of the importance of database connectivity. They claim that a KDD tool ought to be tightly integrated with database or data warehouse systems. Indeed, the large volumes of data typically involved in knowledge discovery make this issue paramount for modern KDD software. Thus they consider the data formats a tool can access, in particular certain file formats and databases, as well as data models (relational vs. single table), query options (SQL for databases, or GUI support), data types supported, and size limitations on the data set.

Concerning data mining characteristics, they distinguish between tasks such as clustering or prediction, and methods to solve the tasks. Data preparation is only considered by a single boolean flag indicating whether a tool has any preparation facilities at all.

A paper that considers data preparation features of software tools in some more detail is (Collier et al., 1999). This paper is also interesting in that it suggests a simple methodology to choose a most suitable tool from a list of tools, using a weighting scheme. While the authors do not relate their methodology to standard software product evaluation methods, see section 8.1.1, it is easy to see that their weighting scheme corresponds to the written procedure that prescribes the assignment of values for a metric, in phase 2 of the standard evaluation process according to ISO 14598. Though such a scheme is not new, the authors applied it to knowledge discovery software for the first time. The authors point out that the investigation of some effort into the choice of a suitable tool

will pay off easily, considering the work saved later in the application. Indeed, the total costs of ownership (TCO) of KDD software are hardly influenced by the licence fees, but much more by how much expert work the software can save.

Collier et al. (1999) also apply tool selection in two stages, filtering the bulk of tools away in the first stage under simple but hard criteria, such as support for the intended system environment. The second stage is more refined in their approach, however. Having grouped selection criteria into five groups (performance, functionality, usability, data preparation, other), they assign weights to the criteria in each group such that the sum of weights within a group equals 1.0. The groups themselves are also assigned weights. The authors then propose to choose one of the candidate tools as *reference tool*; one could choose a personal favourite tool based on past experiences of some of the evaluators, but any candidate can be used for reference. Then, each tool is given a score in each criterion that measures its strength *relative* to the reference tool. The score is assigned by human evaluators who have some experience with the tool. The reference tool gets a medium score in all criteria. Finally, the weighted scores of all tools imply a ranking for tool selection.

Focusing on these authors' data preparation criteria, they use mainly the presence and quality of the following data preparation operators: Value mapping, Row selection, Discretisation, Attribute derivation, and Missing value replacement. The brief discussion points out that an extensive list of functions is needed for Attribute derivation. Also, exportability of models is a criterion. Finally, one interesting criterion is called *Metadata manipulation*; it assigns a score based on the availability and manipulability of data descriptions and data types. Section 8.3 will develop rather more detailed criteria based on the ways of handling metadata supported by a tool.

A thorough study on data mining software solutions is the book by Gentsch et al. (2000), which provides detailed descriptions of 12 tools. For direct comparison, this study considers seven rather broad criteria that summarise the detailed descriptions. These are data import, data transformation (preparation), mining methods, visualisation of data and models, handling (usability), documentation, and special aspects (strengths of each tool in areas not covered by the other criteria, such as integration with other tools, code generation from models (criterion 3 below), etc.). Data import is related to the support of data types as discussed in section 3.3.1. The authors stress the importance of data preparation and mention the preparation operators that each tool provides in their detailed descriptions. They consider Attribute derivation, Value mapping, Aggregation, Scaling, and Missing value replacement. However, the discussion of preparation operators is not done in a systematic way, as it is not based on a (minimal) list of operators. Because the study comprises the whole KDD process, data preparation is just one aspect and is not discussed in any detail, though its importance is pointed out clearly.

Another list of criteria is suggested by Giraud-Carrier and Povel (2003). While an evaluation based on the criteria is not included in the paper, the criteria list is rather extensive. This discussion focuses again on the criteria related to data preparation. Their criteria include the presence or absence of facilities for: reading data from flat files, databases or XML files; data characterisation by statistical measures; data visualisation; row selection; attribute selection; and data transformation, under which point any other

preparation operators seem to be subsumed. Data cleaning (outlier detection) is also mentioned but not included in the final criteria list.

An example from a slightly different field is (Maier & Reinartz, 2004) which examines web mining tools. When mining data from web server logs, special preprocessing operations are needed to bring the data into attribute-value format, which is the input for data preparation as discussed in this work. The availability of some such preprocessing operations is included in the criteria list set up by Maier and Reinartz (2004).

8.2. Methodology

Several methodological deficiencies can be recognised in the previous work as discussed in section 8.1.2:

- The evaluations do not follow an accepted, standard evaluation procedure, nor do they use standard quality characteristics or concepts.
- The list of evaluation criteria is not justified in a systematic fashion, and is often rather short.
- Many approaches use boolean criteria which, on the one hand, often subsume many important aspects under one yes/no-flag, while on the other hand an overview is hard to keep if there are many criteria.
- No metric to flexibly quantify the degree to which a tool fulfils the criteria is given.
- No detailed methods prescribing how to apply the criteria to new tools are given.

This section explains the methodology used for tool evaluation in this chapter, which

- employs the conceptual level introduced in section 2.2 to abstract from technical details, thus allows to compare all criteria across tools and applications easily;
- follows the ISO 14598 standard of a software product evaluation process, but adds some aspects to it;
- systematically develops a list of evaluation criteria by following the notion of "technology deltas" by Brown and Wallnau (1996), see section 8.1.1;
- introduces *n-of-m* criteria as a concise, quantitative metric for complex quality characteristics, where the assignment of values can be done objectively and reproducibly;
- is adaptable to various levels of detail, thus to various audiences;
- uses all evaluation criteria found in previous work, and adds many more;
- is independent of human subjective evaluation;
- considers the complete KDD process;

- employs the list of operators from appendix A as another source for systematic evaluation; and
- provides a test case that allows a step-by-step evaluation of all criteria on new tools.

In the following, the methodology is developed following the four phases of the standard product evaluation process introduced in section 8.1.1. See also (Euler, 2005a).

8.2.1. Establishing evaluation requirements

The ISO 14598 standard requires the specification of the purpose of the evaluation, the type of products to be evaluated, and the quality model in this phase. The purpose of the evaluations in this chapter is to provide a detailed, yet clear picture of the strengths and weaknesses of currently available software tools that support KDD applications. It is not the purpose to test any software, nor to evaluate the tools under general software criteria such as reliability, portability or maintainability. Nor is it the purpose to select a single best tool or to give recommendations about tools; rather, a general framework is developed that allows the evaluation of further KDD tools easily.

The evaluation is restricted to such KDD products that include strong data preparation facilities, but cover the complete KDD process, and provide at least some conceptual support as discussed in previous chapters. Tools that offer only mining algorithms, with little or no data preparation, are excluded.

The quality model used in this work follows the purpose of the evaluation. The strengths and weaknesses of a tool are examined in the light of the conceptual aspects developed in previous chapters, which are in fact KDD-specific. Thus *only functional* criteria are applied. Hence, all the criteria used in the quality model here, which are listed in section 8.3, bear on the quality characteristic "Functionality", in particular its subcharacteristic "Suitability", in that they are used to examine the capability of the software tools to provide the set of functions that have been found to be appropriate for KDD tasks and objectives in the previous chapters.

The development of the criteria list followed the idea of technology deltas introduced in (Brown & Wallnau, 1996). This approach is particularly useful for functional criteria. Though the present work does not use the history of a technology to identify new, distinctive features, as Brown and Wallnau have done, it compares *features* of different products in order to identify the distinctive ones. A feature is deemed *distinctive* if it is present in one or more tools, absent in one or more other tools, and considered useful in the sense that it supports some of the conceptual aspects developed in the previous chapters. In this way a list of criteria is gained that provides a maximum amount of information when comparing the tools based on them. In a few cases, the inspection of distinctive features leads to the discovery of a few more *desirable* features that are not present in any tool examined.

Choosing the granularity of features is an issue. In some cases, one tool may provide a group of related functionalities that the other tools do not offer at all. For example, MiningMart is the only tool that uses the estimation of data characteristics. In such cases one could see a large number of distinctive features (estimation of value lists, estimation

of output size, ...) that only this one tool exhibits. However, it is a better contribution towards a clear comparison if only *one* distinctive feature that represents the whole group of functionalities is introduced in such cases. In other words, the features should only be as fine-grained as necessary to be distinctive.

In line with the conceptual approach of this work, the evaluation criteria address those functionalities of a KDD tool that are explicitly supported in the user interface. For example, some tools offer a scripting language that enables the execution of a graphically modelled process from outside the tool. The power of the scripting language can sometimes be exploited to achieve some functionality that is not offered in the user interface, for example the automatic testing of parameter settings (see section 2.1.4). However, in such a case, the criterion is not considered fulfilled because no high-level support is given for this functionality. The aim of this chapter is to provide measures for the conceptual support in KDD, that is, for the potential of a tool to save user efforts, and low-level programming is likely to require rather more than less user efforts.

8.2.2. Specification of the evaluation

Having found the quality model in the previous phase, each of its criteria is now assigned a metric, in the sense defined in ISO 14598 (see section 8.1.1). During work with the various KDD tools, most of the technology deltas identified corresponded to rather small, specific features, which are present in some tools and absent in others. A simple metric would assign a boolean value to each feature, indicating either its presence or absence. This would lead to a very long list of criteria, counteracting the evaluation goal stated in the previous phase of providing clear overviews of each tool's strengths and weaknesses. However, many small groups of features were found to be related in a rather natural way. Therefore, such naturally related features are grouped together in this work, and each group forms a *criterion*. The *n-of-m* metric is used to indicate the strength of a tool with respect to such a criterion: $m > 0$ is the number of features grouped together for this criterion, and n ($0 \leqslant n \leqslant m$) is the number of features that are present in the given tool. Thus each n-of-m criterion could be transformed into m boolean criteria. A simple score can be assigned to each tool under each criterion, which is the real value $0 \leqslant n/m \leqslant 1$.

This method allows much flexibility concerning the groupings of the basic features. For a quick overview or superficial comparison, only the more important features can be used, or larger inherently related groups can be formed. This corresponds to larger average values of m. For detailed surveys, like in this work, more fine-grained criteria can be used, so that the list of criteria is longer but the average value of m is lower. Thus the n-of-m method is adaptable to different granularities of detail, leading to different representations of the same evaluation scores. The different representations can be used for different audiences, like technicians or developers compared to decision makers. Section 8.6 provides two representations of the evaluation data collected for this work.

The measures for several single criteria can be combined to more integrated scores by building weighted sums, where the sum of the weight coefficients should be 1.0. For example, to assess the strength of a tool in data modelling, all criteria listed in section C.2 can be evaluated and combined to a single value. If desired, a single global score could be computed for every tool to get a ranking of the tools, though such a ranking would hide many aspects that the detailed score list can provide.

Some features could not be related to others and are listed as boolean criteria. These features should take one of the values 0 or 1.0 in order to be integratable with other criteria.

Though the above metrics are recommended for the type of criteria in this work because they are simple, transparent, and easily combinable, other scoring methods are applicable based on the given criteria list as well. For example, the method by Collier et al. (1999), described in section 8.1.2, can be applied as well as a simpler scoring method described in (Maier & Reinartz, 2004). Since each evaluator is likely to have their own priorities with respect to their application, the choice of the scoring method is open in this methodology. In section 8.6, which presents the results of some evaluations done for this work, the recommended metrics above are used.

The methodology described here results in objective criteria, with a written procedure that prescribes how to identify the presence or absence of each feature in a criterion. The procedures are given with each criterion in section 8.3, fulfilling the demand of objectivity and reproducibility. Further, a test case is provided in section 8.4 that provides clear explanations about how to evaluate each criterion based on a concrete example.

Though the methodology sketched here relies on inter-product comparisons for the development of criteria (see previous phase), it provides a set of criteria that can be applied to single software products, in contrast to the method by Collier et al. (1999) which is described in section 8.1.2, and which relies on inter-product *scores*.

A limitation to this methodology may be that, when applied to a different type of software products, not all technology deltas might correspond to boolean features that can easily be grouped. Some features, such as performance-related features, require a real-valued, continuous scale. However, such metrics can be mapped to the real interval [0..1] easily, which makes them easily combinable with n-of-m metrics. A more serious limitation is that different n-of-m criteria can result in identical values when evaluated, although the respective values of n and m are different. It is not clear whether the fulfilment of 2 out of 4 features of a criterion "means" the same strength as the fulfilment of 4 out of 8 features. Further, the features within a criterion are not weighted or prioritised here, though this could be added easily. However, to compare the tools under any given criterion, the same value of m is always used, so that the metric is valid.

8.2.3. Design of the evaluation process

The initial experiments for this work were done by implementing the model application described in chapter 5 in a number of tools. As the model application is based on two complex real-world applications, profound experiences could be made about a large number of issues that typically arise when realising complicated KDD processes, and about how different features of the tools support the implementation. This allowed to identify the technology deltas and develop the criteria as explained above.

However, now that a list of criteria is available, a simpler evaluation plan can be given. Section 8.4 describes a procedure to implement a test case in an arbitrary KDD tool and check various criteria in every step of the procedure. All criteria are covered. This corresponds to an evaluation plan, though elements like resource assignment are missing, as they are not applicable: the evaluation can be done by a single evaluator, and does not consume big computational resources. Hence, no team coordinations or fixed schedules

are needed. The main costs are likely to be incurred if the evaluator is new to the tool to be evaluated. In this case, the average time the evaluator needs to find out whether and how the given tool supports a functionality that is examined in a certain step of the test case will dominate the overall costs. This situation can be different, though, if external stakeholders (paying clients, for instance, who impose deadlines or other resource restrictions) need to be taken into account when executing the plan.

8.2.4. Execution of the evaluation

Executing the evaluation consists of following the execution plan, taking the measurements required by the criteria, and documenting them. The results of several such evaluations performed for the present work are presented in section 8.6.

8.3. Criteria for KDD tool evaluation

This section presents the criteria that were developed following the methodology described in section 8.2. As explained there, each criterion is accompanied by a precise description of how to evaluate it in an arbitrary KDD tool. This serves not only tool selection by end users but can also provide guidelines for developers of new tools. No tool covers all aspects discussed in this section; rather, the elaborations here can be seen as describing an "ideal" tool, towards which existing solutions should be developed.

This section first discusses some criteria whose detailed examination is excluded from this work, in section 8.3.1. This is followed by a discussion of some more general criteria, in section 8.3.2, which have been found in the literature on KDD evaluations (section 8.1.2), or have been mentioned in previous chapters. The relation of these criteria to the more detailed criteria that are based on the methodology used here is explained. Those more detailed criteria are listed in appendix C. They form a main contribution of this work.

For ease of reference, every criterion receives a number. The order of presentation of criteria is not significant. A list of all criteria with a reference to the page on which they are described can be found in appendix C on page 228.

8.3.1. Excluded criteria

As section 8.2.1 explains, only functional criteria are used in this work. From the perspective of the KDD process, only criteria pertaining to the more technical KDD phases data understanding, data preparation, mining and deployment are developed, as the conceptual support approach concentrates on these phases, while business understanding does not lend itself so well to conceptual modelling (compare section 4.5, and also section 6.6).

One important aspect of KDD tools concerning the mining phase is obviously the range of learning algorithms they provide, as well as the range of parameters that can be set for each algorithm. Yet, no minimal or complete list of algorithms, or even parameters for one algorithm, can be identified, because the sets of algorithms and parameters are open and likely to be extended by research progress in the future. Even today no single tool offers all varieties of algorithms that have already been described in the literature. Approaches to include the range of mining algorithms could perhaps be based on an ontology of

mining tasks and algorithms, such as the one given in (Cannataro & Comito, 2003), but there is no accepted standard ontology yet. Therefore, the range of learning algorithms and parameters, which has often been used as an evaluation criterion in previous work (see section 8.1.2), is not used as a criterion here.

In spite of this, the methodology developed in this chapter is also applicable to the mining phase. The approach used here is to judge the extent to which a mining tool supports basic, mining-related processing and control steps such as automated parameter search, cross-validation, or ensemble learning with arbitrary base learners. These concepts are explained in sections 2.1.4 and 4.5. However, it has to be said that not many tools offer strong coverage of such conceptual aspects of *both* the data preparation and mining phase. Therefore, separate evaluations might be appropriate for each phase.

Some studies from section 8.1.2 have used the accuracy of learned models on a given data set as a criterion to compare KDD tools. However, a ranking of tools based on one data set is not necessarily similar on a different data set, which is why model performance related criteria are not used in this work.

An important criterion in practice is execution speed. Despite similar architectures, different tools can reveal substantial differences in terms of processing speed. Since speed is highly dependent on the hardware infrastructure used, actual performance times are of little worth, but the ranking of tools that they imply can be expected to be consistent across platforms. This criterion does not concern a conceptual, functionality-related feature, but is directly related to the ISO 9126 subcharacteristic "Time behaviour" of characteristic "Efficiency". Therefore it is not used in this work.

Rather detailed criteria might be developed concerning the visualisation of data sets, data characteristics and learned models or functions. Many tools that were examined offer some visualisation features, but they are difficult to compare as each tool has its particular emphasis on certain visualisation methods. Visualisations of models or learned functions are not comparable if a tool lacks the mining algorithm whose visualisation is the strength of another tool. For data sets with more than three attributes, any visualisation of the data must include a dimensionality reduction, which is useful for human understanding but not necessarily helpful for mining. Further, visualisation of data sets and data characteristics mainly belongs to the data understanding phase of the KDD process, while this work focuses on aspects related to data processing. For these reasons, visualisation issues are not included here. Similarly, reporting functionalities, which some KDD tools offer to ease the production of documents reporting on the results of a KDD process, are not examined in this work. This includes facilities to draw charts based on mining performance or similar, statistical data produced during a KDD application.

A set of criteria that is left out from this section concerns the general software quality characteristics which are not specific to KDD tools, in line with the purpose of this work as described in section 8.2.1. This does not mean that general software quality issues are irrelevant for KDD software, only that they are not in the focus of this work. Criteria related to these issues can be found in the literature discussed in section 8.1.1.

Finally, as a related point, recall from section 8.2.1 that any functionality listed in the criteria below can not be fulfilled by low-level constructs such as integrated programming languages, but must be explicitly provided in the user interface in order to count as conceptually supported.

8.3.2. General criteria for KDD software

This subsection lists some criteria for KDD software that can be found in the literature cited in section 8.1.2, or in the previous chapters, but are not directly included in the list of detailed criteria developed in this work. The purpose of this subsection is to relate these more general criteria to the detailed criteria where possible, in order to ease the recognition of criteria known from the literature, or known from previous chapters, as the terminology cannot always be identical. Note the remarks on excluded criteria in section 8.3.1, though. This work's detailed criteria are given in appendix C.

1 Adaptability: The importance of this criterion has been stressed in chapter 6, in particular section 6.6. As discussed in that section, mainly the addition and deletion of conceptual metadata, as well as the *propagation* of such changes, must be supported. In general, adaptation and reuse are easiest if the software follows the two levels approach proposed in section 2.2 in the area of data sets. Therefore this criterion is related in particular to criteria 15, 17, 18, 19, 25, and 27.

2 Scalability: KDD software has to be capable of handling large data sets. Some studies in section 8.1.2 have included limits on the number of attributes or rows that a software can process as a criterion. However, the tools examined for this work do not explicitly state such limits, so that any such limitations depend on hardware resources. Other studies have stressed the importance of support for parallelisation or for client-server environments, where the server deals with large data sets and control is executed from the client. Criterion 8 is the main related criterion in this work.

3 Interoperability: If users want to use special software for some subtask, such as reporting or mining, they should be able to do so easily. In KDD, the interface to other systems is often a data set on the file system or in a database, so that this criterion is related to the data formats a tool supports (criterion 7); yet if mining results (like learned rules) are to be used outside the tool, integration with other software can be more difficult. See criteria 50 and 55.

4 Guide to KDD process: The software should offer some support to guide the user through the complex stages of a KDD process, and avoid erroneous user inputs. This support should be offered for several levels of assumed previous experiences of users. Related criteria are 26 and 54.

5 Documentation: It should be possible to add free text comments to every object involved in the KDD process. As all tools examined for this work offer facilities for this, it is not a technology delta but is obviously very important, especially for the reuse of process models (see chapter 6).

6 Business problem: Some approaches have attempted to use the types of business problems a tool can solve as a criterion. In the absence of a theory on how business problems are related to mining tasks (see section 2.1.4), they have used very simple

mappings of stereotypical business problems to mining tasks, so that this criterion is related to the range of mining algorithms, which is not discussed in this work as explained in section 8.3.1.

8.3.3. Specific criteria for KDD software

Appendix C lists the criteria that were developed following the methodology described in section 8.2. They are categorised into a number of areas. As section 8.2.2 explains, a criterion consists of m boolean features (questions), but in principle the number (m) and the exact grouping of questions for a criterion is flexible. Given all features, different groupings into criteria can be formed to reflect purpose-specific aspects. One may also choose to leave out some features with low priority. Since priority is application-dependent, no weights are given to features or criteria in this work, but the grouping of features into criteria here is a recommendation based on experiences made during the implementation of the model case. Further, this particular listing of criteria allows a quantitative, detailed, yet clear comparison of KDD tools, as demonstrated in section 8.6. Finally, the test case described in section 8.4 is designed to check exactly these criteria.

8.4. A test case to check all criteria

In this section a test case is provided that is as small as possible but still enables to check all criteria from appendix C, given a KDD tool. The test case describes a small KDD process. Its implementation is described step by step, with reference to every criterion that is tested in each step. The case can be seen as a baseline scenario whose implementation should be possible in any KDD tool, but perhaps with varying difficulty. It can be implemented in less than an hour, thus giving an effective method to objectively evaluate a KDD tool in practice, under the criteria given here. It corresponds to a detailed evaluation plan as explained in section 8.2.3. Since every step of the test case concerns particular criteria which are given in the description below, some steps can be omitted if the criteria tested there are known to be less important for the particular evaluation purpose.

Appendix E (page 244) provides an SQL program that realises the test case, to give a formal reference, while 8.2 shows a graphical overview of the case as realised with MiningMart.

The order of steps in this test case is based on the data flow that is modelled, rather than on the order of the criteria. An interesting alternative would be to order the test case such that those criteria which appear to be most challenging are tested last. This would render a single score for each tool that is evaluated, namely the point in the test case at which it cannot support the tested functionality any longer. To capture the notion of difficulty, or how challenging a criterion is, the number of tools evaluated in this work that fulfil each criterion can be used for ordering the criteria. However, an implementation of the test case that follows the data flow is easier to describe, understand and realise.

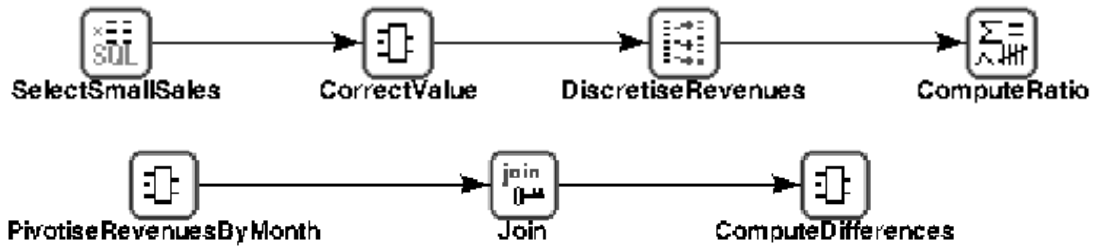

Figure 8.2.: Overview of the steps of the test case.

Table SalesData:

EmplId	Month	Sales	Profit
1	1	3	40.5
1	2	2	22.8
1	3	-1	10.0
2	1	5	54.2
2	2	7	58.6
2	3	4	41.0
3	1	-1	10.0
3	2	2	38.1
3	3	4	44.3

Table EmployeeData:

EmplId	EntryDate	Position
1	02-12-1988	Senior
2	01-06-1998	Trainee
3	01-01-1990	Senior

Figure 8.3.: Input data for the test case.

The data

Figure 8.3 shows two small data sets which are the input to the test case. These data sets can be easily provided as flat files, with any file format that is deemed relevant, or as database tables. To test criterion 7, the data sets are imported into the tool (to test the output facilities, they can be written from the tool to different files/tables as well). At this point, already a number of other criteria can be checked. An obvious and important criterion is whether the tool models the data explicitly, in a graphical way (criterion 24). If the data sets can be displayed inside the tool, criterion 13 is fulfilled. Criterion 14 lists recommended facilities for attribute import. Also it can be seen whether conceptual data types are used in the tool (criterion 15), and if they are correctly recognised (criterion 16); for example, is the column `EntryDate` automatically given a correct technical and conceptual data type (e.g. *Date*)? Can the data types be changed? Can automatic recognition of types be deferred to a later point in time? The recognition of data characteristics can also be tested (criterion 19).

A different approach is to test criterion 34 by attempting to set up data models (concepts) without actually importing data.

Data preparation

After these preliminaries, the first processing step is a ROW SELECTION, applied to SalesData. Its output concept contains only the rows with `Sales` $\leqslant 5$. Is the output explicitly modelled, and clearly related to the input, as demanded by criterion 24? Are the data characteristics of the output concept available without executing the operator, in particular, is the highest value of `Sales` adjusted to 5 (criterion 25)? Changing the selection condition to `Sales` < 0 can be used to test criterion 35 (empty data set recognition) after executing the operator.

The next operator corrects the values of `Sales` by replacing all occurrences of -1 (taken to be missing values or typos) with 0. The operator VALUE MAPPING should be available, but if it is not, ATTRIBUTE DERIVATION can be used with an if-then-else type of derivation formula. The values of the `Sales` attribute must be available in the graphical user interface when specifying the parameters for VALUE MAPPING (criterion 20), preferably without having executed the previous operator (criterion 25).

The next operator discretises the `Profit` attribute of SalesData into two categorical values. The operator DISCRETISATION (using a given number of intervals) should be available; otherwise ATTRIBUTE DERIVATION must be used. Criterion 28 tests whether the discretisation formula used by DISCRETISATION is accessible and changeable (after execution of the operator). The two categorical values created by DISCRETISATION should also be changeable: if yes, they are set to 0 and 1 now, if no, an extra VALUE MAPPING step is inserted to do so. The technical and conceptual data types of the discretised result must be created by the system (criterion 23).

The fourth operator computes the ratio of the higher of the two intervals formed in the previous step. This is realised by AGGREGATION, where the *Group By* attribute is `EmplId`, and the average function is used as aggregation function, applied to the discretised attribute. This tests criterion 18 (robustness of type mapping), as the categorical values 0 and 1 created in the previous step are used as real numbers here. If this usage (and thus the criterion) fails, criterion 17 can be tested by attempting to explicitly convert the data type.

These four operators can be executed in the order given here, as each operator's input is produced by the previous operator. The four operators could be collected in a chunk, testing criterion 38 (chunking support). There should be an option to view the data collection that results from this chunk (criterion 31). Criteria 40, 41, 42, and 43 (the execution-related criteria) can be tested by executing the four operators together. It can be attempted to vary the place of processing (criterion 8, data handling). Further, the caching-related criteria 9, 10, 11 and 12 can be tested, for example by checking whether the result of the execution is still available afterwards (11, automatic caching), or by trying to find out where the intermediate data was stored (12, caching transparency).

With this short chain, also the important criterion 27 about the propagation of conceptual changes can be tested. To this end, the second step is deleted. Is the attribute it creates (with the corrected values of the `Sales` attribute) automatically removed from the input and output of the following steps? If the step is added again, does the attribute show up in later steps automatically?

For further tests, a new short chain of operators is set up. The data from the SalesData table contains monthly information about the sales and profit that every employee

achieved. This data is converted to single-row information about each employee by a PIVOTISATION application, where the index attribute is `Month`, the pivotisation attribute is `Profit`, the aggregation operator is summation and the *Group By* attribute is `EmplId` (compare the example in section A.3.2). If this operator is missing, three ATTRIBUTE DERIVATIONs can be used, one for each month, followed by an AGGREGATION. By using the ATTRIBUTE DERIVATIONs, criterion 45 can be tested, as the three derivations are very similar. For example, in SQL, the three derivations would be `(CASE WHEN Month=`i `THEN Profit ELSE 0 END)`, with i ranging from 1 to 3, resulting in three new attributes which would be aggregated using summation.

The result of the previous step is now joined with the result of the first chunk (chain), using JOIN BY RELATIONSHIP (or a simple join), and testing criterion 22 (attribute matching) when setting up this operator. The key for joining is of course the attribute `EmplId`; can its key status be stated explicitly (criterion 15 about conceptual data types)?

To test criteria 32 and 33 about the support for and iteratability of ATTRIBUTE DERIVATIONs, a final operator is added which computes the differences between the profit achieved in the third month and those achieved in the first and second month. The derivation formula should iterate over the two fields with the profit for the first and second month. A further test of criterion 27 about the propagation of changes can be done now, by deleting the pivotisation step(s). The formula for the difference in this last step should then not be automatically deleted; instead, the last step (or its derivation formula) should be clearly marked as invalid (criterion 26, checking wellformedness).

Criteria 44 about the transparency of export files, and 46 about the arrangement of operator applications in a processing graph, can be checked using the whole test case.

Criterion 39 about the unrestricted structure of the preparation graph is missing so far; it can be tested by applying two ROW SELECTIONs to the same input concept, and then applying UNION to the two results. If this is possible, it follows – together with the above – that the tool allows any directed acyclic graph.

Mining

Finally, the criteria related to mining and deployment, which are all boolean, can be tested. No particular scenario is needed to test them; in most cases, the help system or the manual will be sufficient to decide whether the criteria are fulfilled. This is also true for some "static" criteria like 29, 30, 36, 37 or 54. For example, the availability of cross validation or model export facilities in a tool will surely be reflected in the documentation. The same is true for facilities to publish process models in a detailed way, based on criterion 53. One criterion to be included in this phase is criterion 21 about attribute roles; it should be possible to declare for each attribute that is present in the concept used as input for mining whether it is label or predictor, or whether it should not take part in the mining.

8.5. Evaluated KDD software

This section briefly describes the KDD software packages that were evaluated in this work. These tools were chosen according to their general strength in the conceptual

support of data processing, and they serve well to exemplify different aspects of many criteria.

8.5.1. MiningMart

MiningMart[1] is introduced in chapters 6 and 7 as a graphical front-end to relational databases that offers a broad range of KDD-oriented data preparation operators (Morik & Scholz, 2004). It leaves all processing to the underlying database system by translating the preparation graph to SQL commands. Such graphs, called *Cases* in MiningMart, can be exported and uploaded to a central web repository (section 6.5), where they are browsable and downloadable by anyone looking for example KDD applications. This is the only tool encountered during this work that uses an explicit representation of concepts and their links. Version 1.1 was used in this evaluation, which includes all features described in chapters 6 and 7.

8.5.2. SPSS Clementine

Clementine[2] is a tool intended to support all phases of the KDD process. It includes many data preparation facilities. It was used for this examination in the standalone version, thus entirely file system based, but a client server version is also available that can delegate some data processing tasks to the database server. Version 8.1 was used in this work.

Clementine allows to use abstract data types and attribute roles when dealing with the data to be prepared, but it does not use an explicit model of the data tables and how they are linked, as MiningMart does based on chapter 3. Clementine has many preparation operators, but like the other tools below, it lacks most of the operators that change the organisation of the data (section A.3). Without these operators the application from chapter 5, for one example, is inconvenient to realise.

8.5.3. Prudsys Preminer

Preminer, sold by Prudsys[3], is a specialised tool for data preparation that belongs to a family of products supporting the complete KDD process. Its architecture is different from the preceding two tools in that it uses an extra data server for intermediate storage of data. This enables the user to process data from heterogeneous sources using the same front end. For example, a data set from a text file can be joined with a database table (if the keys match). The evaluation in this work was based on Version 1.3. For evaluating the mining facilities the Discoverer module version 3.2 was used.

8.5.4. IBM Intelligent Miner

Intelligent Miner by IBM[4] is a group of products to cover data preparation, mining and deployment based on IBM's DB2 database. The graphical front-end is the Intelligent

[1] `http://mmart.cs.uni-dortmund.de`
[2] `http://www.spss.com/clementine`
[3] `http://www.prudsys.com`
[4] `http://www.ibm.com/software/data/iminer`

Miner for Data, whose version 8.1 was used for the evaluation in this work. While both flat file data and database tables can be input to mining, the data preparation operators can only be applied to database tables, as they are realised by SQL views, in a way similar to MiningMart (8.5.1). Also, learned models are available as DB2 procedures, which leverages their deployment on large data sets.

8.5.5. SAS Enterprise Miner

The Enterprise Miner is one of several analysis modules available in the SAS system[5]. The SAS environment is a powerful workbench for many aspects of data analysis. It offers client-server processing distribution as well as data warehousing support. The Enterprise Miner provides several mining algorithms and many data inspection facilities, though the latter can be complemented by other SAS modules. The Enterprise Miner includes some data preparation functionality, but its focus is on the mining step and on visualisations of data sets and mining results. Therefore it lacks many of the essential operators. They can be replaced by integrating small programs in the internal SAS language. However, as explained in section 8.2.1, such programming constructs do not support conceptual, high-level work, and the functionality they may offer is not seen as fulfilling any criterion. Version 4.3 of the Enterprise Miner was used in this evaluation.

8.5.6. NCR Teradata Warehouse Miner

The Warehouse Miner[6] by Teradata, a division of NCR, is a tool specifically developed to support mining Teradata databases. Apart from an ODBC interface, it can only be used on Teradata databases, from a Windows client. It leaves as much data processing as possible to the underlying database, issuing automatically created SQL statements in a way similar to MiningMart and the Intelligent Miner. It offers a number of operators for processing, but also relies heavily on SQL programming for some of the more complex operators (in which it resembles the Enterprise Miner by SAS). It does not use an explicit data model, nor does it display the data flow in a graph. Version 3.2 was used in this evaluation.

8.6. Evaluation results

This section provides the evaluation of the tools described in 8.5 under the criteria from appendix C. As explained in section 8.2.2, the list of criteria is amenable to several methods of scoring and weighting. In the evaluation in this section, each tool receives the score (measure) $0 \leqslant n/m \leqslant 1$, where $m > 0$ is the number of boolean features that make up a criterion, and $0 \leqslant n \leqslant m$ is the number of these features that the tools fulfils. Thus the m boolean features of a criterion are not weighted (prioritised) here, as a weighting would be very dependent on the intended application and environment for the KDD tools. Similarly, no weighting of the criteria themselves is used here.

[5] `http://www.sas.com`
[6] `http://www.teradata.com`

No	Name	m	MM	Clem.	Prem.	IBM	SAS	NCR
1	Data access	17	0.65	0.71	0.53	0.59	0.59	0.53
2	Data modelling	31	0.9	0.81	0.39	0.58	0.61	0.32
3	Preparation process	65	0.75	0.49	0.46	0.32	0.43	0.26
4	Learning Control+Deployment	6	0.83	0.5	0.33	0.67	0.5	0.33
5	KDD standards	4	0.25	0.75	0.5	0.25	0.25	0.25
	All features	123	0.76	0.61	0.45	0.44	0.5	0.32

Table 8.1.: A different representation of the data in table 8.2, using a coarser grouping of the 123 boolean features into criteria.

Table 8.2 on page 185 contains the scores for each criterion based on the list of criteria from appendix C. Table 8.1 provides scores which are computed based on the same list of 123 boolean features, but a different grouping into criteria, namely into fewer criteria using higher values of m. As section 8.2.2 explains, these alternative scores are a different representation of the same data that may be more suitable for certain audiences, for example for decision makers. The criteria in table 8.1 use the grouping into overview criteria that is indicated by section headlines in appendix C, and by horizontal lines in table 8.2.

A further table (table 8.3 on page 186) contains a detailed list of the preparation operators listed in appendix A that are available in each tool. To illustrate the effect of the availability of powerful operators, note that the test case described in section 8.4 required – without mining – 7 operator applications in MiningMart and the Teradata Warehouse Miner, 11 in Clementine, and 9 in Preminer and Intelligent Miner, respectively. In SAS Enterprise Miner the test case was only partially implemented, as this tool lacks the join operator.

8.7. Summary

This chapter has found a methodology for the comparison of software products that is suitable for judging the extent to which a tool supports the conceptual level of an application domain. The restriction to the conceptual level is done by taking only functionality into account that is based on notions that are explicitly represented in the user interface. This idea is part of the comparison methodology developed in section 8.2. A main aspect of the methodology is that it renders metrics that are adaptable to different evaluators or purposes. The methodology has been applied to the major current KDD software packages that support data preparation. A detailed criteria list, presented in appendix C, is one result. Section 8.4 shows how such criteria can be assembled into a tight evaluation plan by providing a small test application, here a small KDD process. The scores that the compared KDD tools receive under "neutral" (non-weighting) metrics are given as another result, in section 8.6. While they serve mainly as an exemplification of the methodology, they also indicate the different levels of maturity that the compared tools have achieved, as far as support of the conceptual level is concerned.

No	Name	m	MM	Clem.	Prem.	IBM	SAS	NCR
7	Data formats	6	0.33	0.83	0.5	0.5	0.66	0.33
8	Data processing	3	0.33	1.0	0.66	0.66	0.66	0.33
9	Caching control	2	1.0	1.0	0.5	1.0	0	1.0
10	Caching size estimation	1	1.0	0	0	0	0	0
11	Automatic caching	2	1.0	0	0	0.5	0.5	0.5
12	Caching transparency	2	1.0	0.5	1.0	1.0	1.0	1.0
13	Data inspection	1	1.0	1.0	1.0	0	1.0	1.0
14	Attribute import	3	0.66	0.66	0.66	0.33	0.33	0.33
15	Conceptual data types	1	1.0	1.0	0	1.0	1.0	0
16	Type recognition	5	0.8	1.0	0.4	0.8	0.8	0.2
17	Flexibility of type mapping	3	1.0	1.0	0	0.66	1.0	0
18	Robustness of type mapping	1	1.0	1.0	1.0	1.0	0	1.0
19	Data char. recognition	6	1.0	0.66	0	0.66	0.66	0.33
20	Data char. deployment	1	1.0	1.0	1.0	0	1.0	0
21	Attribute roles	4	0.75	1.0	0.75	0.75	0.75	1.0
22	Attribute matching	2	1.0	1.0	1.0	0.5	0	0.5
23	Data type inference	2	1.0	1.0	0.5	0.5	1.0	0
24	Abstract data model	2	1.0	0	0	0	0	0
25	Characteristics estimation	1	1.0	0	0	0	0	0
26	Syntactic validity checks	4	0.75	0.5	0.75	0	0.25	0.25
27	Propagation of changes	5	1.0	1.0	1.0	0.2	0.8	0.2
28	Operator transparency	2	0.5	0	0	0	1.0	0.5
29	Availability of operators	19	0.95	0.58	0.42	0.53	0.37	0.47
30	Assign operators to prep. tasks	1	1.0	0	0	0	0	0
31	Intermediate views on data	1	1.0	0	0	0	0	0
32	Attribute derivation support	2	0	1.0	1.0	0.5	0.5	0
33	Iteration attribute derivation	3	0	0.33	0	0	0	0
34	Independence from data	1	1.0	1.0	0	0	0	0
35	Empty data sets recognition	1	1.0	0	0	0	1.0	0
36	Representation of data flow	1	1.0	1.0	1.0	0	1.0	0
37	Pseudo-parallel processing	1	1.0	0	0	0	0	0
38	Support for chunking	2	1.0	0.5	0	1.0	1.0	0
39	Graph structure	1	1.0	1.0	0	1.0	1.0	1.0
40	Execution transparency	7	0.71	0.14	0.43	0.29	0.43	0.14
41	Execution automation	3	0	0.33	0.66	0	0	0
42	Execution administration	7	0.71	0.29	0.43	0.29	0.57	0
43	Execution in background	1	1.0	1.0	1.0	1.0	0	1.0
44	Export transparency	1	1.0	0	1.0	0	1.0	1.0
45	Editing flexibility	1	0	1.0	0	1.0	0	1.0
46	Visual graph arrangement	1	1.0	1.0	1.0	0	0	0
47	Splitting training and test set	1	1.0	1.0	1.0	1.0	1.0	1.0
48	Model evaluation	1	1.0	1.0	1.0	1.0	1.0	1.0
49	Mining subprocess support	1	0	0	0	0	0	0
50	Export of models	1	1.0	1.0	0	1.0	1.0	0
51	Deployment in databases	1	1.0	0	0	1.0	0	0
52	Post-processing	1	1.0	0	0	0	0	0
53	Published meta model	1	1.0	0	0	0	0	0
54	CRISP support	1	0	1.0	0	0	0	0
55	PMML support	2	0	1.0	1.0	0.5	0.5	0.5

Table 8.2.: Evaluation table. $m = 1$ indicates boolean criteria.

Operator	MM	Clem.	Prem.	IBM	SAS	NCR
Attribute selection						
– Manual selection	Yes	Yes	Yes	Yes	Yes	
– Automatic selection	Yes				Yes	
Row selection	Yes	Yes	Yes	Yes		
Sampling	Yes	Yes	Yes	Yes	Yes	Yes
Aggregation	Yes	Yes	Yes	Yes		
Discretisation						
– fixed no of intervals	Yes	Yes		Yes	Yes	Yes
– fixed width	Yes	Yes				Yes
– fixed cardinality	Yes	Yes				Yes
Scaling	Yes					Yes
Value mapping	Yes	Yes	Yes	Yes		Yes
Attribute derivation						
– String processing	Yes	Yes	Yes		Yes	
– Numeric arithmetics	Yes	Yes	Yes	Yes	Yes	Yes
– Date/time arithmetics	Yes	Yes	Yes	Yes	Yes	Yes
– Model application	Yes	Yes	Yes	Yes	Yes	Yes
Join	Yes	Yes	Yes	Yes		Yes
Join by relationship	Yes					
Aggregate by relationship	Yes					
Union	Yes	Yes	Yes			
Missing value replacement						
– By default value	Yes	Yes		Yes	Yes	
– By average or median	Yes	Yes			Yes	
– By learned function	Yes				Yes	
Filtering outliers					Yes	
Dichotomisation	Yes	Yes				Yes
Pivotisation						
– normal	Yes					Yes
– n-fold	Yes					
Reverse pivotisation	Yes			Yes		
Windowing	Yes					
Segmentation						
– By value	Yes					
– Randomly	Yes					Yes
– By learned clusters	Yes					
Unsegmentation	Yes					

Table 8.3.: Availability of preparation operators from appendix A for each KDD tool. No entry = not available.

9. Conclusions

This chapter summarises this thesis and points out its contributions to the state of the art (section 9.1), before discussing some ideas for future work (section 9.2).

9.1. Summary of contributions

Easing user efforts in the development and reuse of data preparation for KDD has been given as the overall goal of this work in section 1.2. Chapters 3 to 8 have contributed both theoretical and practical steps towards this goal, which will be summarised below. Almost all the contributions are centred on, or enabled by, the conceptual level that has been described for KDD applications in this work. The MiningMart environment provides the means to create, manipulate, exchange, and reuse KDD application models by using a metamodel designed to support the conceptual level.

The following paragraphs clarify the particular contributions of the author of this thesis, and point out the corresponding chapters of this work.

A data model for KDD

This work has defined an adequate way of conceptual data modelling for the area of knowledge discovery. This idea is a rather natural one in view of the many data-centred tasks during data preparation. It helps users in organising the mining process in domain-related terms. It goes back to the common knowledge representation language (CKRL) of the machine learning toolbox MLT (Morik et al., 1991), but today, abstract data models are still not used in KDD software, except in MiningMart. The data model in MiningMart is based on the work by Morik et al. (2001), and was refined by the author of this work in order to create an alternative, dual view on the KDD process (see below).

For the present work, the requirements for a conceptual data model to be useful for KDD have been analysed (section 3.2.1), and have led to the choice of the entity-relationship model as the basic model. This choice represents a balance between usability, which demands a clear and simple abstract data view, and the flexibility to model important semantic aspects explicitly. Another important requirement for the model was to allow to structure the intermediate data representations, since rather a lot of them are created during a typical preparation process, and it has been argued that they are useful artifacts of this process. This requirement motivated the use of two particular types of generalisation, namely specialisation and separation (section 3.2.1), because many preparation operators produce these links between their input and output, so that a web of (representations of) data sets emerges whose links reflect how the data sets are created from each other.

The author has implemented all functionality related to this data model in the MiningMart system. In particular, this involves creating concepts (data representations) and

the semantic links between them automatically as soon as the operator that will create the data is instantiated (section 7.1.1). This allows to switch to the data view at any time.

In order to allow the convenient use of the conceptual data models, a propagation algorithm has been designed to support the automatic adaption of dependent elements of the above-mentioned web whenever a data representation is edited, for example to reuse it on new data (section 7.1.2); a schema matching algorithm has been designed to connect an abstract data model to concrete data sets (section 7.1.4); and the estimation of data characteristics in the absence of actual data has been provided (section 7.1.3). These technical contributions support re-using KDD process models, which has been an important motivation for this work, as discussed in chapters 1 and 6.

Section 3.2.1 has also considered the idea of using more expressive ontology formalisms for conceptual data modelling. It was waived in favour of a meta model that would render clearer overviews of the web of data sets (compare figure 5.10 on page 88). However, more powerful formalisms have other advantages. This is discussed further in section 9.2.

Preparation operators for KDD

This work has specified a range of important preparation operators for knowledge discovery. The list includes all operators that have been used in the literature or in any KDD software. This work has identified five major high-level preparation tasks (section 2.1.3) and has associated each operator to one of them.

This list of preparation operators can serve as a reference standard for data preparation in KDD, and forms a major component of the conceptual level. Using the conceptual data model in the specifications of the operators allows to set up syntactically valid chains of data transformations; the validity checks reduce the number of test cycles needed during development. These validity checks are based on explicit pre- and postconditions of the operators. The operator specifications also allow the estimation of data characteristics of an operator's output before it has actually been computed. Here the present work has contributed ways of estimating not only the data size, as in previous work, but also other characteristics (compare section 3.3.3).

Dual views on the KDD process

The conceptual data model and the list of operators have been designed such that two views on the KDD preparation process arise, both of which provide the information about the structure of the process, but from different angles. Each view puts the focus on different types of additional information; one is data-centred, the other is based on the chains of operators. Changes to one view can be made visible immediately in the other. The MiningMart system currently offers complementary functionality in both views, but there is no principle that prohibits extending the options in each view such that complete control of the process can be offered in either of the views.

A single view with both types of information can be imagined, but would probably be graphically overloaded in complex applications; nevertheless, this idea has some advantages and is therefore discussed in section 9.2.

MiningMart

The MiningMart framework and system have been developed by a team of which the author of this thesis is a member. The contributions of this author include the development and implementation of all aspects related to the conceptual data model, see above. MiningMart is thus now the first system that supports the dual views. Several important operators have also been implemented by this author, the more interesting of which are described in section 7.2. In particular, the automatic translation of the results of mining algorithms into stored procedures for databases has been realised exemplarily for one complex mining algorithm (section 7.2.5). The current version of MiningMart's web repository of KDD models, and its indexes for case retrieval, are also the work of this author, see below. All in all, roughly 40% of the MiningMart code, as measured in lines of code, have been implemented by this author.

The MiningMart compiler creates database views that represent the output of operators; it has been developed by Martin Scholz (Scholz, 2007). It is complemented by a materialisation strategy developed by the author of the present work, which speeds up the execution of longer processes significantly, as shown by experiments described in section 7.3. This is important for handling large data sets.

Contributing towards the aim of reducing development time, especially on large data sets, some measures have been suggested, and implemented in MiningMart, which support developing at least parts of a preparation process using only the conceptual level, without requiring its immediate execution. Syntactic validity checks are possible because the conceptual data model includes data type information, and because output representations are immediately constructed when an operator is specified, as described in section 7.1.1. The validity checks themselves are based on declarative constraints, most of which have been developed by Martin Scholz. Also, online computation of estimated data characteristics (section 7.1.3), solely the work of the author of this thesis and independent of data processing, supports the independence of modelling from execution, by providing orientation as to the results of the path of preparation a user is currently working on. Further, the estimations are useful for the instantiation of operators whose output data schema depends on input data characteristics, see section 7.2.2.

Generally, MiningMart represents a general and advanced method of supporting KDD developers, especially during data preparation. Several ways in which MiningMart extends the state of the art can be identified:

- No other KDD tool today uses a two-level data model, with semantic links between intermediate data representations, to organise the data preparation. Thus no other tool uses dual views on a preparation process, either.
- No web portal for the exchange of KDD solutions had existed before the one for MiningMart was created (see below).
- MiningMart is the only KDD tool that is based on a public, freely available meta model. Other tools use proprietary, intransparent formats.
- MiningMart is currently the most suitable environment for the specification and reuse of general *patterns* among successful preparation processes, which can be formalised as *templates*. See below.

- No other KDD software offers the kind of pseudo-parallel processing that is available in MiningMart, whose usefulness is demonstrated by the example application from section 1.1.1.

- MiningMart offers the most comprehensive list of preparation operators found in any KDD software.

- No other KDD tool today includes specific measures for supporting the reuse of KDD process models, such as mapping a given data model to new data.

An evaluation of MiningMart has been done by a third party, a service providing company for telecommunications, who performed one of their large data mining applications by SQL programming, and then again using MiningMart (Richeldi & Perrucci, 2002b). The authors report that developing their application took 12 days of SQL programming, but only 2.5 days of modelling in MiningMart, for staff who was not familiar with MiningMart. The results in terms of the discovered knowledge are the same. Additional operators that can solve some of the tasks involved in this study more directly have been added to MiningMart after the study was completed, so that the development time for such applications can be expected to be even lower now. This is clear evidence supporting the claim that the goal of supporting human users during data preparation has been achieved.

A reusable model of a real-world application

This thesis includes the first detailed documentation of a complex data preparation process, modelled after two real-world knowledge discovery applications (chapter 5). An annotated, operational model of this process is available in the MiningMart web repository. Thus its technical details are easy to study for anyone. The model demonstrates the two dual views, as well as the difference between a conceptual-level model and a technical realisation in a formal language in terms of usability, maintainability and reusability.

Templates for data preparation

This thesis has argued that preparation processes from different KDD applications can have common substructures (see sections 6.5.1 and 6.6.2) which are used to solve similar or identical subproblems. Several such subproblems have been identified by the author of this work, and solutions for them have been created with MiningMart, have been documented and annotated comprehensively, and published as "templates" in the web repository (section 6.5.3). Some of the templates are based on previous (informal) work by other authors, but most were contributed by this author. The result is the first public collection of directly usable data preparation solutions, which is both a useful library for experts, saving the work to re-implement these solutions, and a helpful tutorial for less experienced analysts.

This work is also the first to suggest the *automatic* discovery of preparation subproblems that have been solved several times in a similar way. The basis for the proposed method that can achieve this goal is a collection of KDD applications modelled in the

same framework. While this work has created the infrastructure to get such a collection (the case base, see below), there are not enough application models available yet. Therefore, a frequent subgraph discovery algorithm tailored for this context has been developed and proposed in section 6.5.4, but has not been implemented yet. The algorithm can work on the level of operators, or on the more abstract level of operator groups, effectively using different similarity measures for defining the similarity of subsolutions.

Providing templates, whether automatically discovered or manually contributed, has been compared to providing design patterns in software engineering in section 1.1.1. An important difference is that the MiningMart templates are *operational*, so they can be applied to new problems and executed in the MiningMart system directly, whereas design patterns need to be translated to new problems by human experts, in a complex and error-prone process.

The case base

MiningMart has provided the first public infrastructure for the documentation and exchange of KDD application models, the web repository of KDD cases, or case base. These models can be closely inspected, in all technical details, using an ordinary web browser, without having to download them or having to install MiningMart (see section 6.5). The idea is that HTML pages represent elements of the conceptual level, such as concepts, attributes, or steps, and links between the pages represent how these elements are related. Creating such HTML files offline for a KDD model is a MiningMart functionality provided by this author, replacing an online version by Stefan Haustein. Only the direct reuse of the application models requires the MiningMart system. This work has discussed a number of ways to support users when searching for a suitable application model to reuse on their own problem (section 6.5.5). The most important means to this end is the documentation of each application with background information, organised into five topics. This information can be searched using any internet search engine.

In section 6.6, the tasks involved in reusing a KDD application have been analysed in more detail. It was noted that the deletion and addition of some elements from/to the conceptual model are central tasks because they affect not only the element where they are performed, but many dependent elements. For example, when an attribute is removed from a concept, it has to be removed as well from any copy of this concept created by operators anywhere in the process. The propagation algorithm for changes to a concept, already mentioned above and presented in section 7.1.2, thus provides important support for the reuse of KDD models.

For reusing a KDD application, its data representations have to be matched to the actual new data sets. This work has argued that not only the data sets that the original application used as input, but also any intermediate data view, are candidates for matching. The intermediate data view of a step is the view on the data created by the path up to and including that step. An algorithm for computing this view, given a step, has been presented in section 7.1.4. The same section includes a schema matching algorithm developed by this author, which finds the most similar candidate for matching and makes suggestions for mapping it to the new data sets. Thus the algorithm finds the best "entry point" for reusing an application model, based on syntactic and structural information. Any such mapping can be automatically created in MiningMart, but can

also be manually edited. Finding such mappings automatically is only useful when the data sets come from a similar application domain. Where only the ways of preparation are similar, the mapping has to be provided by a KDD expert.

Software product evaluation

This work has presented the first adaptable methodology for finding and evaluating objective criteria for software product evaluation (section 8.2). Previous work in this area has not used systematic ways of finding the criteria for comparison, nor metrics which are adaptable to different audiences or purposes. The methodology uses empirical "technology deltas" for finding criteria; further criteria may be found by analysis of the functional requirements of the domain. Detailed boolean features are collected into criteria, where the average number of features in a criterion determines the granularity of the evaluation. Different granularities are useful for different audiences. The *n-of-m*-metric has been introduced as an objective way of scoring, which can integrate any scheme of weighting/prioritising the criteria.

This methodology is independent of the KDD domain, but has been applied to get the first objective, in-depth comparison of KDD software packages that support data preparation. More than 50 criteria have been identified (see section 8.3). A test case that allows to evaluate these criteria quickly has been designed, see section 8.4; it corresponds to an evaluation plan, which helps to make further evaluations easier and more objective. The results of the comparison of six KDD tools are given on two different levels of granularity in section 8.6. The purpose of this evaluation was not to find a "best" tool, since the suitability of a tool depends on the purposes for which it is used; this suitability can be evaluated by applying corresponding weighting schemes to the scores in table 8.2 (page 185), putting more weight on those criteria that support the desired purpose. Instead, the evaluation has been presented as an example for the strength of the general methodology.

9.2. Future work

In this section some possibilities for extending the research presented in this thesis are discussed. Keeping in mind that successful knowledge discovery can, at present, not be fully automated, since much human intuition is needed, the goal of this thesis will remain relevant in the near future, namely to support humans during development and reuse of KDD application models. Although this thesis has achieved much progress towards this goal, some alternative approaches are possible and should be examined, and developments beyond what has been reached in this work should be pursued.

The most interesting opportunity, in the eyes of this author, is offered by integrating approaches that model an application domain with the help of rich ontology formalisms, with knowledge discovery. This idea is examined in section 9.2.2. But before that, some extensions to the MiningMart framework are discussed which assume that the present conceptual data model remains.

9.2.1. MiningMart extensions

A prominent feature in the current MiningMart framework is that it provides the two dual views on the KDD process, the web of concepts and the graph of operator applications. It is a natural idea to integrate these two views into one. The integrated view would still present a directed acyclic graph, but a bipartite one, with two different types of nodes, one for concepts and one for steps (operator applications). An edge in this graph would never connect two steps or two concepts, but only go from concepts to a step, indicating the inputs for the step, and from a step to one concept which represents the output data of the step. An immediate consequence of this requirement is that the MiningMart option of allowing operators to add only an attribute to a concept, without creating an own output concept, should be dispensed with, otherwise the effects of such an operator would be difficult to visualise. This design would make a few technical issues, like the propagation of changes to a concept to dependent concepts, easier to realise. But would it offer a clearer view of the process to the user? One the one hand, all information is available in a single view; currently MiningMart sometimes enforces inconvenient switches between the two views. On the other hand, an integrated view can quickly become graphically cluttered. But there is a remedy for this, which is to make extensive use of chunking as discussed in section 4.4. The contents of small chunks will remain clear to the user. So this integration of the two views is an interesting option for KDD tools. Since in MiningMart it would require to change many internal modules, this is left for future work.

An interesting recent development in the area of data transformations is the design of formal languages that integrate metadata and data, like SchemaSQL and FIRA, discussed in section 4.1.1. These languages natively include operators like PIVOTISATION, which may change the status of metadata to data or back. Some theoretical work in this area remains to be done; for example, the notion of transformational completeness is not yet a mature or precisely defined concept. But, taking FIRA as the more advanced example, its set of operators is small and well-defined, so it could also be used as the main component of the process model, instead of the operators suggested in this thesis. This is a promising option. There is a danger of confusing the user, however, because metadata and data are not well separated in the FIRA framework. In the framework of this thesis, metadata is a main component of the conceptual level while the data sets are located at the technical level, a separation that has been defended extensively in this work. While the smooth handling of operators like PIVOTISATION requires some additional efforts, nonetheless the separation of the two levels can be kept up almost everywhere during the development of a KDD application, as this work has shown. It remains to be examined how a similar degree of conceptual user support and reusability can be achieved using a framework like FIRA.

If FIRA implementations were widely available, they could be used at the technical level for data processing; they could realise the operators from this work without conflict.

A simpler extension at the technical level would be to allow the processing of flat file data (in tabular format), in addition to the processing inside a relational database. This would confirm the advantages of introducing a separate conceptual level. However, it seems simpler to include an operator that loads flat file data into the database, then perform the processing as before and write the results back into a flat file. In this way the virtual data representations offered by database views can be kept.

From an engineering point of view, reconsidering the way some of the declarative knowledge about operator applicability constraints is stored in M4, MiningMart's meta model, could offer some advantages. Currently the constraints that link the input and output data representations of an operator (see table 7.1 on page 162) are specifically designed for their respective purposes, which gives some of them an unintuitive meaning. One might be tempted to use a general-purpose formal language here, which would allow to formulate the constraints directly. This would remove the need for a system to interpret them (see also below, section 9.2.2). Another advantage would be that the constraints are directly readable, and unambiguous, where they are declared (assuming the reader is familiar with the language used). A disadvantage is that more complex constraints could introduce errors, simply by misdesign or by complex interactions with other operators (since the output of one operator is the input of another). The current constraints ensure at least that those who use them to specify an operator do not introduce inadequate side effects.

Regarding the case base, some interesting approaches can be realised as soon as more models of successful KDD applications have been collected in it. Experience has shown that research can benefit greatly from publicly available collections of algorithms, or benchmark data sets, or similar infrastructures. Besides offering an open modelling standard for KDD, a richer case base can be examined for frequently occurring subproblems, and can be used for collaborative work and for education purposes. It will be interesting to see results of applying the frequent subgraph discovery algorithm that has been proposed in this thesis. One might also be able to develop larger blueprints, for specialised application domains like telecommunications or banking, than are given by the current templates, based on collected experiences from such a domain.

The explicitly modelled conceptual level also allows to explore the options of distributed computing for KDD, or grid-based data processing. This would require more complex solutions at the technical level, but it should be possible to use the conceptual level without any changes. Distributed computing requires to model an application independent of where and when it is executed, exactly what this work enables for KDD. Current research efforts in this area (see section 6.1.2) should thus be able to benefit from the conceptual analyses contributed by this thesis.

9.2.2. Using ontologies in the knowledge discovery process

Concerning a conceptual model of the data and the data schemas to be used in a KDD application, this work has proposed to use a clear and not too sophisticated conceptual data model, the ER model from section 3.2.2. It only models metadata, allowing a rather strict separation from the actual data. This strict separation is violated by only a few operators which transform data to metadata or the other way round. In this work this separation has been defended extensively, in order to ease reusability, which is a prerequisite for the case-based approach described in chapter 6. The ER-based meta model has enabled a clear and legible view on the complex graph of data set representations created in a typical KDD application.

A price for using this rather understandable model is that formal reasoning based on it had to be defined and implemented separately. This reasoning concerns the "signature" of output concepts (their attributes and conceptual data types), in order to get valid

operator chains, and characteristics of the data these concepts represent.

One promising direction for future research is to use description logics (Baader et al., 2003) for conceptual data modelling, because this formalism allows reasoning directly, so that existing implementations of reasoners could be employed. Descriptions logics are a family of modern, powerful, logic-based knowledge representation formalisms (ontology formalisms) which allow reasoning. A description logic language corresponds to some fragment of first-order predicate logic, but uses a more concise syntax. Description logics have already been used for conceptual data modelling, including the abstract modelling of relational databases, so that one can build on existing research, see (Borgida et al., 2003).

Using description logics can allow additional reasoning beyond the tasks mentioned above. For example, inconsistencies in a data model that lead to a concept whose extension must always be empty can be recognised automatically. Such a case could be introduced, in a data preparation context, by a join over key attributes known to be disjunct, for example. In general, however, to support such reasoning, the data must be modelled to more detail, yielding more complex conceptual views. Suitable graphical representations would have to be developed.

A good example for this, and in general for the opportunities that description logics may offer for data preparation, is the work by Franconi and Ng (2000). These authors present a tool that supports the integration of a number of information systems, using description logics-based conceptual models of their data schemas. The relationship between data integration and data preparation has been discussed in section 4.1.1. The tool can express connections between different schemas with inclusion dependencies, which are native elements of the employed description logic language, and whose semantics are similar to those of the separation links used in the present work. Thus the tool can be used to create a global, integrated data schema and show its dependencies on the source schemas.

But for this section another aspect of the tool is more interesting. It includes an extended data model, described in (Franconi & Sattler, 1999), that can be used to model dimensions of aggregation functions. For example, to compute the average length of phone calls for different types of calls (e.g. calls to mobiles, internet providers, free call numbers etc.), the dimension *type of call* is explicitly modelled by including the different types as elements into the conceptual data model. This allows to explicitly represent an aggregated view in terms of what it aggregates (linking the element that represents the aggregation with the elements that represent the types of calls, for example). Interestingly, the authors have first defined the conceptual model as an extension of the ER model, adding elements representing dimensions to those representing entity types and relationships, and have then defined a translation into a description logic language. The ER model serves the graphical representation while the logic is used for inferences in the background. Franconi and Ng (2000) describe an example for reasoning, in which a certain aggregation is concluded to be necessarily empty because it involves aggregation over non-occurring value combinations. Translated to data preparation, this means that an interesting property of the output of a preparation operator could be inferred without executing the operator. The same inference would be possible in the framework of the present work, under certain circumstances, based on the data characteristics, but this

particular check for emptiness of the output has not been examined in this work while it comes for free with description logic reasoners. Emptiness of output is also an issue for the JOIN and ROW SELECTION operators. The price is that the ER model, which provides the user interface, is more complicated, and seems to lead to very complex graphs if extended to a complete KDD process. Allowing other types of inference for other preparation operators requires even more explicitly modelled aspects of the data. Nevertheless, this is an interesting direction for future research if the clarity of the visualisation can be kept.

Using ontological formalisms in KDD might be even more worthwhile if more data mining algorithms were able to directly exploit structures in their input data. However, currently almost all algorithms are applied to "flat", tabular inputs. For example, generalised association rule mining is used for finding sets of items that are frequent in a given database, when the items are ordered by a taxonomy; nevertheless the algorithm is applied to input in which the taxonomy structure is flattened, by simply adding all parent items to each item set in the database (Srikant & Agrawal, 1995). Even the recent approaches for learning in structured output spaces (Tsochantaridis et al., 2005) employ a flattened, vector-based "joint feature representation". Thus, exploiting ontological structures is currently more an issue for data preparation than for mining, and has therefore been discussed above. Future work on mining algorithms might bring up ideas to incorporate taxonomies etc. directly into the algorithm, which could stimulate more research on using ontologies in all phases of the KDD process.

Appendix A: Preparation operators

This appendix lists all data preparation operators. Their choice and the schema of description is discussed in section 4.2. They are organised into sections (groups) according to the high-level preparation tasks identified in section 2.1.3.

A.1. Data reduction operators

A.1.1. Attribute selection

Description This operator creates an output concept which is a copy of the input concept, but has some attributes removed. Two versions of this operator are considered, depending on how the selection of attributes to be removed is done. In the first version the user simply specifies a list of attributes to be removed (or to be retained). In this version the shape of the output concept does not depend on the input data. However, for advanced applications, *automatic* attribute selection is needed, using redundancy criteria with respect to the input data, or the performance of a mining algorithm on different attribute sets. So in this version the selection of attributes to be removed may depend on the data. No restrictions on the algorithms for automatic attribute selection are imposed.

Relevance to mining Manual selection of attributes can remove information that is obviously useless for finding patterns in the data, such as telephone numbers. Automatic selection can be used for the same purpose when the usefulness of attributes is difficult to judge for humans (Liu & Motoda, 1998). Fewer attributes for learning enable the learning algorithm to find the relevant patterns faster.

Input and output The input is any concept C with at least two attributes, $|attr(C)| \geqslant 2$. The output is a concept C' of which the input concept is a specialisation: $attr(C') \subset attr(C)$ so that $C <_{sp} C'$.

Parameters The input concept, and the list of attributes to be removed, or the method how to select such attributes automatically (see below). Another variant of this operator receives the list of attributes to be retained, rather than removed.

Constraints The input concept must have at least two attributes.

Conditions None.

Assertions The data types and roles of the selected attributes are copied from the corresponding input attributes.

Estimates The characteristics of the selected attributes are unchanged.

Special options

- Removal of attributes according to criteria which are computable from the attribute's values, such as ratio of missing values.
- Automatic attribute selection according to criteria such as correlation of attributes, or information gain with respect to a given class attribute.
- Automatic attribute selection by training and evaluating a mining algorithm on different attribute sets; various search methods among the attribute sets (Liu & Motoda, 1998).

Application example Removal of the birthday attribute after a derived age attribute is computed.

A.1.2. Row selection

Description This operator creates an output concept which is a copy of the input concept, but has certain entities removed from its instance. It is sufficient if the operator can select entities according to the values of a binary attribute in the input concept; then arbitrary selections are possible by deriving this binary attribute first, using the operator ATTRIBUTE DERIVATION (A.5.4). However it may be more convenient to allow arbitrary selection formulas for this operator directly.

Relevance to mining The operator can be used to select subgroups of the data for particular analysis or preparation.

Input and output The input is any concept C. The output is a concept C' that is a separation of the input concept: $attr(C) = attr(C')$ and $C' \leqslant_{sep} C$.

Parameters The input concept and a selection criterion.

Constraints None.

Conditions None.

Assertions The data types and roles of the output attributes are copied from the input attributes.

Estimates Refer to section 3.3.3 for a general discussion of how histograms (the value distribution statistics) can be used to estimate the output size of the selection operator. Many simple entity selection operations are based on value selections for one attribute, like selecting all entities where the attribute `colour` takes the value `green`, or similar. Here the value list and distribution of the corresponding attribute in the output concept can easily be adjusted. When the condition for selection is composed by simple conditions on several single attributes using the logical AND-operator, the value distributions can similarly be computed. When OR is the logical operator, this is not possible anymore; applying optimistic estimation, the list of values in the output does not change (though estimating their distribution would be too optimistic).

Application example Removal of entities whose value of a certain attribute is missing.

A.1.3. Sampling

Description This operator is a specialisation of ROW SELECTION that chooses the output entities according to some random function.

Relevance to mining The main purpose of sampling is data reduction, but changing the distribution of the data can also be useful for mining (see the special options below). More advanced sampling approaches are described by Scholz (2007), for example; such approaches integrate sampling with mining, and would require separate operators.

Input and output The input is any concept C. The output is a concept C' that is a separation of the input concept: $attr(C) = attr(C')$ and $C' \leqslant_{sep} C$.

Parameters The input concept, and a sampling rate or a target sample size.

Constraints None.

Conditions None.

Assertions The data types and roles of the output attributes are copied from the input attributes.

Estimates The value lists of the input attributes can optimistically be assumed to be unchanged. The output size can be estimated rather accurately from the sampling rate and the input size, or from the target sample size. From the output size and input size, the sample rate can be estimated if only the target sample size is given as a paramter; then the value distribution for the output attributes can be estimated by multiplying the input frequencies of each value with the sample rate.

Special options

- Uniform sampling: each entity from the input has the same probability of being selected.
- Stratified sampling: uniform sampling is done separately from a number of mutually exclusive subgroups in the data, in order to keep the distribution among the subgroups. An additional parameter must identify the subgroups (for example by the distinct values of a discrete attribute).
- Label-based under-sampling: entities identified by a certain value of an attribute with the *label* role (see section 3.3.2 for attribute roles) have a lower probability to be selected than others. Additional parameters must specify this lower probability and the label value. See (Chawla et al., 2002) for reasons why this is useful in data mining.

Application example Sampling a training set from a set of labelled data.

A.1.4. Aggregation

Description This operator aggregates values of the input concept according to the values of given *Group By*-attributes. Aggregation attributes are chosen in the input concept; in the output concept, values that are aggregated over an aggregation attribute appear for each combination of values of the Group By-attributes.

Relevance to mining Besides data reduction, aggregation can also be used to represent data at different levels of granularity. The most suitable level of granularity depends on the application domain and the capabilities of the mining algorithm.

Input and output The input is any concept C with at least two attributes ($|attr(C)| \geqslant 2$). The output is a new concept C' that is linked to C by a relationship type $R = (C, C', oneOrMore, one)$. The keys of the technical realisation of the relationship are given by the Group By-attributes.

Parameters The input concept, the Group By-attributes, the aggregation attributes and the aggregation operator for each aggregation attribute.

Constraints The Group By-attributes must be discrete. The aggregation attributes must be numeric, except if the aggregation operator is *count* or *countdistinct*.

Conditions The Group By-attributes must not have only missing values.

Assertions The data types and roles of the Group By-attributes in the output are copied from the corresponding input attributes. The data type of the aggregation attributes in the output is *continuous*. Only the Group By- and aggregation attributes are available in the output.

Estimates The value lists of the Group By-attributes remain unchanged. The value frequencies of the Group By-attributes can be determined (for example, if there is only one Group By-attribute, all its values will occur exactly once). Similarly, the size of the output can be computed. The value lists of the other output attributes are unknown. There may be missing values in the output.

Special options Aggregation functions include *minimum*, *maximum*, *average*, *median*, *sum*, *count*, and *countdistinct*.

Application example Given a concept containing employee information, including the department where the employee works, compute the number of employees for each department.

A.2. Propositionalisation operators

These operators exploit the presence of relationship types between concepts to safely integrate the concepts. "Safely" means here that the relationships signify the semantic compatibility of the concepts to be joined, so that two concepts whose entities denote incompatible things cannot be joined because no relationship would exist between them. Of course, users could set up such semantically flawed relationships, but the probability that they do so erroneously is certainly lower than that of erroneously joining incompatible concepts. In order to be able to join concepts wherever needed, a system that provides these operators must allow to create relationships between concepts at any time.

The operator UNION in this section is an exception, as it does not require a relationship between its input concepts, but since it is only applicable on concepts with equal signature (sets of attributes), the chances of applying it erroneously are also low.

A.2.1. Join by relationship

Description This operator joins two concepts that are linked by a relationship type. All attributes from the input concepts occur in the output concept, except that the join attributes are not duplicated in the output but occur only once. The join attributes are specified by the relationship type. The operator realises the well-known natural (equi-)join from the relational algebra.

Relevance to mining Propositionalisation of data is needed for most mining algorithms, as they expect a single data table as input.

Input and output The input are 2 concepts C_1, C_2 which are linked by a relationship types.

The output is a concept C' for which the following holds: $attr(C_1) \subseteq attr(C'), attr(C_2) \subseteq attr(C')$. Exactly one representative of each join attribute occurs in the output concept.

The operator produces specialisation links from the output to each input concept: $C' <_{sp} C_1, C' <_{sp} C_2$.

Parameters The relationship type by which to join the two concepts that it links.

Constraints The two concepts that are linked by the relationship type must not contain like-named attributes, unless they are the keys used in the relationship.

Conditions The two concepts and the relationship type must have the same number of instances.

Assertions The data types and roles of the output attributes are copied from the input attributes.

Estimates The lists of the values of the output attributes, as well as minimum and maximum bounds, can be optimistically estimated, i.e. left unchanged. Their value distributions cannot be inferred nor estimated. The number of entities in the output concept can be inferred from the details of the relationship (see section 3.3.3).

Application example Joining customer contract data with data about what products the customers ordered.

A.2.2. Aggregate by relationship

Description This operator extends its input concept by an attribute that contains aggregated values computed from a concept linked to the input concept by a relationship. The particular version of this aggregation operator was introduced by Perlich and Provost (2003), whose work is discussed in section 4.1.2. They discuss a few other, similar operators, for which this one is exemplarily included in this chapter. The computation of aggregated values is done only for those entities of the input concept for which related entities are available in the linked concept (the latter are then aggregated). What is more, the aggregation is specified to range only over particular entities (of the linked concept), namely those whose value of a given target attribute matches the value of that target attribute that is most frequent in the relationship. See the application example below. So this operator relies on the information given in a relationship between the input concepts.

Relevance to mining Propositionalisation of data is needed for most mining algorithms, as they expect a single data table as input. This operator can add information from a different concept to the concept whose instance holds the examples for learning, extending the representation of the data that is used as input for mining.

Input and output The output is a concept that is a specialisation of the concept to which the aggregated value is added.

Parameters The relationship, the aggregation operator, and the target attribute of the second concept (whose values are going to be aggregated).

Constraints The attribute to be aggregated must be *continuous* unless the aggregation operator is *count*.

Conditions The two concepts and the relationship type must have the same number of instances.

Assertions The data type of the newly created attribute is *continuous*. The data types and roles of the other output attributes are copied from the input attributes.

Estimates The size of the output is equal to that of the input.

Special option If the input concept contains a discrete attribute whose role is *label*, the aggregation can be done with respect to the classes given in the label attribute. See the application example.

Application example Two concepts with data about customers and products of a company might be linked by a relationship that indicates which product has been bought by which customer. Taking the customers concept as the input concept and the products as linked concept, this operator can compute the number of times a customer has bought the product that has been bought most often by any customer. Thus the operator computes a single new aggregated value for each entity in the customer concept (the value may be empty if the customer has not bought the frequent product). If the special option above is realised, the operator would compute the difference between the number of times a customer from a particular class has bought the most frequent product (on average) and the number of times other customers have bought this product.

A.2.3. Union

Description This operator unifies two or more concepts that have the same attributes. The instance of the output concept contains all entities of all instances of the input concepts. If entities occur multiple times, they do so in the output, too. If an entity occurs in more than one input concept, its numbers of occurrences in the input concepts are added to get the number of occurrences in the output.

Relevance to mining This operator is mainly useful for unifying two or more subsets of some data that have been prepared in different ways.

Input and output Every input concept $C_1, \ldots, C_n$ is a separation of the output concept C': $C_1 \leqslant_{sep} C', \ldots, C_n \leqslant_{sep} C'$.

Parameters The input concepts (at least two).

Constraints All input concepts must have the same signature (the same attributes).

Conditions All input concepts must have the same number of instances.

Assertions The data types and roles of the output attributes are copied from the input attributes.

Estimates The lists of values can be unified for matching attributes. Optimistic estimates for the value distributions are gained by adding the number of occurrences of each value or interval, and the number of entities in the output is the sum of the number of entities in the inputs. Minimum and maximum bounds, and the number of missing values, can also be gained from combining the corresponding input characteristics.

Special options Allows to include or exclude duplicate entities in the output (bag or set semantics).

Application example Unify data sets with different target labels (for example the positive and negative examples), in a classification task, after they have been prepared differently.

A.3. Operators changing the data organisation

A.3.1. Dichotomisation

Description This operator takes a discrete attribute and produces one new attribute for each of its values. Each new attribute indicates the presence or absence of the value associated with it by a binary flag.

Relevance to mining This operator can be used to create continuous attributes from discrete ones, by using the numbers 0 and 1 for the binary flag. This is useful for mining algorithms that only handle continuous input. The operator is also useful for association rule discovery algorithms that expect a boolean matrix for representing transactions. Compare the template "PrepareAssociationRulesDiscovery" in section 6.5.3.

Input and output The output concept C' is a specialisation of the input concept C: $C' <_{sp} C$.

Parameters The input concept and a discrete attribute in it.

Constraints The target attribute must be *discrete*. (It can be *binary*, too, but then this operator only copies the attribute.)

Conditions None.

Assertions The data type of the new attributes is *binary*. The number of newly created attributes is known if the value list of the attribute to be dichotomised is known. The data types and roles of the other output attributes are copied from the input attributes.

Salesperson	Week	Sales
Smith	1	3
Smith	2	4
Marks	1	7
Marks	2	6
...	...	...

Salesperson	SalesWeek1	SalesWeek2
Smith	3	4
Marks	7	6
...	...	...

Figure A.1.: Example input (left) and output concept, with instances, of a PIVOTISATION application, explained in the text.

Estimates The value list of each new attribute is clear from the symbols that are used for the binary flag. The value distribution can be inferred if (and only if) it is known for the input (for example, the number of occurrences of 1s for a new attribute corresponds to the number of occurrences of the value it represents in the input attribute). If numeric symbols (like 0 and 1) are used for the binary flags, they also specify the minimum and maximum values of the output. The number of missing values of each new attribute can be optimistically taken from the dichotomised input attribute, divided by the number of values in that attribute.

Application example Change of representation of discrete attributes to technically numeric attributes if 0 and 1 are used for the flag values. This is useful for some mining algorithms that cannot handle discrete attributes.

A.3.2. Pivotisation

Description Pivotisation means to take the values that occur in an *index attribute* (of discrete conceptual data type) and to create a new attribute for each of these values (Cunningham et al., 2004). Each new attribute contains the (aggregated) values of a *pivot attribute* for those entities (or aggregated over those entities) that contain the index value associated with the new attribute. Thus the pivot values are distributed over the new attributes which correspond to the index value (compare the application example). Aggregation is optional; it is done by the values of *Group By*-attributes.

Relevance to mining This operator is useful for re-representing some information that is stored in *values* of a single attribute, as *attributes* for learning. The operator thus also supports propositionalisation, as it allows to represent the information as attributes of single examples for learning, rather than having several entities with the different values. Compare the application example.

Input and output The input is any concept C with the required attributes. The output is a new concept C' that is linked to C by a relationship type $R = (C, C', oneOrMore, one)$. The keys of the technical realisation of the relationship are given by the Group By-attributes.

Parameters Input concept, index attributes, Group By-attributes (optional), pivot attribute, and an aggregation operator (none if no Group By-attributes are given).

Constraints The index attribute must be *discrete*. The Group By-attributes, if there are any, must also be *discrete*.

Conditions Neither the Group By-attributes nor the index attribute must contain only missing values.

Assertions The number of newly created attributes is known if the value list of the index attribute is known. The conceptual data type of the new attributes is given by that of the pivot attribute. The type of the Group By-attributes is *discrete* in the output, too.

Estimates When aggregation is used, the estimates for the Group By-attributes and the output size are the same as for AGGREGATION. The value list and value distribution of the new attributes are unknown then. When no aggregation is used, the value lists of the new attributes can optimistically be copied from the pivot attribute.

Special option Generalisation to n-fold pivotisation: there are n index attributes ($n > 1$), and one pivot attribute. All *combinations* of values of the index attributes lead to a new attribute in the output.

Application example Figure A.1 shows input and output concept, with extensions, of an example application of this operator. The input concept contains weekly sales performed by some salespersons of a company. The output lists the sales for each salesperson in new attributes. Here, the index attribute is `Week` and the pivot attribute is `Sales`. In the example, no aggregation is necessary, but `Salesperson` is used as a Group By-attribute; if more than one `Sales` entry was available per `Week`, they could be aggregated using summation, for instance.

A.3.3. Reverse pivotisation

Description This operator is the reverse operator to pivotisation without aggregation. Certain attributes of compatible technical data type are folded into one attribute, such that the output contains more records than the input; in the remaining attributes, the values are filled up. See the application example of PIVOTISATION (figure A.1), but exchange input and output.

Relevance to mining This operator allows to re-represent information by creating *values* from different attributes. Thus it creates a set of examples (entities) from one example (entity). This can be used to create more examples for learning which are differentiated by the values of a single attribute, rather than by several attributes.

Input and output The input is any concept C with at least two attributes of the same conceptual data type. The output is a new concept C'. It is linked to C by a relationship type $R = (C', C, oneOrMore, one)$ if C has additional attributes not involved in the reverse pivotisation. The keys of the technical realisation of the relationship are given by these additional attributes.

Parameters Input concept, two or more pivot attributes of the same type, and the index values these pivot attributes represent.

Constraints The pivot attributes must have the same conceptual data type.

Conditions The technical realisations (e.g. database columns) of the pivot attributes must have the same technical data type.

Assertions The newly created attribute with the index values is *discrete*. The newly created single attribute with the pivot values is of the same type as the input pivot attributes.

Estimates The output size is the input size times the number of index values. The value list of the index attribute is given by the parameter with the index values. The index attribute does not have missing values. The value list of the pivot attribute in the output is the union of the value lists of the pivot attributes in the input.

A.3.4. Windowing

Description This operator is useful for value series data. It changes the representation of a value series to a representation based on sliding a window of fixed width over the series. The input concept must contain an *index attribute* and a *value attribute*. The output concept will contain one entity for each window. It includes two attributes indicating the start and end index for each window, and as many further attributes as given by the window width; these contain the values of the value attribute for each window, and are therefore called *window attributes*. See the example in figure A.2.

Relevance to mining This operator is paramount for handling time-stamped data. It makes a time or value series accessible for a mining algorithm by representing it as a set of examples of the same kind.

Input and output The input is any concept C with the required attributes. The output is a new concept C' that is linked to C by a relationship type $R = (C, C', one, zeroOrOne)$. The keys of the technical realisation of the relationship are given by the index attribute for C and the start or end index attribute for C'.

Parameters The input concept, an attribute of type *Time* for the index, the window width, and an attribute for the value series.

Time	Pressure
1	95
2	97
3	96
4	96
5	97
6	95

Start	End	Pressure1	Pressure2	Pressure3
1	3	95	97	96
4	6	96	97	95

Figure A.2.: Example input (left) and output concept, with instances, of a WINDOWING application. The window width is 3. `Time` is the index attribute and `Pressure` the value attribute in the input. `Start` and `End` are the start and end index attributes, while `Pressure1`, `Pressure2` and `Pressure3` are the window attributes.

Constraints The index attribute must be of type *Time*. The window width must be positive.

Conditions None.

Assertions The conceptual data type of the start and end index attributes in the output is *Time*. The conceptual data type of the window attributes in the output is given by that of the value attribute. The number of window attributes is given by the window width.

Estimates The number of entities in the output is given by that of the input divided by the window width. The value list of the start and end index attribute can be optimistically estimated to be the same as the value list of the index attribute in the input. Similarly, the value lists of the window attributes can be optimistically estimated to be equal to the value list of the value attribute, unless aggregation is used. Finally, the value frequencies, and the number of missing values, of the window attributes can also be copied optimistically, but divided by the window width.

Special options Another version of this operator computes an aggregated value for each window, so that only one window attribute is created.

Application example This operator might be used to compute the moving average of a time series, for example a series of blood pressure measurements of a single patient at an intensive care unit, resulting in average blood pressure values per time unit, where the time unit corresponds to the window width.

A.4. Data cleaning operators

A.4.1. Missing value replacement

Description This operator fills missing or empty values (see section 2.1.3) in a specified input attribute.

Relevance to mining Most mining algorithms cannot handle missing values. Instead of deleting entities with missing values, which can also be a useful strategy, this operator attempts to fill the gaps. The operator must be used with care so that the representativeness of the data is not impaired. For more information, see (Pyle, 1999).

Input and output See ATTRIBUTE DERIVATION (A.5.4).

Parameters Input concept and an attribute in it (the target attribute for replacement).

Constraints If replacement is done by an average value, the attribute whose values are replaced must be *continuous*.

Conditions None (if there are no missing values in the input, the operator does not change this).

Assertions The data types and roles of the output attributes are copied from the input attributes. The new attribute has the same data type as the one whose values are replaced, and it does not have missing values.

Estimates The number of missing values can be set to zero for the output attribute. The list of values for this attribute can be updated to exclude the special value that represents a missing entry (if the target attribute is discrete). If a default value is used for replacement it can be included in the value list. For continuous target attributes, the minimum and maximum values are not changed (unless the default value is the new minimum or maximum). The value distribution of the output attribute can be updated if a default, median or average value is used for replacement (for example, the number of occurrences of the default value in the output can be increased by the number of missing values in the input). In the other cases, the value frequencies of the output attribute after replacement can be optimistically assumed to be uniformly increased (by the number of missing values in the input, divided by the number of occurring other values).

Special options The value for replacement can be determined by using

- one default value; or
- the median or average of existing values; or
- values selected randomly with a bias that does not change the statistical distribution of the values of the attribute; or

- a predictive model trained on the remaining attributes. This option should be integrated into this operator, because otherwise a non-trivial set of operators for selecting entities with and without missing values, training a model, applying it, and combining the predicted values with the non-missing values into a single attribute would be necessary to realise this option.

A.4.2. Filtering outliers

Description This operator offers various statistical measures that indicate "outliers", i.e. entities with extreme values that are expected to disturb the mining results more than making them generalisable. Such outliers are not copied to the output.

Relevance to mining Outliers can deterioate the mining result of distance-based algorithms due to their extreme values. In most cases, outliers are simply input errors of the data collecting process, and thus should be removed.

Input and output The input is any concept C, the output is a new concept C' that is a separation of the input: $C' \leqslant_{sep} C$.

Parameters Input concept and an attribute in it (the target attribute in which outliers are searched).

Constraints None.

Conditions None.

Assertions The data types and roles of the output attributes are copied from the input attributes.

Estimates Optimistic estimation leaves the value list or the value distribution of the output attribute unchanged, in the hope that there are no or only a few outliers.

A.5. Feature construction operators

All operators in this section have one additional parameter in common which specifies the name of the newly constructed attribute/feature.

A.5.1. Discretisation

Description This operator discretises a continuous attribute. That is, the range of values of the continuous attribute is divided into intervals, and a discrete value is given to every entity according to the interval into which the continuous value falls.

Relevance to mining Some mining algorithms only handle discrete input. Others discretise continuous input internally, in which case the KDD expert may want to keep control by doing it explicitly beforehand. Like aggregation, discretisation also changes the level of granularity of information.

Input and output The input is any concept C. The output is a concept C' that is a specialisation of the input concept: $attr(C') = attr(C) \cup a'$ with $a' \in A$ but $a' \notin attr(C)$. Thus $C' <_{sp} C$. The instance i' of C' contains exactly the entities of the instance i of C, extended by the value for the new attribute a'.

Parameters Input concept and a continuous attribute in it.

Constraints The attribute to be discretised must be *continuous*.

Conditions None.

Assertions The data type of the additional attribute is *discrete*. If only two discretisation intervals are chosen, it is *binary*. The data types and roles of the other output attributes are copied from the input attributes.

Estimates The number of constructed intervals is known (for most of the discretisation methods), as well as the symbols to be used for each interval in the output; this determines the list of values in the newly created attribute. If the option below to specify interval construction by the number of entities to fall into each interval is used, even the value distribution of the output attribute is known. The number of missing values in the new attribute equals that of the undiscretised input attribute.

Special options Interval construction can be determined by specifying

- the interval bounds; or
- the number of intervals; or
- the width of the intervals; or
- the number of entities to fall into each interval; and
- whether the so constructed intervals should be of equal width or equal cardinality.

Application example Forming age groups (like child, young adult, adult, pensioner) from an age attribute.

A.5.2. Scaling

Description This operator rescales a continuous attribute to a new given range. Different ways of scaling, like linear or logarithmic scaling, are offered. Scaling values of different attributes to a common range is sometimes also called normalisation.

Relevance to mining The scale of continuous attributes can be important for distance-based mining algorithms, like clustering or the support vector machine (SVM): attributes with larger values can have more influence on the result than those with a smaller range. Scaling can be used to normalise all attributes to the same value range.

Input and output The input is any concept C. The output is a concept C' that is a specialisation of the input concept: $attr(C') = attr(C) \cup a'$ with $a' \in A$ but $a' \notin attr(C)$. Thus $C' <_{sp} C$. The instance i' of C' contains exactly the entities of the instance i of C, extended by the value for the new attribute a'.

Parameters Input concept, a continuous attribute in it, and the minimum and maximum value of the new range for the values of that attribute.

Constraints The attribute to be scaled must be *continuous*. The minimum value of the new range must be lower than the maximum.

Conditions For logarithmic scaling, all values in the attribute to be scaled must be positive.

Assertions The data type of the additional attribute is *continuous*. The data types and roles of the other output attributes are copied from the input attributes.

Estimates The minimum and maximum of the values in the newly created output attribute are given by the corresponding parameters of this operator.

Application example Scaling the income of customers to the normal range [0..1].

A.5.3. Value mapping

Description This operator maps values of a discrete attribute to new values. In this way, different values can be mapped to a single value, thus be grouped together, if they should not be distinguished later in the process.

Relevance to mining This operator can be used for different purposes. A typical application is to correct wrong input, such as misspellings. But it may also be used to change the level of granularity of information, like DISCRETISATION does for continuous attributes. For example, the operator can introduce a category for single items, like a product group for single products.

Input and output The input is any concept C. The output is a concept C' that is a specialisation of the input concept: $attr(C') = attr(C) \cup a'$ with $a' \in A$ but $a' \notin attr(C)$. Thus $C' <_{sp} C$. The instance i' of C' contains exactly the entities of the instance i of C, extended by the value for the new attribute a'.

Parameters Input concept and a discrete attribute in it.

Constraints The attribute whose values are to be mapped must be *discrete* or *binary*.

Conditions None.

Assertions The data type and role of the additional attribute are the data type and role of the input attribute whose values are mapped. The data types and roles of the other output attributes are copied from the input attributes.

Estimates From the specification of the operator, the list of values in the newly created output attribute is known directly if the input value list is available (if not, there might be input values that are not mapped, so they would appear in the output but are unknown). The value frequencies can also be computed: for example, if two different input values are mapped to the same output value, the output value's frequency equals the sum of the frequencies of the input values.

Application examples

- Assignment of meaningful names to discretised intervals (like age group names).
- Correction of misspellings or outliers in the input.

A.5.4. Attribute derivation

Description This is a very general operator to create a new attribute, and values of this attribute for each entity in the input concept's instance. The new values must be computable based on values of existing attributes (though these values can of course be ignored, for example to create random values for the new attribute). To allow this, extensive date, string and numeric arithmetics must be offered by this operator. In fact, a computationally complete formalism such as a programming language is needed. Note that this operator, as the only one in this work, requires the user to access the technical description level. Only the syntactic signature of this operator is fixed at the conceptual level (it adds an attribute to its input concept). This operator can be used as a fallback option for unusual preparation tasks, by the flexible computation of attributes whose values are derived from the given data. Such flexible computations are indispensable for supporting advanced preparation ideas by experienced users. In the data mining literature, this is called *feature construction* (Liu & Motoda, 1998). There are automatic approaches to feature construction, but it is also an important tool for manual preparation. See also section 4.3.

Below under "Special options", some suggestions for frequently needed functions for attribute derivations are listed. They could be offered as special operators rather than as options of this elementary operator. However, the combination of these options is often useful, and is simpler if they are available in one operator.

The name of the new attribute can either be specified as a parameter, or it can be computed from some values in the instance of the input concept, or it can be computed from the name of the input concept or from the names of its attributes. This may be necessary to enable the change of status from data to metadata, compare section 4.1.1.

Further, the values of the new attribute may depend on the names of the input concept or its attributes, to change metadata to data.

Relevance to mining This operator gives KDD experts the flexibility to realise new ideas of representing and computing additional information. The operator can be used as a fallback option for situations in which the other operators that compute new attributes do not suffice. In particular, it can be used to combine values from different attributes.

Input and output The input is any concept C. The output is a concept C' that is a specialisation of the input concept: $attr(C') = attr(C) \cup a'$ with $a' \in A$ but $a' \notin attr(C)$. Thus $C' <_{sp} C$. The instance i' of C' contains exactly the entities of the instance i of C, extended by the value for the new attribute a'.

Parameters The input concept, the name of the new attribute (optional), and a formula for its derivation.

Constraints None.

Conditions None.

Assertions The data types and roles of the output attributes are copied from the input attributes, except for the newly created attribute.

Estimates The technical data type of the new attribute depends on, and is deducible from, the operations that create the attribute. The conceptual data type can be guessed from it.

In general, it is obviously impossible to predict the value distribution of the derived attribute from the function used for the derivation, without computing the function on all values. However, for optimistic metadata administration, partial information such as the list of values without their distribution can also be helpful. For the special case of constant functions, or logical functions returning one of a number of constants, the possible values of the result are known. When random values from a given interval are created, their minimum and maximum values are known beforehand; they may also be known in other cases. The new attribute can be optimistically expected not to have any missing values.

The number of entities in the output is equal to that in the input.

Special options

- Numeric arithmetics: basic mathematical operators, trigonometric functions, maths library (absolute value, logarithms, exponentiations, roots, minimum/maximum/mean/median etc.)
- Date/time arithmetics: extraction of year, month, day, weekday, hour, minute, second from dates and times; addition and subtraction of dates and times; availability of system time

- String processing options: Substring extraction, concatenation function, case conversion etc.
- Logical operations: and, or, not, if then else
- Bitwise operations: shift, and, or, not etc.
- Comparisons: equal to, less than etc. for each conceptual data type
- Type conversions of technical data types: string to number etc.
- Handling of missing/empty fields (e.g. `Null` values)
- Generation of values (for example, running integers, or random values from a given set)
- Computation of principal components (each component resulting in one derived attribute)

Application examples

- Computation of the age of a person given the birthday and the current system time
- Creation of a primary key for the input concept
- Creation of a binary indicator for the presence or absence of a certain value in a certain attribute
- Renaming an attribute by creating a copy of it with a new name

A.6. Operators for pseudo-parallel processing

The two operators in this section help to process several tables with the same schema behind a single conceptual representation. This was motivated in section 1.1.1. The first operator splits a table into several parts, all represented by the single output concept; the second operator unifies several tables that are attached to a single concept, so that the output concept represents the union.

A.6.1. Segmentation

Description This operator segments the instance of the input concept into a number of instances of the output concept, whose attributes are either the same as in the input concept, or lack exactly one of the input attributes. The instances of the output concept (the segments) are disjoint. Three methods of segmentation are distinguished: (i) the values of a particular, discrete attribute of the input concept determine the segments (each value corresponds to one segment); (ii) a fixed number of segments is created by randomly assigning input entities to the segments; (iii) a fixed number of segments is created by clustering the input segments according to some similarity measure, so that each cluster corresponds to one segment. For the first method, the number of output

instances depends on the input instance. When this method is used, the output concept does not have the attribute by whose value the input instance is segmented. For the last two methods, the number of output instances is known by an input parameter, and the output concept has the same attributes as the input.

Relevance to mining This operator allows to split a data set into several parts that can be processed alike. Thus this operator is one tool by which several identical processes can be executed using one solution model.

Input and output The input is any concept C. The output is a concept C' that is a separation of the input concept: $C' \leqslant_{sep} C$.

Parameters The input concept, a method of segmentation, and for the last two of the segmentation methods: the number of output instances (segments).

Constraints For segmentation by the values of a particular attribute, this attribute must be *discrete* or *binary*.

Conditions None. One may want to exclude the possibility of missing values in the attribute by which the segments are found.

Assertions The types and roles of the output attributes are copied from the input.

Estimates Estimates are given as if for one instance. If the value distribution of the segmentation attribute is given (referring to the first segmentation option above), the number of entities in each segment is known. The other value lists are optimistically estimated to remain the same, while the value frequencies may be estimated by dividing them through the number of segments. When random segmentation is used, the number of output instances (segments) is given as a parameter, and the number of entities of a segment can be approximated from the known bias of random selection (usually a uniform distribution will be used, meaning that roughly the same number of entities is assigned to each segment). The situation is similar, based on optimistic assumptions, when automatic clustering is used.

Application example Producing a random split of a concept into training and test set.

A.6.2. Unsegmentation

Description This operator reverses SEGMENTATION. Its input concept may represent several data tables with the same schema. Its output concept will be attached to the data table that contains the union of the input tables. If the segmentation had been done by a segmentation attribute, this attribute is no longer present in the data; its name and its values can be given by parameters. Its values can also be found by implementational tricks if the operator SEGMENTATION attaches the values to the instances, for example (a solution used in MiningMart).

Relevance to mining Re-unifying separately processed data may be useful when only the preparation, or only a part of the preparation process, but not the mining phase, requires processing several identical data sets in the same way.

Input and output The input concept is a separation of the output concept. If the output concept has the additional, reconstructed segmentation attribute that is missing in the input, the output is a specialisation of the input.

Parameters The input concept, and the name and type of the segmentation attribute if there was any.

Constraints None.

Conditions None.

Assertions The types and roles of the output attributes are copied from the input. If there was a segmentation attribute, its type is known from a parameter of this operator.

Estimates The value lists of the input attributes can be copied to the output. The number of entities in the output, and the value distributions for the output attributes, are not known.

Appendix B: Templates

This appendix lists the templates that are explained in section 6.5.3.

B.1. Aggregation

Problem description Large data sets may provide too detailed, fine-grained information for direct mining. A coarse-grained representation is desired. This template applies to data about some products a company sells; the products are organised in a taxonomy of product groups. Rather than looking at information about single products, information about product groups is desired. Further, some statistical values describing the data set are needed, such as the distribution of low-priced, medium-priced (etc.) products.

When several data sets are available, one may want to extend the information in one of them using data from another one. In this template, customer information is available in a data set to which the product data is related, by a many-to-many relationship indicating which customer has bought which product. The customer data is going to be extended by an attribute that contains, for each customer, the frequency of buying the most frequent product.

Solution description The DISCRETISATION operator (A.5.1) is used to encode price groups (low, medium, etc.) based on the detailed prices. Then AGGREGATION (A.1.4) is applied, using the product group attribute for grouping. The operator computes the sum of prices and the number of products per product group. In addition the distribution over the price groups is computed based on the encoding computed in the previous step.

The aggregation over multiple concepts is solved by the operator AGGREGATE BY RELATIONSHIP (A.2.2).

Preparation concepts demonstrated DISCRETISATION; generalising over a hierarchy over data items (see section 3.3.1); aggregation of data over a single and multiple input concepts.

MiningMart concepts demonstrated The MiningMart operator that corresponds to AGGREGATION offers a facility for computing the distribution counts of the distinct values of a given attribute. The output concept has one attribute per distinct value. This is illustrated by this template.

B.2. ChangeDistributionOfValues

Problem description Sometimes a data set has an undesirable distribution with respect to some attribute, for example the target attribute for mining. Before applying a mining

algorithm, one may want to correct the distribution. In this template, the input data is supposed to provide personal data, but 80% of the data applies to male people. For the output an equal distribution of the data for each gender is desired.

Solution description The input is split into two data sets, one for each gender. From both parts an equal number of entities is randomly sampled. The two samples are then unified again.

Preparation concepts demonstrated SEGMENTATION (A.6.1); sampling (A.1.3); changing distributions (section 2.1.3, see also (Pyle, 1999)).

MiningMart concepts demonstrated Pseudo-parallel processing as briefly explained in section 6.4. The output concept of the SEGMENTATION step represents both data sets, for male and female persons. Since these data sets have the same structure, they can be represented by the same concept. MiningMart applies all succeeding steps to all data sets attached to an input concept, until the MiningMart operator UNSEGMENTATION unifies all data sets attached to the input concepts, so that the output of that operator has only one data set again. This facility can be very convenient in real applications (Euler, 2005d); it is demonstrated in this template.

B.3. ChangeNominalAttribsToNumeric

Problem description Some mining algorithms can only process numeric input data (compare table 2.1 on page 17). When discrete attributes are present, they must be converted.

Solution description The operator DICHOTOMISATION (section A.3.1) produces binary output, but since the two output values are 1 and 0 in the MiningMart version of this operator, they can also be interpreted as numbers. This demonstrates one method of re-encoding discrete attributes. Another solution, also included in this template, is to simply map discrete values to numbers. If any ordering can be found in the discrete values, the numbers should reflect it. In this template the discrete values of the `Wind` attribute of a weather data set, `Stormy`, `Breezy` and `Still`, describe wind conditions and reflect decreasing wind strength. So there is an ordering, which is retained in the output by mapping `Stormy` to 3, `Breezy` to 2 and `Still` to 1.

Preparation concepts demonstrated Changing conceptual data types from discrete to continuous; respecting the ordering of discrete values; DICHOTOMISATION; VALUE MAPPING.

MiningMart concepts demonstrated To change the old values to the new ones, the parameters of (the MiningMart version of) VALUE MAPPING must provide a unique mapping. The operator has two parameters for the old and new values; to establish the mapping, these parameters are *coordinated*, which means that the first value of the first

parameter matches the first of the second, the second pair of values matches as well, and so on. Parameter coordination is declared in the MiningMart framework by a specific constraint, see section 6.3.2. It is signalled in the MiningMart GUI when the parameters are edited.

B.4. ChangeUnitOfMeasurement

Problem description Sometimes continuous values of attributes are given on a different scale, or a different unit of measurement, than desired for the final data representation. Examples are currency values or physical measurements. In this template, rain values of a weather data set are given in litres, but needed in millilitres.

Solution description The SCALING operator is not useful for this task, as it relies on fixed upper and lower boundaries of the new scale. Instead, ATTRIBUTE DERIVATION can be used with a simple formula for changing the input values.

Preparation concepts demonstrated ATTRIBUTE DERIVATION with a simple formula.

MiningMart concepts demonstrated There are two MiningMart operators corresponding to ATTRIBUTE DERIVATION. The one used in this template can only take arithmetic formulas which are expressible in SQL, and apply them on single entities, so that properties of other entities cannot be included. The operator takes the conceptual-level names of attributes for its formula, and translates them to the technical level internally, as can be seen in this template. A more general MiningMart operator that realises ATTRIBUTE DERIVATION fully is described in section 7.2.1.

B.5. ComputeAgeFromBirthdate

Problem description This is a very common data preparation task. Personal data sets usually contain people's birth date rather than their age, since only the former is constant over time. The current age at the time of mining, however, provides more relevant information.

Solution description ATTRIBUTE DERIVATION can be used with a specific formula. However, MiningMart provides a convenience operator that extracts years, months or days of the week from date values, since the format of the latter varies with the underlying database system. Thus the operator hides the technical level of storing dates. ATTRIBUTE DERIVATION is therefore used as a second step that computes the current age (in the example, as of August 2004) using the previously extracted year and month values.

Preparation concepts demonstrated Handling date and time related information; ATTRIBUTE DERIVATION.

MiningMart concepts demonstrated As discussed in section 5.2, MiningMart does not produce a new output concept when ATTRIBUTE DERIVATION or one of its specialised versions are applied. Instead, the system adds the new attribute to the input concept. The template illustrates that the old (birth date) attribute, from which the desired attribute is derived, as well as intermediate attributes, are still present in the concept that is being prepared. Thus an ATTRIBUTE SELECTION operator is used to yield the version of the input concept that has only the desired attributes. Such a final ATTRIBUTE SELECTION operation shows up in several templates.

B.6. CorrectTypos

Problem description Another very common preparation task is to correct misspellings in discrete values.

Solution description The operator VALUE MAPPING is used to map all recognised misspellings of a discrete value to the correct value.

Preparation concepts demonstrated Data cleaning; VALUE MAPPING.

MiningMart concepts demonstrated Mapping several old values to one new value can be done by listing the old values in a single parameter entry, which is coordinated (see template ChangeNominalAttribsToNumeric) with the parameter entry for the single new value.

B.7. Discretisation

Problem description Another very common preparation task concerns the discretisation of continuous attributes into discrete values. In this template the amount of rain fallen at some location on some day is available in the input data. In the output, only a few discrete values describing the amount of rain qualitatively are desired. How many discrete values there should be is not necessarily known.

Solution description Different variants of DISCRETISATION (as introduced in section A.5.1) are applied to demonstrate different solutions.

Preparation concepts demonstrated DISCRETISATION in several variants.

MiningMart concepts demonstrated In MiningMart, there is not a single discretisation operator whose parameters determine the method of discretising, but there is a different operator for each method. The most important ones are included in the template.

B.8. ExtractIntegerTimeIndexFromDate

Problem description A time series is given in the input data (here, weather data has been collected over time). The time index is thus given by date entries. To simplify the further analysis, a monotonically increasing integer time index is desired.

Solution description Since the weather data has been collected on a daily basis, the number of days since a particular date provide a suitable integer time index. Rather than using ATTRIBUTE DERIVATION with a complex formula, three versions of it with simpler formulas are applied. The first is again the MiningMart convenience operator that encapsulates the extraction of years or months from date values. The second computes the number of days since the first of January from the month and day values. The third computes the number of days since the particular start date, by using the result of the second step.

Preparation concepts demonstrated Handling date and time related information; ATTRIBUTE DERIVATION.

MiningMart concepts demonstrated Like in the template ComputeAgeFromBirthdate, the MiningMart operator that extracts simple representations for the year, month, day, hour or minute occurring in a date or time attribute is demonstrated.

B.9. GeneralisationOfAnAttribute

Problem description Sometimes an attribute takes values over which a taxonomy can be defined. The taxonomy may or may not be reflected in the data (by providing parent values for every value). One may want to use the higher levels of the taxonomy rather than the lower ones for analysis. In the template, data about cities is given, but the interest is in data about their regions.

Solution description In this template the taxonomy is not in the data, but is explicitly introduced in the template. VALUE MAPPING is suitable for this, as it can introduce a new region value for all cities that belong to that region. The template demonstrates two taxonomy levels by also mapping regions to states, in a second step. Further, the template demonstrates AGGREGATION over the first level, similarly to the template Aggregation (see above); here, the average number of inhabitants per region is computed, which would not have been possible using the input data directly, since the region information is missing there.

Preparation concepts demonstrated Introducing background domain knowledge; using taxonomies over domain values; VALUE MAPPING.

MiningMart concepts demonstrated MiningMart versions of VALUE MAPPING and AGGREGATION.

B.10. InformationPreservingDataCompression

Problem description Large data sets may be unwieldy for analysis. The desire is to reduce the amount of data while losing as little information as possible.

Solution description One possible approach to the problem, demonstrated in this template, is to apply automatic attribute selection using an information gain criterion for attribute selection. This reduces the dimension of the data. Due to this first reduction, the data may then contain duplicate entities, since some entities may have differed only in values of attributes that are now removed. Therefore duplicate entities are removed in the second step of this template, using a convenience version of ROW SELECTION that MiningMart offers for this task. This may involve a considerable reduction of the data volume, depending on the characteristics of the data set.

Preparation concepts demonstrated Automatic attribute selection; data compression; duplicate entity removal.

MiningMart concepts demonstrated One of the operators of automatic attribute selection is demonstrated (see (Berka et al., 2002) for the full list), as well as the convenience operator for duplicate entity removal. The first operator can use a sample of the whole data if necessary. It applies a greedy search over the attributes, adding attributes as long as the information gain with respect to a target class increases, and as long as the number of attributes does not exceed the threshold specified as a parameter of this operator.

B.11. IntegrateDifferentDataSources

Problem description Data sets that are linked by a relationship have to be joined to yield a single table, as desired for most data mining algorithms. But it is desired to prepare the data sets separately before joining, in order to reduce the amount of data in the expensive join operation.

Solution description The data sets are prepared separately. Then the relationship that linked the original concepts is re-created between the output concepts of the separate preparation processes. A special MiningMart operator is available for this. The relationship is technically realised by a cross table, whose name is a parameter to the operator, if it is a many-to-many relationship. The operator creates a new cross table on the technical level, ensuring that its references to the two concepts to be joined are valid. On the conceptual level, the operator simply creates a relationship that links the two input concepts.

The template also demonstrates two options to make use of the new relationship: a join that uses the relationship for key specification (which is more convenient than having the user specify the keys), and AGGREGATE BY RELATIONSHIP (see the template Aggregation and sections A.2.2 and 7.2.3).

Preparation concepts demonstrated Use of relationships; creation of relationships; joining data sets.

MiningMart concepts demonstrated To ensure the validity of the created relationship, the two concepts between which it is created must be connected to a database *table* on the technical level, because views cannot be constrained by primary keys. Therefore the concepts resulting from the separate preparations must be *materialised* together with their primary keys, for which a special MiningMart operator is available. The template demonstrates the materialisation. This is a point where the strict separation between conceptual and technical level is weakened. Compare section 7.3.

B.12. MaterialisationDemo

Problem description In longer preparation graphs, efficient data processing can become a problem. On the one hand it is inefficient to store the output data of every preparation step permanently, as this requires too much storage space (consider large data tables prepared by dozens of steps as in chapter 5). Since many preparation steps make only minor modifications, the storage would also be highly redundant. On the other hand, processing all data in main memory can quickly become inefficient as well, if the chain of preparation steps is not rather short. When the data is stored in databases, *views* are a good solution to avoid redundant storage, but deeply nested views on views, resulting from long preparation chains, are again inefficient to read data from.

Solution description A solution to this problem is to define certain points in the preparation graph where data should be stored permanently (this is called *caching* in chapter 8, and *materialisation* in section 7.3). Inbetween these points, processing is done by views or in main memory. One heuristic to determine suitable points is to consider steps whose output is "consumed" by several following steps, since this means that data is read several times from the considered output. The template demonstrates the use of the MiningMart operator for materialisation of views in exactly such a situation. Section 7.3 describes how such suitable materialisation points are automatically found and realised in MiningMart.

Preparation concepts demonstrated Materialisation or "caching"; efficient data handling.

MiningMart concepts demonstrated Materialisation of database views.

B.13. MissingValueHandling

Problem description The problem of missing and empty values is introduced in section 2.1.3.

Solution description The template illustrates four approaches to dealing with missing values. One is to delete entities with missing values. The second is to fill the values with a default value. The third is to fill them with values that are randomly selected, but in such a way that the overall distribution of existing values of the attribute concerned does not change. The last approach is to use entities where the value exists to train a machine learning algorithm that can predict the value for entities where it is missing. This last approach is implemented by a convenience operator, since it involves a complex subprocess: selection of training and test set from the input, training the model, applying the prediction function, and merging the predicted with the existing values.

Preparation concepts demonstrated Missing value handling, with simple and sophisticated methods.

MiningMart concepts demonstrated The operator for replacing missing by predicted values uses an external support vector machine (SVM) implementation. On the technical level, the support vectors and the kernel function needed for prediction are stored in database tables between learning and prediction. A PL/SQL function which is created by the MiningMart compiler (see section 6.4) realises the prediction. For more details see section 7.2.5.

B.14. Normalisation

Problem description Values of continuous attributes may have to be rescaled to lie within given bounds. For example, before applying a support vector machine (SVM), scaling all attributes to the range from 0 to 1 is advisable.

Solution description The operator SCALING provides the desired functionality. The template demonstrates two ways of scaling, linear and logarithmic.

Preparation concepts demonstrated SCALING

MiningMart concepts demonstrated Two scaling operators.

B.15. PivotisationDemo

Problem description As mentioned in section 4.1, sometimes the organisation of a data set needs to be changed such that meta data (or schema elements) become data, and vice versa. The template illustrates such a case. It prepares a data set with weather conditions measured by different sensors. In the input, there is one attribute with qualitative (discrete) values for wind conditions, and another with qualitative values that describe the overall weather tendency. The desired output is to have an attribute for each of the occurring wind conditions, and also one for each of the weather tendencies. These new attributes are filled with values from another sensor (amounts of rain in the template).

Solution description The operator PIVOTISATION (section A.3.2) provides the desired functionality. The template illustrates 2-fold pivotisation (one new attribute for each combination of weather tendency and wind condition; compare section A.3.2). It also combines pivotisation with aggregation, as this is often desired, but the operator can also omit aggregation.

The template further includes the reverse operation. Since the aggregation cannot be reversed, the output of reverse pivotisation does not match the input to the first pivotisation operator exactly. However, the structures of the data sets (their schema) does match.

Preparation concepts demonstrated Exchanging schema and data elements; PIVOTISATION; n-fold pivotisation; reverse pivotisation.

MiningMart concepts demonstrated The slightly complex use of the MiningMart operators PIVOTIZE and REVERSEPIVOTIZE is exemplified in this template. See also section 7.2.2.

B.16. PrepareAssociationRulesDiscovery

Problem description For frequent itemset or association rule mining (for an introduction see (Agrawal et al., 1993) or others), specific data representations are needed. This template considers a particular representation, which is directly suitable for the rule mining operator in MiningMart. However, for other rule mining implementations, such as the one in Yale, this representation has to be changed.

The given representation has one entity for each product in each transaction. The input concept thus has one attribute each for customer ID, product ID and transaction ID. The desired representation is to have one entity only per transaction, with boolean flags indicating for each product whether it has taken part in the transaction.

Solution description The operator DICHOTOMISATION (section A.3.1) is applied first, to create the boolean flags (1 or 0) indicating the presence of a product in a transaction. Since the resulting concept still has one entity per product, instead of one entity per transaction, AGGREGATION (A.1.4) is applied next, with the customer and transaction ID as group-by attributes, and *maximum* as the aggregation operator. Whenever a product occurs in any of the input entities that belong to the same transaction, the maximum value is 1, otherwise 0. This is an example of interpreting a discrete (binary) conceptual data type as numerical on the technical level. The output concept now has one entity per transaction and per customer, and can be used as input for mining algorithms that expect this representation.

Preparation concepts demonstrated DICHOTOMISATION; AGGREGATION; flexible mapping of conceptual to technical data types (see section 3.3.1 and criteria 17 and 18 in appendix C).

MiningMart concepts demonstrated Applying a mining operator (here APRIORI); preparing the result of preparation to be used as input for a Yale experiment; using *loops* in an operator (explained in section 6.3.2).

B.17. TimeSeriesAnalysis

Problem description Time series data (or, more generally, value series data (Mierswa & Morik, 2005)), is usually not directly accessible for mining algorithms because of its representation as a series rather than a collection of examples to learn from. An important preparation task is therefore to create such a collection of examples.

Another common preparation task is to encode seasonality. The series may have a monotonically increasing (time) index, such as dates, but may be based on real-world phenomena that have a cyclic nature, such as the days of the week or the seasons of the year. One often aims to encode the current phase of the cycle in the data.

Many other time series preparation problems exist, but these are the most common and basic ones, and the ones that are currently supported by the MiningMart system.

Solution description WINDOWING (section A.3.4) is the operator that transforms a linear series into a collection of examples. It is illustrated in this template on weather data.

Based on a windowed representation, a weighted average of values in the window can be computed; using a distance of 1 between the windows means to smooth the series values (compare (Pyle, 1999)). This is also illustrated in this template, by a specific operator that MiningMart provides for this purpose.

Seasonality encoding is exemplified based on the output of the template ExtractIntegerTimeIndexFromDate. The monotonically increasing integer time index modulo 7 is computed (by ATTRIBUTE DERIVATION), in order to encode the weekly cycle (the time index reflects daily measures in this data set, compare template ExtractIntegerTimeIndexFromDate).

Preparation concepts demonstrated WINDOWING; encoding markers for cyclic phases; smoothing of series values.

MiningMart concepts demonstrated The slightly complex MiningMart operators for windowing and computation of a weighted average over a window are demonstrated.

Appendix C: List of Criteria

This appendix presents all specific criteria that serve to compare KDD software packages, as explained in section 8.3.3. The numbering of the criteria continues the numbering started in section 8.3.2.

C.1. Data access

As was said in section 3.1.2, the two common types of data sources are flat files and databases. Thus one criterion is the ability to load data from at least these sources:

7 Data formats: At the very least, flat files in various common formats, such as comma-separated or sparse representations, and ODBC, the open database connection standard, must be accessible. This applies to data input and output. More precisely, the following boolean features form this criterion ($m = 6$):

- possibility to read flat files at all;
- possibility to specify any column-delimiting character when reading flat files;
- possibility to read the first line of a file as attribute names;
- possibility to read tables via ODBC;
- possibility to read sparse representations;
- applicability of all of the above to both input and output of data.

However, once the data is loaded into the KDD software, every data transformation step produces intermediate data tables. When handling large data sets, storing all intermediate tables would multiply the size needed by the original data set by a large factor (roughly corresponding to the number of data preparation steps). Thus only some intermediate results should be stored. The question of data storage is an important feature to distinguish KDD software tools. Some leave all processing to the database, so that the data never leaves the database. Others rely fully on the local file system.

Processing in databases has the advantage that structured search is possible on every intermediate concept, and that the use of views allows this essentially without consuming extra storage. Further, databases are usually installed on fast hardware with large storage devices; see also (Musick & Critchlow, 1999). However, database management systems include features such as transaction safety and concurrent access, which are not essential for KDD but may slow down processing. In contrast, using the file system might be faster, but does not allow structured search on intermediate results; further, the file system of the workstation from which the KDD application is controlled may not be sufficient to handle large volumes of data. As explained in section 2.1.4, this pertains more to data preparation and deployment than to mining, as the latter should be performed in main memory anyway.

Setting	Time for short processing chain in minutes	Time for complete model application in hours
DB to DB	129	39.4
DB to file	104	
File to file	29	7.8
File to DB	68	

Table C.1.: Comparison of execution times.

In order to compare the two data handling approaches, a few experiments were done in the context of this work. A short data preparation chain with three attribute derivations and one attribute selection was applied to the CDR table with 61 million records (described in section 5.6). This data preparation chain was executed using four settings:

1. inside the database, starting and ending with a materialised table (DB to DB);
2. reading from the database table, processing in main memory of the client (in batches that fit into main memory) and writing to a result file (DB to file);
3. reading from and writing to a file, processing in batches in main memory of the client (file to file); and
4. reading from a file, processing in batches in main memory of the client, and writing to a database table (file to DB).

The last setting can be relevant when data is collected from different sites to a central database, for example in distributed data mining scenarios. Depending on the application, one may want to prepare the data before combining it with data from other sources, in order to reduce the global amount of data. Thus there is some data preparation to be done on the distributed clients' file systems before loading the data to a central site.

Setting 1 was implemented in the KDD tool MiningMart which accessed an Oracle database installed on a Sun Enterprise 250 server with 1.6 GB of main memory and two 400 Mhz CPUs. For the other three settings, Clementine (see section 8.5.2) was used in the standalone version without a server, on a Windows client with 512 MB main memory and a Pentium 1600 Mhz CPU which was connected to the Oracle database via ODBC and a 100 Mbit/s Ethernet connection. The two KDD tools are described in section 8.5. The data table that was processed takes more than 2 GB of storage space in the database, so that processing could not take place completely in main memory in any setting.

Table C.1 shows the execution time for each setting. In setting 2 and 4, most of the processing time is spent on reading or writing to the database, respectively. The purely file system based processing (setting 3) is fastest. This finding is repeated when the processing time of the complete data preparation part of the model use case (chapter 5) is compared for the first and third setting. It is also consistent with the experiments reported in (Musick & Critchlow, 1999) for general data access, and (Sarawagi et al.,

1998) in the data mining context, where rule mining was tested in a number of different data handling scenarios, and caching data on the file system turned out to be the fastest scenario. An interesting approach to remedy the database efficiency problem is presented by Gimbel et al. (2004), who use pipelining and a management strategy to keep the data sorted according to various indexes. This approach explicitly takes KDD-related operations into account. However, it is not yet implemented in practically used database management systems. In practice today, whether the advantages of structured search and efficient storage that databases offer are worth the performance loss is dependent on the application. So the next criterion is obtained.

8 Data processing: Ideally, the KDD software should be able to process data both inside a given database and on the file system. If both places of processing are possible, the user must be able to specify which one to use at any point in the processing graph. This allows to distribute the computing load to the appropriate hardware. Hence, $m = 3$.

In databases, views can be employed for intermediate results, which take essentially no extra storage space. However, processing deeply nested views as resulting from a long sequence of data preparation operators is slow, so that a materialisation should take place at regular points in the data flow. Most suitable are those points in the operator dependency graph (section 4.4) where the output of one operator is consumed by several other operators. A similar argument holds for flat file based processing: here it must be possible to state which intermediate tables of a preparation chain should be stored on the file system. There should be a mechanism for this which is independent from dedicated data output operators, since such operators cannot be connected to further processing steps, and thus interrupt the preparation graph inconveniently. This leads to the following criterion.

9 Caching control ($m = 2$):

- possibility to specify the points of materialisation/data storage during processing;
- independence of this option from dedicated output nodes.

10 Caching size estimation: The size needed for storing an intermediate table to a file, or for materialising a table, should be estimated, at least roughly, by the software before the execution of the preparation graph. How such estimations can be done based on metadata is discussed in section 3.3.3. See also criterion 25. This criterion is boolean.

11 Automatic caching: For long preparation chains or at end points of the data flow, caching of results should be automatically done, or at least offered, by the software, so that no resource-intensive process is started whose results are inadvertently not stored. Moreover, sometimes caching is required by specific circumstances of which the user may not be aware. In two tools examined for this work, long preparation chains on big data sets had to be separated into several parts, each storing the output in a specific file for the next part to read from, because otherwise some hidden temporary files that the tools employed to store the several intermediate results (rather than only one result as when

caching) would have got too large for the available disk space. Thus the need for caching must be recognised by the software. So $m = 2$:

- automatic caching at end points of a process is done;
- automatic recognition of the need for caching is done.

12 Caching transparency: The files used for caching, or the materialised tables, must be accessible, and must be clearly linked (e.g. by name) to their concept or to the operator which outputs the data stored in them. This enables the user to follow the data storage processes and arrive at own estimations of resource consumption. So $m = 2$ (accessibility and linkedness).

13 Data inspection: Intermediate data tables (the extensions of the concepts) must be inspectable from the tool. This facilitates the control of the ongoing work by the user. This criterion is boolean.

C.2. Data modelling

14 Attribute import: The names of attributes must be automatically recognised from database sources. For flat files, it is common to reserve the first line of the file for attribute names; if such files are read, this must be recognised by the software. Also, common formats storing attribute information in a separate file, such as ARFF, should be supported. However, attribute names must not be fixed. Thus $m = 3$:

- automatic recognition of column names, to be used as attribute names;
- possibility to edit attribute names;
- support for reading attribute information from a separate file.

15 Conceptual data types: The distinction between the actual storage type of data and the way it is used conceptually should be made. This criterion is boolean.

16 Type recognition: A strong mechanism to automatically recognise the technical as well as the conceptual data types of all attributes when importing data is a must. For large data sets, recognition can take quite some time; thus recognition based on a sample of the data, or recognition at a later point in the preparation graph, must be supported, and this must be controllable by the user. The user must have the option to specify the types by hand, to avoid long recognition processes on large data sets, or to correct wrongly recognised conceptual types. Hence $m = 5$:

- automatic recognition of technical data types;
- automatic guessing of conceptual data types;
- possibility for the user to specify when recognition takes place;
- availability of recognition based on a data sample;
- possibility to specify and change the conceptual data type by hand (at any time).

17 Flexibility of type mapping: The relation between the conceptual data type of an attribute and its technical counterpart must be transparent, flexible and controllable by the user. It must be changeable at any point in the preparation graph, not only at the beginning when data is imported. Thus $m = 3$ (transparency of mapping, changeability by user, and independence from import).

18 Robustness of type mapping: If a preparation operator uses a technical data type in a way not consistent with the conceptual data type it is currently mapped to, the mapping ought to be changed, perhaps with a warning to the user, but this should not lead to an error as it is a rather common situation (see section 3.3.1). This criterion is boolean.

19 Data characteristics recognition: Similarly to type recognition (criterion 16), the set and distribution of values occurring in the data must be recognised by the software during import or at a later point, based on the entire data or on a sample, and control over this must be given to the user. For continuous attributes, the range of occurring values instead of the set of all values should be stored. Again, it must be possible to specify all of this information manually. See also criterion 34. This criterion can be extended by boolean features concerning further data characteristics, such as average values, number of unique values in an attribute, and so on. Here, $m = 6$:

- recognition of range of values of a continuous attribute;
- recognition of distribution of values of a discrete attribute;
- recognition of the number/percentage of missing values;
- possibility for the user to specify when recognition takes place;
- availability of recognition based on a data sample;
- possibility to specify and edit data characteristics by hand (at any time).

20 Data characteristics deployment: The data characteristics from criterion 19 must be available during the declaration of the KDD process in the tool. For example, for the operator VALUE MAPPING, the values of the selected input attribute (to be mapped to other values) must be selectable during operator specification. For this functionality, the availability of data characteristics in separate charts or tables is not enough. This functionality is the boolean criterion.

21 Attribute roles: Support for the four roles identified in section 3.3.2 must be given. That is, the user must be able to specify a role for each attribute. Thus $m = 4$ for the three roles *label*, *predictor*, *key* and *no role*.

22 Attribute matching: On some occasions, attributes of different concepts are mapped to each other. For example, when joining concepts, no duplicate attributes must occur in the output concept, even if the same attribute is present in more than one input concept, a situation that should be recognised by the software; at the same time the keys of the input concepts must be matched. Similarly, the UNION operator requires a matching of all attributes of the input concepts. As another example, when importing learned models

that were exported using PMML (see criterion 55), the attributes on which the model is applicable must be matched to the attributes in the concept to which it is going to be applied. The KDD software can save work by recognising matchable attributes by name and conceptual or technical data type, especially as hundreds of attributes per concept are not uncommon in some applications (more than 90 attributes are used for mining in the model use case, in chapter 5). However, of course the matching must be editable by the user. Hence $m = 2$ (automatic matching and editability).

23 Data type inference: When deriving a new attribute, its technical data type must be inferred. The conceptual type is never uniquely determinable but can be guessed; default types often suffice. So $m = 2$:

- inference of technical data type of derived attributes;
- guessing of conceptual data type of derived attributes.

24 Abstract data model: Chapter 3 has argued that the intermediate data representations created during the KDD process are an important source of information, and that they should be structured as clearly as possible. The two most important aspects of abstract data modelling for KDD are used as features here ($m = 2$):

- representation of data at a conceptual level;
- structuring of data at conceptual level, reflecting the KDD process and how it produces intermediate results.

25 Characteristics estimation: Inference and optimistic estimation of data characteristics are introduced in section 3.3.3. How and when data characteristics can be estimated is specified for each preparation operator in chapter 4.

The importance of this criterion can be seen when noting that some operators, like PIVOTISATION (section A.3.2), rely on knowledge of which values occur in an input attribute (for pivotisation, the index attribute). Tools in which this information is neither inferred nor manually editable force the user to execute all steps that lead to the operator where this information is needed, in order to then recognise the available values automatically in the input. This situation was actually encountered by the author when implementing the model use case in some tools. However, the execution may take a long time, which is unacceptable during the development of an application, when the usefulness of any preparation operation has not been established yet.

Many boolean features can be identified for this criterion based on the estimates in the operator specifications in chapter 4. However, in the tools examined here they are not distinctive. Therefore this criterion is boolean, and is fulfilled if a tool offers any inference or estimation of data characteristics.

C.3. Preparation process

26 Syntactic validity checks: The software must differentiate between syntactically valid and invalid preparation graphs, and support the user in finding reasons for invalidity.

Invalid graphs can result from, for example, deleting attributes at one point which are needed at another point, or by changing data characteristics, through recognition or manually, which some operator's specification depends on. The boolean features are ($m = 4$):

- indication of invalid nodes in the graphical representation of the processing graph;
- indication of ill-formed derivation formulas;
- indication of well-formed derivation formulas that use non-existing attributes;
- clear error messages to hint at the reasons for invalidity.

27 Propagation of changes: This is one of the most important criteria for large applications. In complex preparation graphs, many dependencies exist between attributes and concepts at different locations in the graph. A simple example is a derivation of an attribute early in the data flow; this attribute is available in every following step. If the user decides to rename the derived attribute, the new name must be propagated through the graph. Similarly, if the step deriving that attribute is deleted from the graph, all later steps and concepts must be adjusted. Some steps may become invalid in the process; this should be displayed. These adjustments must be done automatically, as they may be rather complex. Compare section 6.6.

While such propagation of metadata through the graph should be as robust as possible, it must not destroy invalid metadata. For example, if the deleted attribute is used as an input to a complex derivation of another attribute, the formula for derivation must not be deleted but kept in an invalid state, as the user might wish to modify the formula to a different input attribute, for example.

The importance of this criterion, as well as that of criterion 25, was also independently recognised by AlSairafi et al. (2003).

Propagation concerns attributes and concepts, as well as their names and types. As section 6.6 argues, the operations that must be supported by propagation are addition, deletion, renaming and retyping, so there are four boolean features that a KDD tool must fulfil. A fifth feature checks the cautious deletion of dependent information, as explained above. Thus $m = 5$.

28 Operator transparency: The reason for using pre-programmed operators is to save the work of detailed specifications of data transformations. For example, using a discretisation operator whose input is simply the number of intervals spares the user the computation of suitable interval boundaries, because the operator does this automatically. Nevertheless, the results of such automatic specifications must be inspectable and manipulable by the user. Besides giving more control to the user, this is also very important considering that some transformations have to be reversed for deployment (see section 2.1.6), which is only possible for the user if all details of the transformation are accessible (but see criterion 52). So $m = 2$ (inspectability and changeability of derivation or selection formulas that are set up by the tool rather than the user).

29 Availability of operators: All operators listed in appendix A must be available in the tool. This criterion could be extended to use all special options of the operators, but

this would result in a high value of m and the criterion would subsume too many details. Section 8.6 provides a detailed table about the presence or absence of each operator and many of their special options in every tool examined for this work. Thus $m = 19$ is chosen here.

30 Grouping operators in preparation tasks: The association of preparation operators to high-level preparation tasks, as done in chapter 4, is an important guideline for inexperienced users. It helps to find suitable operations for solving particular tasks. This criterion is boolean.

31 Intermediate views on data: The input to a KDD process is a number of tables. In a given line of processing, one or more of these tables are changed. Every processing step produces a new view on the data. To enable the user to view this *current* set of tables after a given processing step, there should be an option to display only this set. This criterion is boolean.

32 Attribute derivation support: For attribute derivation, it must be possible to set up any formula, using basic functions, some of which are listed in section A.5.4. The availability and meaning of these functions must be displayed to the user during set-up of a formula. If such features are lacking, the user cannot know which functions are available and what they compute, leading to frustrating trial-and-error procedures to arrive at correct formulas. So $m = 2$ for the availability of a choice list of provided functions, and for their documentation in the interface where the formula is set up.

33 Iteration of attribute derivation: The operator ATTRIBUTE DERIVATION (section A.5.4) must be configurable to derive more than one attribute based on the same formula, using automatically a specified change in the derivation formula for each derived attribute. For example, given an attribute that contains the months of a year, one new attribute for each month might be created that contains derived values of another attribute. The KDD tool should offer to set up the formula once, with a variable that iterates over the values of the month attribute. This iteration process should create as many attributes as there are values in the month attribute. But the derivation might also iterate over several input attributes, rather than the values of one attribute, or over values outside the data, for example an increasing counter. If an own attribute derivation operator for each new attribute had to be used instead, this would require much more work to set up the operators, and the graph structure would become unnecessarily complex. Thus $m = 3$:

- possibility to use iteration over attribute values of a given attribute, to achieve "parallel" derivation of several attributes;
- possibility to use iteration over attribute names of a given concept, for the same purpose;
- possibility to use iteration over a given value list, again for the same purpose.

34 Independence from data: An operator chain is declarative, thus it ought to be possible to set it up in the absence of input data. This is required, for example, when the data has not been processed far enough yet, so that metadata inference (criterion 25) is not possible. Another scenario is grid-based data mining, in which the allocation of computational resources is independent from the declaration of the KDD process (compare section 2.2 and (AlSairafi et al., 2003), where this criterion is also discussed). Though many operator specifications depend on metadata (see criterion 27), it was also stressed in criterion 19 that it must be possible to provide metadata by hand. This criterion is boolean.

35 Empty data sets recognition: Sometimes operators produce concepts that have an empty extension. This can happen after a join or a row selection. Not all tools recognise this but it can be the source for errors. An error message should be given when this happens. This criterion is boolean.

36 Representation of data flow: The interdependencies of operator instances can become rather complex in big applications, so that they must be graphically displayed. If this feature is lacking, the user has to rely on intermediate data set names to understand the connection between steps. This criterion is boolean.

37 Pseudo-parallel processing: Representing several data tables of the same schema with one element only, in order to allow the pseudo-parallel processing of data as motivated in section 1.1.1, allows to save much user efforts during modelling. This criterion is boolean.

38 Support for chunking: As discussed in section 4.4, it contributes to keeping an overview of complex preparation graphs if they can be partitioned into chunks. The structure of chunks should be most flexible. More precisely, $m = 2$:

- chunks must not be restricted to atmost one input and one output concept;
- chunks must be nestable into hierarchies.

39 Graph structure: The preparation graph (see section 4.4) is, in general, a directed acyclic graph (DAG) without further restrictions. One tool evaluated for this work imposes the restriction that there can be no two different paths from a given operator to a second one. Yet such a situation is rather common and occurs several times in the model use case (chapter 5). This criterion is boolean, and is fulfilled if the DAG is unrestricted.

40 Execution transparency: When a data preparation graph is executed, progress should be clearly indicated to the user. This includes ($m = 7$):

- displaying information which step is currently being executed;
- displaying the number of data rows already processed in this step;
- displaying the number of data rows yet to process in this step;
- displaying the storage space consumed for the current execution;
- displaying the storage space expected to be required for the current execution;

- displaying the time the execution has consumed so far;
- displaying an estimation of the total execution time required.

41 Execution automation: An automatic execution of preparation graphs must be possible, to automate long-time consecutive or conceptually parallel experiments. More precisely, there are three aspects to be considered ($m = 3$):

- scheduling of execution runs to particular points in time;
- the option to automatically change parameters of the processing graph for each execution, so that the same graph can be run on batches of data sets, or with some specific parameter looping through its range for each execution;
- the possibility to specify the order of execution for different parts of the graph.

42 Execution administration: An execution run of a KDD process is an experiment. Information pertaining to this experiment must be automatically stored. This helps to organise the user's work when a lot of experiments are run, or when the execution times exceed the user's memory span. In particular ($m = 7$):

- information which steps were executed in an experiment must be stored;
- the number of rows in input and output must be stored;
- start and end time and date must be stored;
- the names of any involved files or database tables must be stored;
- each experiment must be given a unique ID, which must be stored;
- each experiment must be commentable with free text;
- this information about stored experiments must be searchable.

43 Execution in background: Editing parts of the processing graph must be possible during an execution of the graph. That is, the execution should run in the background, without blocking the system. Yet edits should not pertain to the running execution. This criterion is boolean.

44 Export transparency: Obviously, a way of storing and reloading the data preparation graph with all its parameters is needed. A particular point is that the storage format should be transparent, preferably based on an XML formalism. This gives extra flexibility to the user for complex ways of editing the graphs which are not foreseen by the graphical user interface. But more importantly, it makes the graphs at least readable when the KDD tool that produced them is no longer available, making old applications usable to some degree even when the computing environment changes. This criterion is boolean (transparent storage format).

45 Editing flexibility: On each level of the KDD model (data types, parameters, operators, and chunks), copy and paste functions must be provided. All KDD tools examined in this work offer this. However, flexible editing must also be possible for formulas in attribute derivation or row selection, especially if more than one attribute is going to be

derived in unsystematic ways not supported by the derivation operator (compare criterion 33). Some tools examined for this work lacked this option, resulting in tedious extra work in the situation of deriving many attributes. This criterion is boolean (availability of copy and paste functions for all formulas).

46 Visual graph arrangement: As some applications require complex processing graphs, the KDD software should be able to automatically arrange the nodes of the graph, the operators, on the screen in a clear fashion, for example on a grid. This criterion is boolean.

C.4. Learning control

As explained in section 8.3.1, the criteria for the mining phase in this work concern typical processing and control tasks. The main concepts are introduced in sections 2.1.4 and 2.1.6.

47 Splitting training and test set: A facility to randomly split a data table into a training and a test set, according to a given ratio, must be available. This can be realised by the operator SEGMENTATION (section A.6.1). This criterion is boolean.

48 Model evaluation: A facility to evaluate the performance of any model learned in the tool on a test set must be available, using typical performance measures. Such measures are not listed here because they depend on the type of mining task (examples are accuracy, support, intra-cluster density etc.). But at least one application-independent performance measure must be offered for every type of model. This criterion is boolean.

49 Mining subprocess support: The experiments around the application of the data mining algorithm can be usefully modelled by nested control operators such as cross validation or parameter tuning (Mierswa et al., 2003). Since the tools examined for this work offer virtually no support for this, only a single boolean criterion is used. It is fulfilled if direct support for experiments around mining is present. Although no tool here fulfils the criterion, it is included in order to stress the importance of support for mining experiments.

C.5. Deployment

50 Export of models: For deployment in an actual technical or business process, functions that are learned by the tool must be exportable into source code, to be usable outside the tool in arbitrary environments. This criterion is boolean.

51 Deployment in databases: It is very useful if a learned function can be used directly in a relational database, since operational data is likely to be stored in such databases. The learned function should be modelled in SQL or PL/SQL in order to enable this. This criterion is boolean.

52 Post-processing: To enable the post-processing of data that was "encoded" for mining (see section 2.1.6), the tool should offer an automatically created operator for any reversible transformation that was applied during data preparation. This automatically created operator can be applied to the predictions of a model and reverses the transformation, in order to get predictions from the original domain of the label attribute. This criterion is boolean.

C.6. KDD standards

53 Published meta model: Modelling processes based on a public meta model allows their system-independent publication, like in MiningMart's case base. The various advantages for reuse and education of other users are discussed in depth in chapter 6. One might introduce many boolean features based on this fundamental approach, but they would not be distinctive among the tools examined here. Thus this whole criterion is boolean.

54 CRISP support: The software should support the distinction between the different phases of a KDD process, for example by providing different graphical environments for data understanding (visualisation, characteristics computation), data preparation, mining and deployment. This criterion is boolean.

55 PMML support: Models learned by the software should be exportable to files using the PMML standard (Grossman et al., 2002). Conversely, PMML files should be importable and applicable. See also criterion 22. So $m = 2$ for import and export.

There are other standards around KDD, see (Grossman et al., 2002), but they are more oriented towards KDD developers. From the conceptual point of view of a user, the two standards above are the most relevant ones.

Appendix D: Technical level of model application

The model application presented in chapter 5 was implemented in a few KDD tools to gather experiences with their functionality, their strengths and weaknesses as discussed in chapter 8. One implementation, from which the figures in chapter 5 are taken, was done in the MiningMart system which automatically translates the conceptual data and process models to SQL, the well-known standard language used in relational database management systems today. In this section the automatically created SQL code for one chunk of the model application, the revenue data preparation chunk (section 5.5), is given and briefly explained, to provide an impression of the technical level and a contrast with the conceptual level. For better readability, the SQL code is presented here using indentations, and one type of abbreviation: column names are used instead of fully qualified column names with their paths.

The first steps join the revenue data table (called `IN_WINNINGS`) with the data selection, then remove missing values:

```
CREATE OR REPLACE
       VIEW CS_100107354 AS
              (SELECT PROFIT AS Revenue,
                      CALLER AS Caller,
                      CHURNMARK AS ChurnMark,
                      MONTH AS Month
               FROM IN_WINNINGS, TrainingSetMaterialised
               WHERE IN_WINNINGS.CALLER = TrainingSetMaterialised.CALLER)

CREATE OR REPLACE
       VIEW CS_100107347 AS
              (SELECT Revenue,
                      ChurnMark,
                      Caller,
                      Month
               FROM CS_100107354
               WHERE Revenue IS NOT NULL)
```

The following three listings are created three times, once for each of the three parallel lines of preparation (see figure 5.6 on page 82).

```
CREATE OR REPLACE
       VIEW CS_100107329 AS
              (SELECT Caller,
                      Month,
                      ChurnMark,
                      Revenue
               FROM CS_100107347
               WHERE (ChurnMark = 1) )
```

The step creating the abstract month attribute creates an SQL string defining a virtual column; this string is used again in the view definition produced by the following step, which is given below.

```
CREATE OR REPLACE
       VIEW CS_100107321 AS
              (SELECT Revenue,
                      ((CASE WHEN Month IN ('Jul 2000') THEN '1'
                             WHEN Month IN ('Aug 2000') THEN '2'
                             WHEN Month IN ('Sep 2000') THEN '3'
                             WHEN Month IN ('Okt 2000') THEN '4'
                             WHEN Month IN ('Nov 2000') THEN '5'
                             WHEN Month IN ('Dez 2000') THEN '6'
                             ELSE ('0') END))
                         AS Month1_6,
                      ChurnMark,
                      Caller,
                      Month
               FROM CS_100107329
               WHERE Month1_6 in (1,2,3,4,5,6) )
```

Pivotisation produces:

```
CREATE OR REPLACE
       VIEW CS_100107305 AS
              (SELECT Caller,
                      SUM (CASE WHEN Month1_6 = '6'
                                THEN Revenue ELSE 0 END)
                         AS Revenue_6,
                      SUM (CASE WHEN Month1_6 = '1'
                                THEN Revenue ELSE 0 END)
                         AS Revenue_1,
                      SUM (CASE WHEN Month1_6 = '4'
                                THEN Revenue ELSE 0 END)
                         AS Revenue_4,
                      SUM (CASE WHEN Month1_6 = '2'
                                THEN Revenue ELSE 0 END)
```

```
                    AS Revenue_2,
                 SUM (CASE WHEN Month1_6 = '5'
                           THEN Revenue ELSE 0 END)
                    AS Revenue_5,
                 SUM (CASE WHEN Month1_6 = '3'
                           THEN Revenue ELSE 0 END)
                    AS Revenue_3
            FROM CS_100107321
            GROUP BY Caller)
```

Then the three parallel data sets are unified again:

```
CREATE OR REPLACE
       VIEW CS_100107285 AS
              (  (SELECT Revenue_6,
                         Revenue_1,
                         Revenue_4,
                         Revenue_2,
                         Caller,
                         Revenue_5,
                         Revenue_3
                    FROM CS_100107305)
                 UNION
                   (SELECT Revenue_6,
                           Revenue_1,
                           Revenue_4,
                           Revenue_2,
                           Caller,
                           Revenue_5,
                           Revenue_3
                    FROM CS_100107299)
                 UNION
                   (SELECT Revenue_6,
                           Revenue_1,
                           Revenue_4,
                           Revenue_2,
                           Caller,
                           Revenue_5,
                           Revenue_3
                    FROM CS_100107303) )
```

Materialisation:

```
CREATE TABLE Revenues6Months AS
       (SELECT Revenue_6,
               Revenue_1,
               Revenue_4,
               Revenue_2,
               Caller,
               Revenue_5,
               Revenue_3
        FROM CS_100107285)
```

The two following attribute derivations are again reflected in the final view that is the result of this chain, and that realises the attribute selection step.

```
CREATE OR REPLACE
       VIEW CS_100107213 AS
              (SELECT Caller,
                      Revenue_6,
                      Revenue_3,
                      Revenue_4,
                      Revenue_1,
                      Revenue_2,
                      Revenue_5,
                      ((CASE WHEN (Revenue_1+Revenue_2+Revenue_3+
                                   Revenue_4+Revenue_5+Revenue_6)
                                 < 300.0 THEN ('low')
                             WHEN (Revenue_1+Revenue_2+Revenue_3+
                                   Revenue_4+Revenue_5+Revenue_6)
                                 < 600.0 THEN ('medium')
                             ELSE ('high') END))
                         AS RevSumClass
               FROM Revenues6Months)
```

Appendix E: SQL Implementation of Test Case

This appendix lists an SQL program that realises the test case described in section 8.4. The program can serve as a reference for an evaluation of a KDD tool.

```
-- This SQL script creates two small tables and realises
-- some data preparation operations on them.
-- Author: Timm Euler, University of Dortmund (April 2005)

-- create tables:

DROP TABLE Saleinfo;
CREATE TABLE Saleinfo
  ( Employee NUMBER(2),
    Month NUMBER(2),
    Sales NUMBER(4),
    Revenue NUMBER
  );

DROP TABLE Employeeinfo;
CREATE TABLE Employeeinfo
  ( Employee NUMBER(2),
    Entry    DATE,
    Position VARCHAR2(10),
    CONSTRAINT EmployeePk PRIMARY KEY (Employee)
  );

-- insert some values:

DELETE FROM Saleinfo;
INSERT INTO Saleinfo VALUES (1, 1,  3, 40.5);
INSERT INTO Saleinfo VALUES (1, 2,  2, 22.8);
INSERT INTO Saleinfo VALUES (1, 3, -1, 10.0);
INSERT INTO Saleinfo VALUES (2, 1,  5, 54.2);
INSERT INTO Saleinfo VALUES (2, 2,  7, 58.6);
INSERT INTO Saleinfo VALUES (2, 3,  4, 41.0);
INSERT INTO Saleinfo VALUES (3, 1, -1, 10.0);
INSERT INTO Saleinfo VALUES (3, 2,  2, 38.1);
INSERT INTO Saleinfo VALUES (3, 3,  4, 44.3);
```

```
DELETE FROM Employeeinfo;
INSERT INTO Employeeinfo VALUES (1, to_date('02-12-1988','DD-MM-YYYY'),
                                 'Senior');
INSERT INTO Employeeinfo VALUES (2, to_date('01-06-1998','DD-MM-YYYY'),
                                 'Trainee');
INSERT INTO Employeeinfo VALUES (3, to_date('01-01-1990','DD-MM-YYYY'),
                                 'Senior');

-- chain A:

-- step A1: select rows with Sales < 5

CREATE OR REPLACE VIEW Smallsales AS
    (SELECT * FROM Saleinfo WHERE Sales < 5);

-- step A2: map -1 to 0 for the Sales column

CREATE OR REPLACE VIEW Smallsales_Corrected AS
    (SELECT Employee,
            Month,
            (CASE WHEN Sales = -1 THEN 0 ELSE Sales END) AS Sales,
            Revenue
     FROM Smallsales
    );

-- step A3: discretise Revenue column into 2 bins

CREATE OR REPLACE VIEW Binned_Revenue AS
    (SELECT Employee,
            Month,
            (CASE WHEN Revenue < 40 THEN 0 ELSE 1 END) AS Bin
     FROM Smallsales_Corrected
    );

-- step A4: compute ratio of high revenue months per employee

CREATE OR REPLACE VIEW High_Revenues_Ratio AS
    (SELECT Employee,
            AVG(Bin) AS Ratio
     FROM Binned_Revenue
     GROUP BY Employee
    );
```

```
-- chain B:

-- step B1: pivotise revenues (create 3 new columns,
            one per month)

CREATE OR REPLACE VIEW Pivotised_Data AS
    (SELECT Employee,
            SUM(CASE WHEN Month = 1 THEN Revenue ELSE 0 END)
                AS Revenue_Month1,
            SUM(CASE WHEN Month = 2 THEN Revenue ELSE 0 END)
                AS Revenue_Month2,
            SUM(CASE WHEN Month = 3 THEN Revenue ELSE 0 END)
                AS Revenue_Month3
     FROM Saleinfo
     GROUP BY employee
    );

-- step B2: join

CREATE OR REPLACE VIEW Alldata AS
    (SELECT Employeeinfo.Employee,
            Revenue_Month1,
            Revenue_Month2,
            Revenue_Month3,
            Entry,
            Position
     FROM Pivotised_Data, Employeeinfo
     WHERE Pivotised_Data.Employee = Employeeinfo.Employee
    );

-- step B3: compute changes in the revenues per month

CREATE OR REPLACE VIEW Revenue_Changes AS
    (SELECT Employee,
            (Revenue_Month3 - Revenue_Month1) AS DiffM3M1,
            (Revenue_Month3 - Revenue_Month2) AS DiffM3M2,
            Revenue_Month1,
            Revenue_Month2,
            Revenue_Month3,
            Entry,
            Position
     FROM Alldata
    );
```

Bibliography

Aamodt, A., & Plaza, E. (1994). Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, *7*, 39–59.

Abbott, D. W., Matkovsky, I. P., & Elder IV, J. F. (1998). An Evaluation of High-end Data Mining Tools for Fraud Detection. *IEEE International Conference on Systems, Man, and Cybernetics*. San Diego, CA.

Abiteboul, S., & Vianu, V. (1991). Generic Computation and its Complexity. *Proceedings of the 23rd ACM Symposium on the Theory of Computing* (pp. 209–219).

Abrial, J.-R. (1974). Data Semantics. In J. Klimbie and K. Koffeman (Eds.), *Data Base Management*, 1–59. Amsterdam: North Holland.

Acharya, S., Gibbons, P. B., Poosala, V., & Ramaswamy, S. (1999). Join Synopses for Approximate Query Answering. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 275–286). New York, NY, USA: ACM Press.

Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining Association Rules between Sets of Items in Large Databases. *Proceedings of the ACM SIGMOD Conference on Management of Data* (pp. 207–216). Washington, D. C.

Aho, A. V., & Ullman, J. D. (1979). Universality of Data Retrieval Languages. *Proceedings of the 6th ACM Symposium on Principles of Programming Languages* (pp. 110–117). San Antonio, Texas.

Akahani, J., Hiramatsu, K., & Kogure, K. (2002). Coordinating Heterogeneous Information Services Based on Approximate Ontology Translation. *Proceedings of the AAMAS-2002 Workshop on Agentcities: Challenges in Open Agent Systems.*

AlSairafi, S., Emmanouil, F.-S., Ghanem, M., Giannadakis, N., Guo, Y., Kalaitzopoulos, D., Osmond, M., Rowe, A., Syed, J., & Wendel, P. (2003). The Design of Discovery Net: Towards Open Grid Services for Knowledge Discovery. *High-Performance Computing Applications*, *17*, 297–315.

Altenschmidt, C., & Biskup, J. (2002). Explicit Representation of Constrained Schema Mappings for Mediated Data Integration. *Proceedings of the Second International Workshop on Databases in Networked Information Systems (DNIS)* (pp. 103–132). London, UK: Springer.

April, A. A., & Al-Shurougi, D. (2000). Software Product Measurement for Supplier Evaluation. *Proceedings of the FESMA-AEMES Software Measurement Conference.* Madrid, Spain.

Aubrecht, P., Zelezny, F., Miksovsky, P., & Stepankova, O. (2002). SumatraTT: Towards a Universal Data Preprocessor. *Cybernetics and Systems* (pp. 818–823). Vienna: Austrian Society for Cybernetics Studies.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., & Patel-Schneider, P. (2003). *The Description Logic Handbook.* Cambridge (UK): Cambridge University Press.

Babcock, B., Babu, S., Datar, M., Motwani, R., & Widom, J. (2002). Models and Issues in Data Stream Systems. *Proceedings of 21st ACM Symposium on Principles of Database Systems (PODS)* (pp. 1–16).

Banerjee, J., Kim, W., Kim, H.-J., & Korth, H. F. (1987). Semantics and Implementation of Schema Evolution in Object-Oriented Databases. *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data* (pp. 311–322). New York, NY, USA: ACM Press.

Barbacci, M., Klein, M. H., Longstaff, T. A., & Weinstock, C. B. (1995). *Quality Attributes* (Technical Report CMU/SEI-95-TR-021). Software Engineering Institute.

Batini, C., Ceri, S., & Navathe, S. B. (1992). *Conceptual Database Design: An Entity-Relationship Approach.* Redwood City: Benjamin/Cummings.

Berka, P., Jirousek, R., & Pudil, P. (2002). *Feature Selection Operators Based on Information Theoretical Measures* (Technical Report Deliverable D14.4). IST Project MiningMart, IST-11993.

Bernstein, A., Hill, S., & Provost, F. (2005). Toward Intelligent Assistance for a Data Mining Process: An Ontology-Based Approach for Cost-Sensitive Classification. *IEEE Transactions on Knowledge and Data Engineering*, *17*, 503–518.

Biskup, J. (1995). *Grundlagen von Informationssystemen.* Vieweg.

Boehm, B. W., Brown, J., Kaspar, H., Lipow, M., McLeod, G., & Merritt, M. (1978). *Characteristics of Software Quality.* Amsterdam: North-Holland.

Bogorny, V., Engel, P. M., & Alvares, L. O. C. (2005). Towards the Reduction of Spatial Joins for Knowledge Discovery in Geographic Databases Using Geo-Ontologies and Spatial Integrity Constraints. *Proceedings of the Workshop on Knowledge Discovery and Ontologies (KDO) at the 9th European Conference on Principles and Practice in Knowledge Discovery in Databases (PKDD)* (pp. 51–58). Porto, Portugal.

Borgelt, C., & Berthold, M. R. (2002). Mining Molecular Fragments: Finding Relevant Substructures of Molecules. *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM)* (pp. 51–58). Washington, DC, USA: IEEE Computer Society.

Borgida, A., Lenzerini, M., & Rosati, R. (2003). Description Logics for Databases. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi and P. Patel-Schneider (Eds.), *The Description Logic Handbook*, chapter 16. Cambridge University Press.

Borgida, A., & Mylopoulos, J. (2004). Data Semantics Revisited. *Proceedings of the VLDB workshop on The Semantic Web and Databases (SWDB)*. Toronto: Springer.

Botella, P., Illa, X. B., Carvallo, J. P., Franch, X., & Quer, C. (2002). Using Quality Models for Assessing COTS Selection. *Anais do WER02 - Workshop em Engenharia de Requisitos* (pp. 263–277). Valencia, Spain.

Boulicaut, J.-F. (2004). Inductive Databases and Multiple Uses of Frequent Itemsets: the cInQ Approach. In R. Meo, P. L. Lanzi and M. Klemettinen (Eds.), *Database Support for Data Mining Applications (LNCS 2682)*. Springer.

Boulicaut, J.-F., Klemettinen, M., & Mannila, H. (1999). Modeling KDD Processes within the Inductive Database Framework. *Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery (DaWaK)* (pp. 293–302). London, UK: Springer-Verlag.

Brachman, R. J. (1979). On the Epistomological Status of Semantic Networks. In N. Findler (Ed.), *Associative Networks: Representation and Use of Knowledge by Computers*, 3–50. New York: Academic Press.

Brachman, R. J., & Anand, T. (1996). The Process of Knowledge Discovery in Databases: A Human-Centered Approach. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining*, AAAI Press Series in Computer Science. Cambridge Massachusetts, London England: A Bradford Book, The MIT Press.

Brachman, R. J., & Schmolze, J. (1985). An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, *9*, 171–216.

Brazdil, P., Soares, C., & da Costa, J. P. (2003). Ranking Learning Algorithms: Using IBL and Meta-Learning on Accuracy and Time Results. *Machine Learning*, *50*, 251–277.

Brezany, P., Hofer, J., Tjoa, A. M., & Wohrer, A. (2003). GridMiner: An Infrastructure for Data Mining on Computational Grids. *Proceedings of the APAC Conference and Exhibition on Advanced Computing, Grid Applications and eResearch*. Queensland.

Brezany, P., Janciak, I., Wohrer, A., & Tjoa, A. M. (2004). GridMiner: A Framework for Knowledge Discovery on the Grid—from a Vision to Design and Implementation. *Proceedings of the Cracow Grid Workshop*. Cracow, Poland.

Brisson, L., Collard, M., LeBrigand, K., & Barbry, P. (2004). KTA: A Framework for Integrating Expert Knowledge and Experiment Memory in Transcriptome Analysis. *Workshop on Knowledge Discovery and Ontologies at ECML/PKDD '04* (pp. 85–90). Pisa, Italy.

Brodie, M. L. (1984). On the Development of Data Models. In M. L. Brodie, J. Mylopoulos and J. W. Schmidt (Eds.), *On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages*, 19–48. New York: Springer.

Brown, A. W., & Wallnau, K. C. (1996). A Framework for Systematic Evaluation of Software Technologies. *IEEE Software*, *13*, 39–49.

Burges, C. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, *2*, 121–167.

Cabibbo, L., & Torlone, R. (1999). A Framework for the Investigation of Aggregate Functions in Database Queries. *Proceedings of the 7th International Conference on Database Theory (ICDT)* (pp. 383–397). Springer.

Calvanese, D., Giacomo, G. D., & Lenzerini, M. (2000). Answering Queries Using Views over Description Logics Knowledge Bases. *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI)* (pp. 386–391). AAAI Press / The MIT Press.

Cannataro, M., & Comito, C. (2003). A Data Mining Ontology for Grid Programming. *1st Workshop on Semantics in Peer-to-Peer and Grid Computing at the 12th Int. World Wide Web Conference*. Budapest, Hungary.

Cannataro, M., Congiusta, A., Mastroianni, C., Pugliese, A., Talia, D., & Trunfio, P. (2004). Grid-Based Data Mining and Knowledge Discovery. In N. Zhong and J. Liu (Eds.), *Intelligent Technologies for Information Analysis*. Springer.

Carreira, P., & Galhardas, H. (2004). Execution of Data Mappers. *Proceedings of the International Workshop on Information quality in Information Systems (IQIS)* (pp. 2–9). New York, NY, USA: ACM Press.

Cartwright, M., & Shepperd, M. (2000). An Empirical Investigation of an Object-Oriented Software System. *IEEE Transactions on Software Engineering*, *26*, 786–796.

Carvallo, J. P., Franch, X., Grau, G., & Quer, C. (2004a). On the Use of Quality Models for COTS Evaluation. *Proceedings of the International Workshop on Models and Processes for the Evaluation of COTS Components (MPEC)*. Edinburgh, UK.

Carvallo, J. P., Franch, X., Grau, G., & Quer, C. (2004b). QM: A Tool for Building Software Quality Models. *Proceedings of the 12th IEEE International Conference on Requirements Engineering (RE 2004)* (pp. 358–359). Kyoto, Japan: IEEE Computer Society.

Cattell, R. G. G., Barry, D. K., Berler, M., Eastman, J., Jordan, D., Russell, C., Schadow, O., Stanienda, T., & Velez, F. (2000). *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann.

Cavano, J. P., & McCall, J. A. (1978). A Framework for the Measurement of Software Quality. *Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues* (pp. 133–139).

Cespivova, H., Rauch, J., Svatek, V., Kejkula, M., & Tomeckova, M. (2004). Roles of Medical Ontology in Association Mining CRISP-DM Cycle. *Workshop on Knowledge Discovery and Ontologies at ECML/PKDD.*

Chandra, A. K., & Harel, D. (1982). Structure and Complexity of Relational Queries. *Journal of Computer and System Sciences*, *25*, 99–128.

Chandra, A. K., & Harel, D. (1985). Horn Clause Queries and Generalizations. *Journal of Logic Programming*, *2*, 1–15.

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., & Wirth, R. (2000). *CRISP–DM 1.0* (Technical Report). The CRISP–DM Consortium.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-Sampling Technique. *Journal of Artificial Intelligence Research (JAIR)*, *16*, 321–357.

Chen, C. M., & Roussopoulos, N. (1994). Adaptive Selectivity Estimation Using Query Feedback. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 161–172). New York, NY, USA: ACM Press.

Chen, P. P. (1976). The Entity Relationship Model: Towards an Integrated View of Data. *ACM Transactions on Database Systems*, *1*, 9–36.

Christodoulakis, S. (1983). Estimating Block Transfers and Join Sizes. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 40–54). New York, NY, USA: ACM Press.

Chudzian, C., Granat, J., & Traczyk, W. (2003). *Call Center Case* (Technical Report Deliverable D17.2b). IST Project MiningMart, IST-11993.

Clancey, W. J. (1983). The Epistomology of a Rule-Based Expert System – a Framework for Explanation. *Artificial Intelligence*, *20*, 215–251.

Claypool, K. T., Jin, J., & Rundensteiner, E. A. (1998). SERF: Schema Evalution through an Extensible, Re-usable and Flexible Framework. *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)* (pp. 314–321).

Clear, J., Dunn, D., Harvey, B., Heytens, M. L., Lohman, P., Mehta, A., Melton, M., Rohrberg, L., Savasere, A., Wehrmeister, R. M., & Xu, M. (1999). NonStop SQL/MX Primitives for Knowledge Discovery. *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)* (pp. 425–429). San Diego, CA, USA.

Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, *13*, 377–387.

Collier, K. W., Sautter, D., Marjaniemi, C., & Carey, B. (1999). A Methodology for Evaluating and Selecting Data Mining Software. *Proceedings of the 32nd Hawaii Int. Conference on System Sciences.*

Colombo, R., & Guerra, A. (2002). The Evaluation Method for Software Products. *Proceedings of the International Conference on Software and Systems Engineering and Their Applications (ICSSEA).* Paris, France.

Coppock, D. S. (2003). Data Mining and Modeling: So You Have a Model, Now What? *DM Review Magazine.*

Cortes, C., & Vapnik, V. N. (1995). Support–Vector Networks. *Machine Learning Journal*, *20*, 273–297.

Cunningham, C., Graefe, G., & Galindo-Legaria, C. A. (2004). PIVOT and UNPIVOT: Optimization and Execution Strategies in an RDBMS. *Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB)* (pp. 998–1009). Morgan Kaufmann.

Davidson, S. B., & Kosky, A. (1997). WOL: A Language for Database Transformations and Constraints. *Proceedings of the Thirteenth International Conference on Data Engineering (ICDE)* (pp. 55–65). Washington, DC, USA: IEEE Computer Society.

Doan, A., Domingos, P., & Halevy, A. Y. (2001). Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 509–520). New York, NY, USA: ACM Press.

Domingos, P., & Hulten, G. (2000). Mining High Speed Data Streams. *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00)* (pp. 71–80).

Domingues, M. A., & Rezende, S. O. (2005). Using Taxonomies to Facilitate the Analysis of the Association Rules. *Proceedings of the Workshop on Knowledge Discovery and Ontologies (KDO) at the 9th European Conference on Principles and Practice in Knowledge Discovery in Databases (PKDD)* (pp. 59–66). Porto, Portugal.

Duschka, O. M., Genesereth, M. R., & Levy, A. Y. (2000). Recursive Query Plans for Data Integration. *Journal of Logic Programming*, *43*, 49–73.

Euler, T. (2002a). *Feature Selection with Support Vector Machines* (Technical Report Deliverable D14.3). IST Project MiningMart, IST-11993.

Euler, T. (2002b). *How to Implement M4 Operators* (Technical Report TR12-04). IST Project MiningMart, IST-11993.

Euler, T. (2002c). *Operator Specifications* (Technical Report TR12-02). IST Project MiningMart, IST-11993.

Euler, T. (2002d). Tailoring Text Using Topic Words: Selection and Compression. *Proceedings of the 13th International Workshop on Database and Expert Systems Applications (DEXA)* (pp. 215–219). Los Alamitos, CA: IEEE Computer Society Press.

Euler, T. (2005a). An Adaptable Software Product Evaluation Metric. *Proceedings of the 9th IASTED International Conference on Software Engineering and Applications (SEA)*. Phoenix, Arizona, USA.

Euler, T. (2005b). Churn Prediction in Telecommunications Using MiningMart. *Proceedings of the Workshop on Data Mining and Business (DMBiz) at the 9th European Conference on Principles and Practice in Knowledge Discovery in Databases (PKDD)*. Porto, Portugal.

Euler, T. (2005c). Modelling Data Mining Processes on a Conceptual Level. *Proceedings of the 5th International Conference on Decision Support for Telecommunications and Information Society*. Warsaw, Poland.

Euler, T. (2005d). Publishing Operational Models of Data Mining Case Studies. *Proceedings of the Workshop on Data Mining Case Studies at the 5th IEEE International Conference on Data Mining (ICDM)* (pp. 99–106). Houston, Texas, USA.

Euler, T. (2006). Modeling Preparation for Data Mining Processes. *Journal of Telecommunications and Information Technology*, 81–87.

Euler, T., Morik, K., & Scholz, M. (2003). MiningMart: Sharing Successful KDD Processes. *LLWA 2003 – Tagungsband der GI-Workshop-Woche Lehren – Lernen – Wissen – Adaptivitat* (pp. 121–122).

Euler, T., & Scholz, M. (2004). Using Ontologies in a KDD Workbench. *Workshop on Knowledge Discovery and Ontologies at ECML/PKDD '04* (pp. 103–108). Pisa, Italy.

Euzenat, J., & Valtchev, P. (2004). Similarity-Based Ontology Alignment in OWL-Lite. *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)* (pp. 333–337).

Fahrner, C. (1996). *Schematransformationen in Datenbanken*. Doctoral dissertation, Westfalische Wilhelms-Universitat Munster.

Famili, F., Shen, W.-M., Weber, R., & Simoudis, E. (1997). Data Preprocessing and Intelligent Data Analysis. *Intelligent Data Analysis*, *1*.

Fayyad, U. M., Piatetsky-Shapiro, G., & Smyth, P. (1996). From Data Mining to Knowledge Discovery: An Overview. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining*, chapter 1, 1–34. AAAI/MIT Press.

Fiedler, G., Raak, T., & Thalheim, B. (2005). Database Collaboration Instead of Integration. *Proceedings of the 2nd Asia-Pacific Conference on Conceptual Modelling (APCCM)* (pp. 49–58). Darlinghurst, Australia: Australian Computer Society, Inc.

Formica, A., & Missikoff, M. (2004). Inheritance Processing and Conflicts in Structural Generalization Hierarchies. *ACM Computing Surveys*, *36*, 263–290.

Fortin, S. (1996). *The Graph Isomorphism Problem* (Technical Report 96-20). University of Alberta, Canada.

Franconi, E., & Ng, G. (2000). The i.com Tool for Intelligent Conceptual Modelling. *Proceedings of the 7th International Workshop on Knowledge Representation meets Databases (KRDB)* (pp. 45–53). CEUR-WS.org.

Franconi, E., & Sattler, U. (1999). A Data Warehouse Conceptual Data Model for Multidimensional Aggregation. *Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW)* (p. 13). CEUR-WS.org.

Freeman, E., & Melli, G. (2005). Championing LTV at LTC. *Proceedings of the Workshop on Data Mining Case Studies at the 5th IEEE International Conference on Data Mining (ICDM)* (pp. 107–113). Houston, Texas, USA.

Freitas, A. A., & Lavington, S. H. (1996). Using SQL Primitives and Parallel DB Servers to Speed up Knowledge Discovery in Large Relational Databases. *Cybernetics and Systems: Proceedings of the 13th European Meeting on Cybernetics and Systems Research (EMCSR)* (pp. 955–960). Vienna, Austria: Austrian Society for Cybernetic Studies.

Galhardas, H., Florescu, D., Shasha, D., Simon, E., & Saita, C.-A. (2001). Declarative Data Cleaning: Language, Model, and Algorithms. *Proceedings of 27th International Conference on Very Large Data Bases (VLDB)* (pp. 371–380). Morgan Kaufmann.

Gamma, E., Helm, R., Johnson, R. E., & Vlissides, J. (1995). *Design Patterns. Elements of Reusable Object-Oriented Software.* Amsterdam: Addison-Wesley Longman.

Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., Vassalos, V., & Widom, J. (1997). The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, *8*, 117–132.

Garcia-Molina, H., Ullman, J. D., & Widom, J. (2002). *Database Systems: The Complete Book.* Upper Saddle River, New Jersey: Prentice Hall.

Gaul, W., & Sauberlich, F. (1999). Classification and Positioning of Data Mining Tools. *Classification in the Information Age* (pp. 145–154). Springer.

Gentsch, P., Niemann, C., & Roth, M. (2000). *Data Mining: 12 Software-Losungen im Vergleich.* Oxygon-Verlag. In German.

Gimbel, M., Klein, M., & Lockemann, P. C. (2004). Interactivity, Scalability and Resource Control for Efficient KDD Support in DBMS. In R. Meo, P. L. Lanzi and M. Klemettinen (Eds.), *Database Support for Data Mining Applications (LNAI 2682)*, 174–193. Berlin, Heidelberg: Springer.

Giraud-Carrier, C., & Provost, F. (2005). Toward a Justification of Meta-Learning: Is the No Free Lunch Theorem a Show-Stopper? *Proceedings of the Workshop on Meta-Learning at the International Conference on Machine Learning (ICML)* (pp. 12–19).

Giraud-Carrier, C. G., & Povel, O. (2003). Characterising Data Mining Software. *Intelligent Data Analysis*, *7*, 181–192.

Goebel, M., & Gruenwald, L. (1999). A Survey of Data Mining and Knowledge Discovery Software Tools. *ACM SIGKDD Explorations*, *1*, 20–33.

Grossman, R. L., Hornick, M. F., & Meyer, G. (2002). Data Mining Standards Initiatives. *Communications of the ACM*, *45*, 59–61.

Gruber, T. R. (1993). Towards Principles for the Design of Ontologies Used for Knowledge Sharing. *Formal Ontology in Conceptual Analysis and Knowledge Representation*. Deventer, The Netherlands: Kluwer Academic Publishers.

Gupta, H. (1997). Selection of Views to Materialize in a Data Warehouse. *Proceedings of the 6th International Conference on Database Theory (ICDT)* (pp. 98–112). London, UK: Springer-Verlag.

Gupta, H., & Mumick, I. S. (2005). Selection of Views to Materialize in a Data Warehouse. *IEEE Transactions on Knowledge and Data Engineering*, *17*, 24–43.

Gyssens, M., Lakshmanan, L. V. S., & Subramanian, I. N. (1996). Tables as a Paradigm for Querying and Restructuring. *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)* (pp. 93–103). New York, NY, USA: ACM Press.

Haas, P., Kandil, M., Lerner, A., Markl, V., Popivanov, I., Raman, V., & Zilio, D. (2005). Automated Statistics Collection in Action. *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data* (pp. 933–935). New York, NY, USA: ACM Press.

Haas, P. J., Naughton, J. F., Seshadri, S., & Swami, A. N. (1996). Selectivity and Cost Estimation for Joins Based on Random Sampling. *Journal of Computer and System Sciences*, *52*, 550–569.

Halevy, A. Y. (2001). Answering Queries Using Views: A Survey. *VLDB Journal*, *10*, 270–294.

Hammer, J., Garcia-Molina, H., Nestorov, S., Yerneni, R., Breunig, M., & Vassalos, V. (1997). Template-Based Wrappers in the TSIMMIS System. *Proceedings of the ACM SIGMOD Conference on Management of Data* (pp. 532–535). Tucson, Arizona.

Han, J., & Fu, Y. (1999). Mining Multiple-Level Association Rules in Large Databases. *IEEE Transactions of Knowledge and Data Engineering*, *11*, 798–804.

Han, J., Fu, Y., Wang, W., Koperski, K., & Zaiane, O. (1996). DMQL: A Data Mining Query Language for Relational Databases. *Proceedings of the SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD)*. Montreal, Canada.

Harangsri, B., Shepherd, J., & Ngu, A. H. H. (1997). Query Size Estimation Using Machine Learning. *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications (DASFAA)* (pp. 97–106). World Scientific.

Harinarayan, V., Rajaraman, A., & Ullman, J. D. (1996). Implementing Data Cubes Efficiently. *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* (pp. 205–216). New York, NY, USA: ACM Press.

Haustein, S. (2002). *Internet Presentation of MiningMart Cases* (Technical Report Deliverable D9). IST Project MiningMart, IST-11993.

Haustein, S. (2006). *An interpretative approach to the model-driven development of web applications.* Doctoral dissertation, University Dortmund.

Haustein, S., & Pleumann, J. (2002). Easing participation in the semantic web. *International Workshop on the Semantic Web at WWW2002.*

Hereth, J., & Stumme, G. (2001). Reverse Pivoting in Conceptual Information Systems. *Proceedings of the 9th International Conference on Conceptual Structures (ICCS)* (pp. 202–215). London, UK: Springer-Verlag.

Hermiz, K. B. (1999). Critical Success Factors for Data Mining Projects. *DM Review Magazine.*

Holsapple, C. W. (2003). *Handbook on Knowledge Management.* Springer.

Holsapple, C. W., & Joshi, K. D. (2003). A Knowledge Management Ontology. In C. W. Holsapple (Ed.), *Handbook on Knowledge Management*, 89–124. Springer.

Hornick, M. F., Yoon, H., & Venkayala, S. (2004). Java Data Mining (JSR-73): Status and Overview. *Proceedings of the Workshop on Data Mining Standards, Services and Platforms at the 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)* (pp. 23–29).

Hull, R., & King, R. (1987). Semantic Database Modeling: Survey, Applications, and Research Issues. *ACM Computing Surveys*, *19*, 202–260.

Imielinski, T., & Virmani, A. (1999). MSQL: A Query Language for Database Mining. *Journal of Data Mining and Knowledge Discovery*, *3*, 373–408.

Inmon, W. H. (1996). *Building the Data Warehouse.* New York: J. Wiley & Sons. 2 edition.

Inokuchi, A., Washio, T., & Motoda, H. (2000). An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data. *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)* (pp. 13–23). London, UK: Springer-Verlag.

Joachims, T. (2000). Estimating the Generalization Performance of a SVM Efficiently. *Proceedings of the International Conference on Machine Learning (ICML)* (pp. 431–438). San Francisco, CA, USA: Morgan Kaufman.

Joachims, T. (2001). *The Maximum-Margin Approach to Learning Text Classifiers: Methods, Theory, and Algorithms.* Doctoral dissertation, Fachbereich Informatik, Universität Dortmund.

John, G. H., & Lent, B. (1997). SIPping from the Data Firehose. *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD)* (pp. 199–202). AAAI Press.

Johnson, T., Lakshmanan, L. V. S., & Ng, R. T. (2000). The 3W Model and Algebra for Unified Data Mining. *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB)* (pp. 21–32). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Kalfoglou, Y., & Schorlemmer, M. (2003). Ontology mapping: The state of the art. *The Knowledge Engineering Review*, *18*, 1–31.

Kerber, R., Beck, H., Anand, T., & Smart, B. (1998). Active Templates: Comprehensive Support for the Knowledge Discovery Process. *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining.* New York.

Kietz, J.-U., Vaduva, A., & Zucker, R. (2000). Mining Mart: Combining Case-Based-Reasoning and Multi-Strategy Learning into a Framework to reuse KDD-Application. *Proceedings of the fifth International Workshop on Multistrategy Learning (MSL2000).* Guimares, Portugal.

Kietz, J.-U., Vaduva, A., & Zucker, R. (2001). MiningMart: Metadata-Driven Preprocessing. *Proceedings of the ECML/PKDD Workshop on Database Support for KDD.*

Kitts, B., Melli, G., & Rexer, K. (Eds.). (2005). *Proceedings of the First International Workshop on Data Mining Case Studies at the IEEE International Conference on Data Mining (ICDM).* Houston, Texas, USA.

Klinkenberg, R. (2004). Learning Drifting Concepts: Example Selection vs. Example Weighting. *Intelligent Data Analysis (IDA), Special Issue on Incremental Learning Systems Capable of Dealing with Concept Drift*, *8*, 281–300.

Klosgen, W. (2000). Subgroup Patterns. In W. Klosgen and J. Zytkow (Eds.), *Handbook of Knowledge Discovery and Data Mining.* London: Oxford University Press.

Knobbe, A. (2004). *Multi-Relational Data Mining.* Doctoral dissertation, Universiteit Utrecht.

Knobbe, A., Schipper, A., & Brockhausen, P. (2000). *Domain Knowledge and Data Mining Process Decisions* (Technical Report Deliverable D5). IST Project MiningMart, IST-11993.

Knobbe, A. J., de Haas, M., & Siebes, A. (2001). Propositionalisation and Aggregates. *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)* (pp. 277–288). London, UK: Springer.

Kohavi, R., Mason, L., Parekh, R., & Zheng, Z. (2004). Lessons and Challenges from Mining Retail E-Commerce Data. *Machine Learning*, *57*, 83–113.

Kolaitis, P. G., & Vardi, M. Y. (1995). On the Expressive Power of Datalog: Tools and a Case Study. *Journal of Computer and System Sciences*, *51*, 110–134.

Kosala, R., & Blockeel, H. (2000). Web Mining Research: A Survey. *ACM SIGKDD Explorations Newsletter*, *2*.

Kramer, S., Aufschild, V., Hapfelmeier, A., Jarasch, A., Kessler, K., Reckow, S., Wicker, J., & Richter, L. (2005). Inductive Databases in the Relational Model: The Data as the Bridge. *Proceedings of the 4th International Workshop on Knowledge Discovery in Inductive Databases (KDID)* (pp. 124–138). Porto, Portugal: Springer.

Krogel, M.-A., Rawles, S., Zelezny, F., Flach, P. A., Lavrac, N., & Wrobel, S. (2003). Comparative Evaluation of Approaches to Propositionalization. *Proceedings of the Thirteenth International Conference on Inductive Logic Programming* (pp. 197–214). Springer.

Krogel, M.-A., & Wrobel, S. (2001). Transformation-Based Learning Using Multirelational Aggregation. *Proceedings of the 11th International Conference on Inductive Logic Programming (ILP)* (pp. 142–155). Springer.

Kuramochi, M., & Karypis, G. (2001). Frequent Subgraph Discovery. *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM)* (pp. 313–320). Washington, DC, USA: IEEE Computer Society.

Kusters, R., van Solingen, R., & Trienekens, J. (1997). User-Perceptions of Embedded Software Quality. *Proceedings of the STEP97 Conference*. IEEE Computer Society Press.

Lakshmanan, L. V. S., Sadri, F., & Subramanian, I. N. (1996). SchemaSQL - A Language for Interoperability in Relational Multi-Database Systems. *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB)* (pp. 239–250). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Lakshmanan, L. V. S., Sadri, F., & Subramanian, S. N. (2001). SchemaSQL: An Extension to SQL for Multidatabase Interoperability. *ACM Transactions on Database Systems*, *26*, 476–519.

Langley, P., & Simon, H. A. (1995). Applications of Machine Learning and Rule Induction. *Communications of the ACM*, *38*, 55–64.

Lenzerini, M. (2002). Data Integration: A Theoretical Perspective. *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)* (pp. 233–246). New York, NY, USA: ACM Press.

Libkin, L. (2003). Expressive Power of SQL. *Journal of Theoretical Computer Science*, *296*, 379–404.

Ling, C. X., Sheng, S., Bruckhaus, T., & Madhavji, N. H. (2005). Predicting Software Escalations with Maximum ROI. *Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM)* (pp. 717–720). Los Alamitos, CA, USA: IEEE Computer Society.

Litvak, M., Last, M., & Kisilevich, S. (2005). Improving Classification of Multi-Lingual Web Documents using Domain Ontologies. *Proceedings of the Workshop on Knowledge Discovery and Ontologies (KDO) at the 9th European Conference on Principles and Practice in Knowledge Discovery in Databases (PKDD)* (pp. 67–74). Porto, Portugal.

Liu, H., & Motoda, H. (1998). *Feature Extraction, Construction, and Selection: A Data Mining Perspective*. Kluwer.

Loureiro, A., Torgo, L., & Soares, C. (2005). Outlier Detection Using Clustering Methods: A Data Cleaning Application. *Proceedings of the Workshop on Data Mining and Business (DMBiz) at the 9th European Conference on Principles and Practice in Knowledge Discovery in Databases (PKDD)*. Porto, Portugal.

Madhavan, J., Bernstein, P. A., & Rahm, E. (2001). Generic Schema Matching with Cupid. *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB)* (pp. 49–58). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Maiden, N. A., Ncube, C., & Moore, A. (1997). Lessons Learned During Requirements Acquisition for COTS Systems. *Communications of the ACM*, *40*, 21–25.

Maier, T., & Reinartz, T. (2004). Evaluation of Web Usage Analysis Tools. *Kunstliche Intelligenz*, *1*, 65–68.

Mannino, M. V., Chu, P., & Sager, T. (1988). Statistical Profile Estimation in Database Systems. *ACM Computing Surveys*, *20*, 191–221.

Masolo, C., Borgo, S., Gangemi, A., Guarino, N., & Oltramari, A. (2003). *Ontology Library* (Technical Report Deliverable D18). WonderWeb Project.

Matheus, C. J., Chan, P. K., & Piatetsky-Shapiro, G. (1993). Systems for Knowledge Discovery in Databases. *IEEE Transactions on Knowledge and Data Engineering*, *5*, 903–913.

Mayrand, J., & Coallier, F. (1996). System Acquisition Based On Software Product Assessment. *Proceedings of the International Conference on Software Engineering (ICSE)*.

Melnik, S., Bernstein, P. A., Halevy, A., & Rahm, E. (2005). Supporting Executable Mappings in Model Management. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 167–178). New York, NY, USA: ACM Press.

Melnik, S., Garcia-Molina, H., & Rahm, E. (2002). Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. *Proceedings of the 18th International Conference on Data Engineering (ICDE)* (pp. 117–129). Washington, DC, USA: IEEE Computer Society.

Mena, E., Illarramendi, A., Kashyap, V., & Sheth, A. P. (2000). OBSERVER: An Approach for Query Processing in Global Information Systems Based on Interoperation Across Pre-Existing Ontologies. *Distributed and Parallel Databases*, *8*, 223–271.

Meo, R., & Psaila, G. (2003). *An XML-Based Definition of a Database for Knowledge Discovery* (Technical Report RT74-2003). Dipartimento di Informatica, Universita di Torino.

Meo, R., Psaila, G., & Ceri, S. (1998). An Extension to SQL for Mining Association Rules. *Journal of Data Mining and Knowledge Discovery*, *2*, 194–224.

Meyer, D., & Cannon, C. (1998). *Building a Better Data Warehouse.* Prentice Hall.

Michie, D., Spiegelhalter, D. J., & Taylor, C. C. (1994). *Machine Learning, Neural and Statistical Classification.* New York u.a.: Ellis Horwood.

Mierswa, I., Klinkenberg, R., Fischer, S., & Ritthoff, O. (2003). A Flexible Platform for Knowledge Discovery Experiments: YALE – Yet Another Learning Environment. *LLWA 03 - Tagungsband der GI-Workshop-Woche Lernen - Lehren - Wissen - Adaptivitat.*

Mierswa, I., & Morik, K. (2005). Automatic Feature Extraction for Classifying Audio Data. *Machine Learning Journal*, *58*, 127–149.

Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., & Euler, T. (2006). YALE: Rapid Prototyping for Complex Data Mining Tasks. *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006).* ACM Press.

Mistry, H., Roy, P., Sudarshan, S., & Ramamritham, K. (2001). Materialized View Selection and Maintenance Using Multi-Query Optimization. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 307–318). New York, NY, USA: ACM Press.

Mitchell, T. M. (1997). *Machine Learning.* New York: McGraw Hill.

Morik, K. (2000). The Representation Race - Preprocessing for Handling Time Phenomena. *Proceedings of the European Conference on Machine Learning 2000 (ECML 2000).* Berlin, Heidelberg, New York: Springer Verlag Berlin.

Morik, K., Botta, M., Dittrich, K. R., Kietz, J.-U., Portinale, L., Vaduva, A., & Zücker, R. (2001). *M4 – The MiningMart Meta Model* (Technical Report Deliverable D8/9). IST Project MiningMart, IST-11993.

Morik, K., Boulicaut, J.-F., & Siebes, A. (2005). *Local Pattern Discovery*, vol. 3539 of *Lecture Notes in Computer Science.* Springer.

Morik, K., Causse, K., & Boswell, R. (1991). A common knowledge representation integrating learning tools. *Proc. of the 1st International Workshop on Multistrategy Learning.* Harpers Ferry.

Morik, K., & Köpcke, H. (2005). Features for Learning Local Patterns in Time-Stamped Data. In K. Morik, J.-F. Boulicaut and A. Siebes (Eds.), *Local pattern detection: International seminar, dagstuhl castle, germany, april 12-16, 2004, revised selected papers*, chapter 7, 98–114. Springer.

Morik, K., & Scholz, M. (2004). The MiningMart Approach to Knowledge Discovery in Databases. In N. Zhong and J. Liu (Eds.), *Intelligent Technologies for Information Analysis*, chapter 3, 47–65. Springer.

Muggleton, S. (1995). Inverting Entailment and Progol. *Machine Intelligence 14*, 133 – 187.

Münstermann, D. (2002). Wissensentdeckung in Datenbanken mit dynamischer Anpassung des Hypothesentests. Master's thesis, Fachbereich Informatik, Universität Dortmund. In German.

Musick, R., & Critchlow, T. (1999). Practical Lessons in Supporting Large-scale Computational Science. *ACM SIGMOD Record*, *28*, 49–57.

Newell, A. (1982). The Knowledge Level. *Artificial Intelligence*, *18*, 87–127.

Ngu, A. H. H., Harangsri, B., & Shepherd, J. (2004). Query Size Estimation for Joins Using Systematic Sampling. *Distributed and Parallel Databases*, *15*, 237–275.

Nijssen, G. (1977). Current Issues in Conceptual Schema Concepts. In G. Nijssen (Ed.), *Architecture and Models in Data Base Management Systems*, 31–66. North-Holland.

Niles, I., & Pease, A. (2001). Towards a Standard Upper Ontology. *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS)* (pp. 2–9). New York, NY, USA: ACM Press.

Papakonstantinou, Y., Garcia-Molina, H., & Widom, J. (1995). Object Exchange Across Heterogeneous Information Sources. *Proceedings of the Eleventh International Conference on Data Engineering (ICDE)* (pp. 251–260). Washington, DC, USA: IEEE Computer Society.

Park, B.-H., & Kargupta, H. (2002). Distributed Data Mining: Algorithms, Systems, and Applications. In N. Ye (Ed.), *Data Mining Handbook*, 341–358. IEA.

Paulk, M. C., Weber, C. V., Curtis, B., & Chrissis, M. B. (1995). *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley.

Pechenizkiy, M., Puuronen, S., & Tsymbal, A. (2005). Competitive Advantage from Data Mining: Some Lessons Learned in the Information Systems Field. *IEEE Workshop Proc. of DEXA05, 1st Int. Workshop on Philosophies and Methodologies for Knowledge Discovery (PMKD)* (pp. 733–737). IEEE CS Press.

Peckham, J., & Maryanski, F. (1988). Semantic Data Models. *ACM Computing Surveys*, *20*, 153–189.

Perlich, C., & Provost, F. (2003). Aggregation-Based Feature Invention and Relational Concept Classes. *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)* (pp. 167–176). New York, NY, USA: ACM Press.

Pfahringer, B., Bensusan, H., & Giraud-Carrier, C. G. (2000). Meta-Learning by Landmarking Various Learning Algorithms. *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)* (pp. 743–750). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Pleumann, J. (2007). *Ein Ansatz zur Entwicklung von Modellierungswerkzeugen für die softwaretechnische Lehre.* Doctoral dissertation, Fachbereich Informatik, Universität Dortmund.

Poosala, V., Haas, P. J., Ioannidis, Y. E., & Shekita, E. J. (1996). Improved Histograms for Selectivity Estimation of Range Predicates. *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* (pp. 294–305). New York, NY, USA: ACM Press.

Probst, G., Raub, S., & Romhardt, K. (1999). *Managing Knowledge.* Berlin Heidelberg: Springer.

Punter, T. (1997). Using Checklists to Evaluate Software Product Quality. *Proceedings of the 8th European Software Control and Metrics Conference (ESCOM).* Berlin, Germany.

Punter, T., Kusters, R., Trienekens, J., Bemelmans, T., & Brombacher, A. (2004). The W-Process for Software Product Evaluation: A Method for Goal-Oriented Implementation of the ISO 14598 Standard. *Software Quality Journal*, *12*, 137–158.

Punter, T., van Solingen, R., & Trienekens, J. (1997). Software Product Evaluation. *Proceedings of the 4th Conference on Evaluation of Information Technology (EVIT).* Delft, The Netherlands.

Pyle, D. (1999). *Data Preparation for Data Mining.* Morgan Kaufmann Publishers.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* Machine Learning. San Mateo, CA: Morgan Kaufmann.

Rahm, E., & Bernstein, P. A. (2001). A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal*, *10*, 334–350.

Ramakrishnan, R., Agrawal, R., Freytag, J.-C., Bollinger, T., Clifton, C. W., Dzeroski, S., Hipp, J., Keim, D., Kramer, S., Kriegel, H.-P., Leser, U., Liu, B., Mannila, H., Meo, R., Morishita, S., Ng, R., Pei, J., Raghavan, P., Spiliopoulou, M., Srivastava, J., & Torra, V. (2005). Data Mining: The Next Generation. *Perspectives Workshop: Data Mining: The Next Generation.* Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany.

Raman, V., & Hellerstein, J. M. (2000). *An Interactive Framework for Data Cleaning* (Technical Report UCB/CSD-00-1110). Computer Science Division (EECS), University of California at Berkeley.

Raman, V., & Hellerstein, J. M. (2001). Potter's Wheel: An Interactive Data Cleaning System. *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB)* (pp. 381–390). Morgan Kaufmann.

Rangarajan, K., Swaminathan, N., Hedge, V., & Jacob, J. (2001). Product Quality Framework: A Vehicle for Focusing on Product Quality Goals. *Software Engineering Notes*, *26*, 77–82.

Raspl, S. (2004). PMML Version 3.0 – Overview and Status. *Proceedings of the Workshop on Data Mining Standards, Services and Platforms at the 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)* (pp. 18–22).

Rem, O., & Trautwein, M. (2002). *Best Practices Report* (Technical Report Deliverable D11.3). IST Project MiningMart, IST-11993.

Richeldi, M., & Perrucci, A. (2002a). *Churn Analysis Case Study* (Technical Report Deliverable D17.2). IST Project MiningMart, IST-11993.

Richeldi, M., & Perrucci, A. (2002b). *MiningMart Evaluation Report* (Technical Report Deliverable D17.3). IST Project MiningMart, IST-11993.

Riedemann, E. H. (1997). *Testmethoden fur sequentielle und nebenlaufige Software-Systeme*. Stuttgart: Teubner.

Roddick, J. F., Al-Jadir, L., Bertossi, L. E., Dumas, M., Estrella, F., Gregersen, H., Hornsby, K., Lufter, J., Mandreoli, F., Mannisto, T., Mayol, E., & Wedemeijer, L. (2000). Evolution and Change in Data Management – Issues and Directions. *SIGMOD Record*, *29*, 21–25.

Romei, A., Ruggieri, S., & Turini, F. (2005). KDDML: A Middleware Language and System for Knowledge Discovery in Databases. *Proceedings of the 13th Italian Symposium on Advanced Database Systems (SEBD)*.

Romei, A., Ruggieri, S., & Turini, F. (2006). KDDML: A Middleware Language and System for Knowledge Discovery in Databases. *Data and Knowledge Engineering*, *57*, 179–220.

Ross, K. A., Srivastava, D., & Sudarshan, S. (1996). Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time. *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* (pp. 447–458). New York, NY, USA: ACM Press.

Rüping, S. (1999). Zeitreihenprognose für Warenwirtschaftssysteme unter Berücksichtigung asymmetrischer Kostenfunktionen. Master's thesis, Universität Dortmund, Fachbereich Informatik. In German.

Rüping, S. (2002). Support Vector Machines in Relational Databases. *Pattern Recognition with Support Vector Machines – First International Workshop (SVM)* (pp. 310–320). Springer.

Salton, G., & Buckley, C. (1988). Term Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management*, *24*, 513–523.

Sarawagi, S., Thomas, S., & Agrawal, R. (1998). Integrating Association Rule Mining with relational Database Systems: Alternatives and Implications. *Proceedings of the ACM SIGMOD, International Conference on Management of Data* (pp. 343–354).

Sattler, K.-U., & Schallehn, E. (2001). A Data Preparation Framework Based on a Multidatabase Language. *Proceedings of the International Database Engineering & Applications Symposium (IDEAS)* (pp. 219–228). Grenoble, France: IEEE Computer Society.

Schallehn, E., Sattler, K.-U., & Saake, G. (2001). Advanced Grouping and Aggregation for Data Integration. *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM)* (pp. 547–549). New York, NY, USA: ACM Press.

Scholz, M. (2002). *Representing Constraints, Conditions and Assertions in M4* (Technical Report TR18-01). IST Project MiningMart, IST-11993.

Scholz, M. (2005). Knowledge-Based Sampling for Subgroup Discovery. In K. Morik, J.-F. Boulicaut and A. Siebes (Eds.), *Local pattern detection*, vol. LNAI 3539 of *Lecture Notes in Artificial Intelligence*, 171–189. Springer.

Scholz, M. (2007). *Scalable and Accurate Knowledge Discovery in Real-World Databases.* Doctoral dissertation, Fachbereich Informatik, Universität Dortmund.

Scholz, M., & Euler, T. (2002). *Documentation of the MiningMart Meta Model (M4)* (Technical Report TR12-05). IST Project MiningMart, IST-11993.

Schreiber, G., Wielinga, B., & Akkermans, H. (1993a). Using KADS to Analyse Problem-Solving Methods. In G. Schreiber, B. Wielinga and J. Breucker (Eds.), *KADS – A Principled Approach to Knowledge-Based System Development*, vol. 11 of *Knowledge Based Systems*, 415–430. London: Academic Press.

Schreiber, G., Wielinga, B., & Breucker, J. (1993b). Introduction and Overview. In G. Schreiber, B. Wielinga and J. Breucker (Eds.), *KADS – A Principled Approach to Knowledge-Based System Development*, vol. 11 of *Knowledge Based Systems*, 1–18. London: Academic Press.

Schreiber, G., Wielinga, B., & Breucker, J. (1993c). *KADS – A Principled Approach to Knowledge-Based System Development*, vol. 11 of *Knowledge Based Systems.* London: Academic Press.

Sellis, T. K. (1988). Multiple-Query Optimization. *ACM Transactions on Database Systems*, *13*, 23–52.

Shah, G., & Syeda-Mahmood, T. (2004). Searching Databases for Semantically-Related Schemas. *Proceedings of the 27th Annual International ACM Conference on Research and Development in Information Retrieval (SIGIR)* (pp. 504–505). New York, NY, USA: ACM Press.

Singh, P., Choudur, L., Benson, A., & Mathew, M. (2005). External Search Term Marketing Program: A Return on Investment Approach. *Proceedings of the Workshop on Data Mining Case Studies at the 5th IEEE International Conference on Data Mining (ICDM)* (pp. 60–72). Houston, Texas, USA.

Smith, J. M., & Smith, D. C. P. (1977). Database Abstractions: Aggregation and Generalization. *ACM Transactions on Database Systems*, *2*, 105–133.

Soares, C., Moniz, L., & Duarte, C. (Eds.). (2005). *Proceedings of the Workshop on Data Mining and Business (DMBiz) at the 9th European Conference on Principles and Practice in Knowledge Discovery in Databases (PKDD)*. Porto, Portugal.

Srikant, R., & Agrawal, R. (1995). Mining Generalized Association Rules. *Proceedings of 21th International Conference on Very Large Data Bases* (pp. 407–419). Zurich, Switzerland: Morgan Kaufmann.

Staab, S. (2002). Knowledge Portals. *Kunstliche Intelligenz*, *1*, 38–39.

Staab, S., & Studer, R. (2004). *Handbook on Ontologies*. Berlin: Springer.

Staudt, M., Vaduva, A., & Vetterli, T. (1999a). *Metadata Management and Data Warehousing* (Technical Report). Swiss Life, Information Systems Research and University of Zurich, Department of Computer Science.

Staudt, M., Vaduva, A., & Vetterli, T. (1999b). *The Role of Metadata for Data Warehousing* (Technical Report). Swiss Life, Information Systems Research and University of Zurich, Department of Computer Science.

Storey, V. C. (1993). Understanding Semantic Relationships. *The VLDB Journal*, *2*, 455–488.

Su, H., Claypool, K., & Rundensteiner, E. (2000). Extending the Object Query Language for Transparent Metadata Access. *Database Schema Evolution and Meta-Modeling – Proceedings of the International Workshop on Foundations of Models and Languages for Data and Objects (FoMLaDO/DEMM)*.

Sun, W., Ling, Y., Rishe, N., & Deng, Y. (1993). An Instant and Accurate Size Estimation Method for Joins and Selections in a Retrieval-Intensive Environment. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 79–88). New York, NY, USA: ACM Press.

Suryn, W., Abran, A., & April, A. (2003). ISO/IEC SQuaRE. The Second Generation of Standards for Software Product Quality. *Proceedings of the 7th IASTED International Conference on Software Engineering Applications*. Marina del Rey, CA, USA.

Svatek, V., Rauch, J., & Flek, M. (2005). Ontology-Based Explanation of Discovered Associations in the Domain of Social Reality. *Proceedings of the Workshop on Knowledge Discovery and Ontologies (KDO) at the 9th European Conference on Principles and Practice in Knowledge Discovery in Databases (PKDD)* (pp. 75–86). Porto, Portugal.

Tan, J., Zaslavsky, A. B., Ewald, C. A., & Bond, A. (2003). Domain-Specific Metamodels for Heterogeneous Information Systems. *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS).*

Tang, Z., & MacLennan, J. (2005). *Data Mining with SQL Server 2005.* Wiley & Sons.

Teorey, T. J., Yang, D., & Fry, J. P. (1986). A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. *ACM Computing Surveys*, *18*, 197–222.

Thalheim, B. (2000). *Entity-Relationship Modeling. Foundations of Database Technology.* Springer.

Theodoratos, D., & Xu, W. (2004). Constructing Search Spaces for Materialized View Selection. *Proceedings of the 7th ACM International Workshop on Data Warehousing and OLAP (DOLAP)* (pp. 112–121). New York, NY, USA: ACM Press.

Tsichritzis, D., & Klug, A. C. (1978). The ANSI/X3/SPARC DBMS Framework Report of the Study Group on Dabatase Management Systems. *Information Systems*, *3*, 173–191.

Tsochantaridis, I., Joachims, T., Hofmann, T., & Altun, Y. (2005). Large Margin Methods for Structured and Interdependent Output Variables. *Journal of Machine Learning Research*, *6*, 1453–1484.

Ullman, J. D. (1988). *Principles of Database and Knowledge-Base Systems*, vol. 1. Rockville, MD: Computer Science Press.

Vaduva, A., & Dittrich, K. R. (2001). Metadata Management for Data Warehousing: Between Vision and Reality. *Proceedings of the International Database Engineering & Applications Symposium (IDEAS)* (pp. 129–135). IEEE Computer Society.

Vapnik, V. (1982). *Estimation of Dependencies Based on Empirical Data.* Springer.

Vapnik, V. (1998). *Statistical Learning Theory.* Chichester, GB: Wiley.

Vetterli, T., Vaduva, A., & Staudt, M. (2000). Metadata Standards for Data Warehousing: Open Information Model vs. Common Warehouse Metamodel. *ACM SigMod Record*, *29.*

Vilalta, R., Giraud-Carrier, C., Brazdil, P., & Soares, C. (2004). Using Meta-Learning to Support Data Mining. *International Journal of Computer Science and Applications*, *1*, 31–45.

Wache, H., Vogele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., & Hubner, S. (2001). Ontology-Based Integration of Information—A Survey of Existing Approaches. *Proceedings of the IJCAI 2001 Workshop on Ontologies and Information Sharing.*

Wagner, M. (2005). Schema-Abbildungen fur die Falladaption in MiningMart. Master's thesis, Fachbereich Informatik, Universitat Dortmund. In German.

Wang, H., & Zaniolo, C. (1999). User-Defined Aggregates for Datamining. *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery.*

Washio, T., & Motoda, H. (2003). State of the Art of Graph-Based Data Mining. *ACM SIGKDD Explorations Newsletter*, *5*, 59–68.

Wiederhold, G. (1992). Mediators in the Architecture of Future Information Systems. *IEEE Computer*, *25*, 38–49.

Wielinga, B., Schreiber, G., & Breucker, J. (1993). Modelling Expertise. In G. Schreiber, B. Wielinga and J. Breucker (Eds.), *KADS – A Principled Approach to Knowledge-Based System Development*, vol. 11 of *Knowledge Based Systems*, 21–46. London: Academic Press.

Williams, G. J., & Huang, Z. (1996). *Modelling the KDD Process* (Technical Report TR-DM-96013). CSIRO (Commonwealth Scientific and Industrial Research Organisation), DIT Data Mining.

Wirth, R., Shearer, C., Grimmer, U., Reinartz, T., Schlosser, J., Breitner, C., Engels, R., & Lindner, G. (1997). Towards Process-Oriented Tool Support for KDD. *Proceedings of the 1st European Symposium on Principles of Data Mining and Knowledge Discovery.*

Witten, I., & Frank, E. (2000). *Data Mining – Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann.

Wolpert, D., & Macready, W. (1995). *No Free Lunch Theorems for Search* (Technical Report SFI-TR-95-02-010). Santa Fé Institute, Santa Fé, CA.

Wrobel, S. (1997). An Algorithm for Multi–relational Discovery of Subgroups. *Principles of Data Mining and Knowledge Discovery: First European Symposium (PKDD 97)* (pp. 78–87). Berlin, New York: Springer.

Wu, X., Yu, P. S., Piatetsky-Shapiro, G., Cercone, N., Lin, T. Y., Kotagiri, R., & Wah, B. W. (2003). Data Mining: How Research Meets Practical Development? *Knowledge and Information Systems*, *5*, 248–261.

Wyss, C. M., & Robertson, E. L. (2005a). A Formal Characterization of PIVOT/UNPIVOT. *Proceedings of the 14th ACM International Conference on Information and Knowledge Management (CIKM)* (pp. 602–608). New York, NY, USA: ACM Press.

Wyss, C. M., & Robertson, E. L. (2005b). Relational Languages for Metadata Integration. *ACM Transactions on Database Systems*, *30*, 624–660.

Yan, L. L., Miller, R. J., Haas, L. M., & Fagin, R. (2001). Data-Driven Understanding and Refinement of Schema Mappings. *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data* (pp. 485–496). New York, NY, USA: ACM Press.

Zhang, T., Ramakrishnan, R., & Livny, M. (1996). BIRCH: An Efficient Data Clustering Method for Very Large Databases. *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* (pp. 103–114).

Zhong, N., Liu, C., & Ohsuga, S. (2001). Dynamically Organizing KDD Processes. *International Journal of Pattern Recognition and Artificial Intelligence*, *15*, 451–473.

The URLs cited in this thesis were last visited on June 5th, 2007.

Index