UNIVERSITÄT DORTMUND
Fachbereich Informatik

Lehrstuhl VIII
Künstliche Intelligenz

Data Preparation for Inductive Learning in Robotics

LS–8 Report 19

Anke Rieger

Dortmund, May 22, 1995

Universität Dortmund
Fachbereich Informatik

University of Dortmund
Computer Science Department

Forschungsberichte des Lehrstuhls VIII (KI)
Fachbereich Informatik
der Universität Dortmund

ISSN 0943-4135

Anforderungen an:

Universität Dortmund
Fachbereich Informatik
Lehrstuhl VIII
D-44221 Dortmund

Research Reports of the unit no. VIII (AI)
Computer Science Department
of the University of Dortmund

ISSN 0943-4135

Requests to:

University of Dortmund
Fachbereich Informatik
Lehrstuhl VIII
D-44221 Dortmund

e-mail: reports@ls8.informatik.uni-dortmund.de
ftp: ftp-ai.informatik.uni-dortmund.de:pub/Reports
www: http://www-ai.informatik.uni-dortmund.de/ls8-reports.html

Data Preparation for Inductive Learning in Robotics

LS–8 Report 19

Anke Rieger

Dortmund, May 22, 1995

Universität Dortmund
Fachbereich Informatik

Abstract

The application of logic-based learning algorithms in real-world domains, such as robotics, requires extensive data engineering, including the transformation of numerical tabular representations of real-world data to logic-based representations, feature and concept selection, the generation of the respective descriptions, and the composition of training and test sets, which meet the requirements of the respective learning algorithms. We are developing a tool, which supports a user of inductive logic-based algorithms with handling these tasks. The tool is developed in the context of a robot navigation domain, in which different logic-based algorithms are applied to learn operational concepts.

(This paper will appear in the Proceedings of the IJCAI-Workshop on Data Engineering for Inductive Learning, 1995.)

1 Introduction

When applying logic-based machine learning algorithms to real-world domains, such as robotics, users are faced with the time consuming task of preparing training and test sets for learning and evaluating results, respectively. This task is rather complex and requires the solution of different problems. The main problem addresses representation: The gap between the requirements of the learning algorithms and the characteristics of the real-world data has to be bridged. The numerical, extremely large and extremely detailed data sets, usually represented in tabular form, have to be transformed into a description represented in a restricted first-order logic. Transforming the description of the real-world data into a logic-based representation formalism is not sufficient, because it may contain too many examples and possibly too many (irrelevant) features for learning. As a consequence, there is the danger of burdening the learning algorithms with too many examples and/or too many features, thus overloading it with redundant and irrelevant information.

So, in addition to the question, *how* to represent the data, the users have to decide, *what* to represent, i.e., which concepts are to be learned. When learning operational concepts (see below), it might not be possible, to learn to derive them directly from the real-world data. Thus, intermediate concepts may have to be introduced. Given the concepts to be learned, examples and background knowledge, which constitute the training sets, have to be generated.

Given examples and background knowledge, users of machine learning algorithms have to compose training sets for learning. During this phase, they make a lot of choices with respect to, for example, the portion of positive and negative examples, for learning one versus multiple concepts. These choices often remain hidden and are rarely documented. However, as the success of applying learning algorithms depends heavily on them, there is a need for making them more explicit.

We are developing a tool, which supports users of machine learning algorithms with the solution of the tasks, mentioned above. The need for this tool arose in a specific domain of application, namely the robot navigation domain, developed within the BLearn-project. Within this project, different learning algorithms were developed and applied. With different users and developers involved, there was the urgent need, to develop a framework, within which the data engineering problems could be solved in a uniform way, in order to avoid, that each user produced another program for accomplishing the same task.

In Section 2, we explain, in the context of learning operational concepts for robot navigation, the general setting, in which the tool is working. In Section 3, we sketch the robotics domain, in order to illustrate the data engineering problems with concrete examples. We focus on the requirements for the learning algorithms in Section 4. In Section 5, we present the tool and show in detail, how operations for data structuring, example generation, composition of samples, and preparing training sets for learning are realized. We conclude with a discussion of related and future work in Section 6.

2 The General Setting

We are developing the tool within the context of a robot navigation domain, to which inductive algorithms are applied, in order to learn *operational concepts*. These concepts were introduced in [1]. They are to be used by a robot to perform flexibly user-defined tasks, such as `move through the door`, `turn right`, and `move to the cupboard`. Operational concepts, on one hand, provide the basis for high-level planning. On the other hand, they are symbolically grounded, in the sense, that they can be traced down to sonar sensor measurements and basic robot actions. The point is, that operational concepts cannot be learned directly from the real-world data. Thus, they constitute the highest level of the abstraction hierarchy in Figure 1. In order to bridge the gap between the

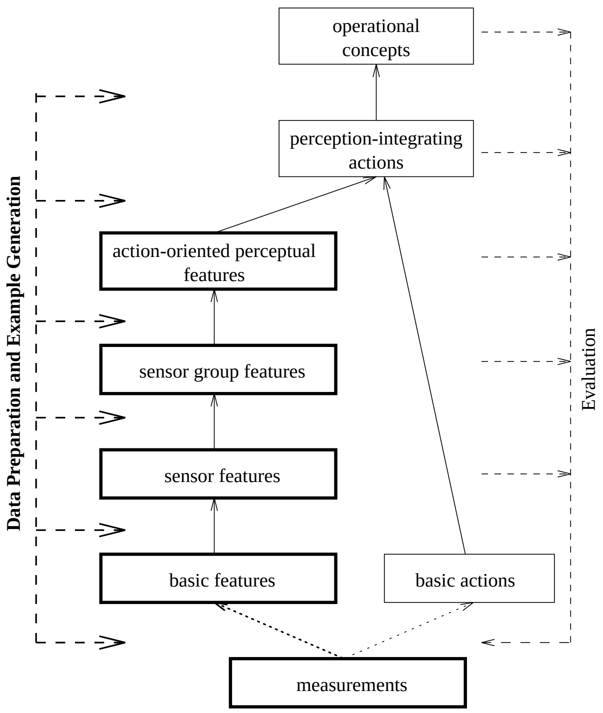

Figure 1: The abstraction hierarchy

real-world data and operational concepts, we introduced intermediate levels of abstraction. We had to decide, which information is represented on each level. Then, the strategy is to apply inductive algorithms, in order to learn, how to derive higher-level features from lower-level ones. The learning steps are indicated in Figure 1 by the directed non-dashed arcs. For each step, examples and background knowledge have to be generated, and training sets have to be put together. After learning, the evaluation of the learning results can yield valuable information for the lay-out of the next learning steps, and feedback for the previous learning steps. Figure 1 is not to be understood in the sense, that learning operational concepts is a process, which is finished after having run bottom-up through the learning steps once. Figure 2 illustrates this cyclic nature of learning. In principle, it represents the same situation as Figure 1. Whereas in the first one we focused on the

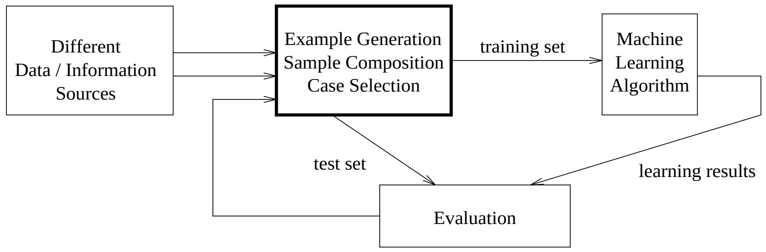

Figure 2: The general setting for learning

domain-dependent representation, here we focus on the domain-independent methodological setting for learning, especially on the role of data preparation, i.e., example generation, composing samples, and case selection.

The general setting in Figure 2 is, conceptually, anything but new. However, wherever it appeared in the literature, the focus turned, in most cases, to the learning algorithms instead of turning the data engineering problems. In the following, we present a tool, which we are developing, in order to offer users of machine learning algorithms a uniform framework for data preparation:

1. The tool supports Horn clauses as *representation formalism*, and thus can be used for, but is not restricted to, inductive logic programming (ILP) algorithms.

2. It allows to *structure* the data, which is used for learning.

3. It facilitates the *access to data* stemming from different sources.

4. It offers operations for *generating examples* of the concepts to be learned, and *background knowledge*, from which the concepts are to be learned.

5. It offers operations for *composing samples* according to *statistical criteria* or *criteria referring to features* of the data.

6. It supports the generation of different *data structures*, e.g., sets of ground facts or *cases* (see below), which are required as input by different learning algorithms.

Our main goal is, that, by using this tool, we will succeed in separating the domain-independent aspects of the data engineering operations from the domain-dependent knowledge about the respective application. This separation has often been neglected, as in most cases the users of the algorithms were also their developers, who, therefore, concentrated more on the learning algorithms. The hope is, that the use of this tool will make the data preparation phase more transparent, thus revealing more clearly its effect on the learning results.

3 The Robotics Domain

In this section, we give an overview of the application domain, in order to illustrate the concrete data engineering problems. The focus is on representational issues, i.e., we deal with the question, about what information is represented at each level of the abstraction hierarchy, and how these descriptions are generated from the real-world data.

In [1], operational concepts were introduced, which are abstract descriptions of concepts, such as `move through the door`, and `move along the wall`. They constitute the interface between a human user and a robot: The user uses operational concepts to guide the robot in known as well as unknown environments. The robot, in turn, uses them to report on its activities.

Operational concepts are learned from data [1] about robot traces in known environments, such as the one, illustrated in Figure 3. During a trace the following information

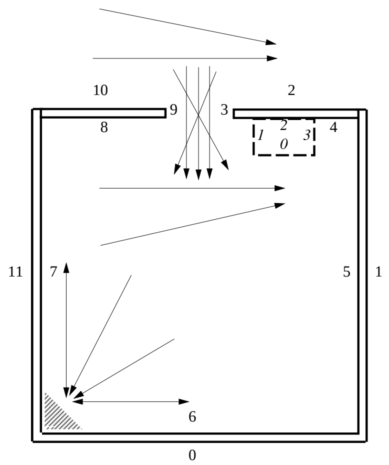

Figure 3: Room with robot traces

is recorded for each of the 24 sensors at successive time points: The sonar sensor measurement, which is interpreted as sensed distance, the sensor orientation and position, as well as the robot orientation and position in the global coordinate system. As the environment is known, the coordinate of the sensed point and the respective object in the scene are recorded as well.

From this data higher-level concepts, such as basic features, sensor features, sensor group features, and action-oriented perceptual features (see Figure 1) are derived. An example for an *action-oriented perceptual feature* is the concept of moving through a doorway. Let `t12` denote one of the traces in Figure 3, in which the robot moves `parallely` through the doorway during the interval from time point `1` to `15`. This fact is represented by the ground literal [2]

`through_door(t12,1,15,parallel) .`

With reference to Figure 3, we illustrate, what happens during robot traces, in which the robot moves through a doorway: The sensors on the robot's right side will first perceive the doorframe, labelled `9`, and then the wall, labelled `7`. Correspondingly, the sensors

[1] The data has been provided by the University of Karlsruhe

[2] We use a Prolog-like notation, i.e., variables begin with capital letters, constants with small letters.

on the robot's left side will perceive wall `3` and `5`. In [4], we introduced the term `jump` for these kind of edge groupings, consisting of two parallel edges. Other edge groupings, which are considered, are `convex` and `concave` corners, and singular edges, called `line`. For example, during trace `t12`, the sensors on the robot's `right` and `left` side perceive "jumps" during the interval from time point `1` to `10`. These facts are represented by the ground literals

`sg_jump(t12,right,1,10,parallel)` and `sg_jump(t12,left,1,10,parallel)`.

Thus, the predicates for *sensor group features* state, that an edge grouping of a specific type (denoted by the predicate name) has been perceived by a group of sensors (second argument) in a trace (first argument) during a time interval (third and fourth argument), during which the robot moved in a relative orientation (fifth argument) along the object. *Sensor features* also combine the perception of an edge grouping with the relative movement of the robot, but this time the information refers to a single sensor, e.g.,

`s_jump(t12,s5,1,10,parallel)`.

While moving, sensor `s5` receives a sequence of sonar sensor measurements, illustrated in Figure 4. They are grouped together in time intervals, during which the tendency of change of the measurements remains approximately the same (for details see [1],[9]). These time intervals are described by the *basic features* on the right side of Figure 4. The

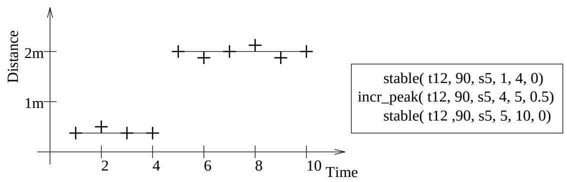

Figure 4: Sequence of sensor measurements

first one states, that in trace `t12` sensor `s5` received during the time interval from time point `1` to `4` approximately `stable` measurements. During this time interval the robot perceived the doorframe, labelled `9`. The stable measurements during the time interval from `5` to `10` correspond to the wall `7`. The measurements at time points `4` and `5` differ significantly, which is reflected by the `incr_peak` predicate. (The second argument of the basic features denotes the orientation of the sensors. It is taken into account in order to ensure, that the sensor orientation does not change. The last argument denotes the average gradient, which was measured during the time interval. It thus adds more details to the classification `decreasing`, `increasing` etc.).

Once we have decided, which information will be represented on the different levels of the abstraction hierarchy, and which predicates will be used, the next problem is to generate instances of these predicates from the initial data, which will be used for learning. For this purpose, different algorithms have been implemented: As we are working within a restricted first-order logic framework, the initial data has to be transformed from a numerical, tabular representation to a relational one. Wessel [9] has implemented a program, which calculates basic features from sensor measurements. The calculation is influenced by different parameters, which reflect the sensitivity to changes in the measurements. For the sequence of measurements illustrated in Figure 5 depending on the parameters, the

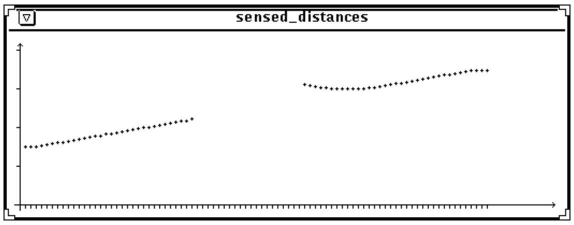

Figure 5: Sensor measurements

algorithm produces different sequences of basic features, e.g.,

```
        increasing(t76,75,s6,3,32,13).
    no_measurement(t76,75,s6,32,53,999).
        decreasing(t76,75,s6,53,59,-9).
            stable(t76,75,s6,59,65,1).
        increasing(t76,75,s6,65,69,11).
something_happened(t76,75,s6,69,70,18).
        increasing(t76,75,s6,70,85,13).
```

or

```
    increasing(t76,75,s6,3,32,13).
no_measurement(t76,75,s6,32,53,999).
        stable(t76,75,s6,53,69,0).
    increasing(t76,75,s6,69,85,13).
```

Another program is used to derive instances of sensor features: Given the knowledge about the environment, i.e., the position of the walls, doorframes, cupboards in the room, and given the knowledge, which components of the scene constitute a certain type of edge grouping, instances of the sensor feature predicates can be derived.

The point is, that there exists a heterogeneous collection of programs, which produce an enormous collection of data files, from which learning and test sets are to be generated. Several users and programmers are involved, so that it is hard to keep track of what each program does and which file was generated by which program with which parameter combination. In the resulting cemetery of data files the users of machine learning algorithms have to find those files, which they want to use.

4 The Learning Algorithms

Given the representation of the concepts at the different levels of the abstraction hierarchy, we sketch the algorithms, which were used to learn, how to derive higher-level concepts from lower-level ones. This is to illustrate the requirements, which come from the learning algorithms, and which the data engineering tool has to meet.

Within the BLearn-project different types of learning algorithms have been applied:

1. Inductive logic programming (ILP) algorithms, which produce rules, represented as Horn clauses, which derive higher-level concepts from lower-level ones.

2. An algorithm, which infers automata, which take as input sequences of predicate instances describing features, and whose final states are associated with concepts, which can be derived from the features.

Both algorithms work on background knowledge B, positive and negative examples $E = E^+ \cup E^-$, and a hypothesis space H.

In the ILP-context, the examples have to be consistent with the background knowledge (i.e., $B, E \not\models \Box$), and must not already be a consequence of the background knowledge ($B \not\models E$). Then, the goal is to find a hypothesis $h \in H$, which is consistent with B and E (i.e., $B, h, E \not\models \Box$), and which explains together with B the positive examples (i.e., $B, h \models E^+$) and none of the negative ones (i.e., $B, h \not\models E^-$).

Different ILP-algorithms, which are integrated in the MOBAL-system [5], have been applied to learn rules, which derive higher-level concepts of the abstraction hierarchy in Figure 1 from lower-level ones (for details, see, e.g., [4], [1], [8]). Some examples of the rules, which derive action-oriented perceptual features, sensor group features, and sensor features, are given in the following:

```
through_door(Trace,Start,End,parallel) <-
  sg_jump(Trace,left,T1,T2,parallel) &
  sg_jump(Trace,right,T1,T2,parallel) &
  Start < T1 & T2 < End.
```

This rule states, that the robot moved `parallel`y through a doorway in a `Trace` during the interval from time point `Start` to `End`, if, during a subinterval, the sensors on the robot's `right` and `left` side perceived the edge grouping `jump`. Sensor group features are derived, if sufficiently many sensors, which are adjacent and belong to the same class, have perceived the same edge grouping:

```
sg_jump(Trace,right,TS,TE,parallel) <-
  s_jump(Trace,Sensor1,TS,TE,parallel) &
  s_jump(Trace,Sensor2,TS,TE,parallel) &
  adjacent(Sensor1,Sensor2) &
  sclass(Trace,Sensor1,T1,T2,right) &
  sclass(Trace,Sensor2,T1,T2,right) &
  T1 < TS & End < TE.
```

Sensor features are defined in terms of sequences of basic features:

```
s_jump(Trace,Sensor,T1,T4,parallel) <-
  stable(Trace,Or,Sensor,T1,T2,Grad1) &
incr_peak(Trace,Or,Sensor,T2,T3,Grad2) &
  stable(Trace,Or,Sensor,T3,T4,Grad3).
```

Note, that in the case of rules, the temporal order of observations, e.g., basic features, is reflected in the chaining of the variables, representing time points. In the case of basic features these time points appear as fourth and fifth argument of the basic features.

In the case of the ILP-algorithms, the hypothesis space, H, consists of Horn clauses. In the case of the algorithm, presented in [7], it consists of the class of deterministic finite state automata and probabilistic automata, respectively. These automata accept as input sequences of observations, i.e., ground instances of basic features, which are accepted, if they lead to a final state, which represent concepts, i.e., sensor features.

A common feature of both types of algorithms is, that B and E consist of ground literals. For the predicates, appearing in E, we also use the term *target predicates*, for the predicates, appearing in B, we use the term *defining predicates*. In the ILP-case, target predicates may appear in the conclusion of a learned rule, defining predicates in it's premise. Training sets are constructed by selecting examples, searching the relevant background knowledge, and merging both, target predicate and defining predicate instances, in a flat set of ground literals. In the case of automata inference, the training set consists of sequences of observations, which have to or do not have to be accepted by the automaton. Thus, each target predicate instance in E has to be associated with the relevant sequence of defining predicates in B, which also has to reflect the temporal order of the observations. So both algorithms pose different requirements for the representation of training sets.

5 The Tool

In order to summarize the situation: Our goal is to learn in several steps logic-based descriptions of abstract concepts from real-world data. In order to accomplish this task, intermediate concepts are introduced. Programs exist, which derive instances of the predicates, representing the respective concepts from the initial data. By introducing different levels in the hierarchy, we abstract from more and more details in the original data. Nevertheless, an enormous amount of data is generated, which has to be structured, in order to support the user with preparing training sets for learning. For each learning step, the user has to start with a set of conceptual decisions, whose realization is supported by our tool:

1. Decision about which information is to be represented at the target level and defining level, respectively.

Each learning step is to bridge the gap between two levels of the abstraction hierarchy. The lower level provides the background knowledge, i.e., the defining predicates, the upper level the examples for the concepts to be learned, i.e., the target predicates.

2. Decision, which programs with which parameter settings will be used to generate instances of the predicates, which are used for representing examples and background knowledge.

3. Decision about which target concepts are to be learned.

In the case of learning sensor features from basic features, for example, the user has to decide, whether the different sensor features are to be learned together or separately in several learning runs.

4. Decision about which negative examples are to be used.

For example, in the case of learning the sensor feature `s_line`, the instances of the predicate `s_convex`, `s_concave`, and `s_line` can be used as negative example for the concept `s_line`.

5. Decision about which target predicate instances are to be put in the example set E.

This decision concerns the questions,

- which features are to be used to select the examples for the concepts to be learned: Given a room (e.g., the one in Figure 3), in which traces were pursued, and given the sensor features for these traces, we might want to assign the positive examples of the sensor feature `s_convex` for the traces `t1,...,t15` to the learning set and the examples for `s_convex` of the other traces to the test set.
- which statistical criteria are used:
  Given a set of examples for sensor features, one might
  - use 60 % of them in the learning set and the rest in the test set;
  - draw a certain percentage or a specific number of examples according to a specified distribution;
  - split up the sample into a number of subsamples (cross validation).

6. Decision about which background knowledge is to be used during learning.
7. Decision, about which learning algorithm is to be used.

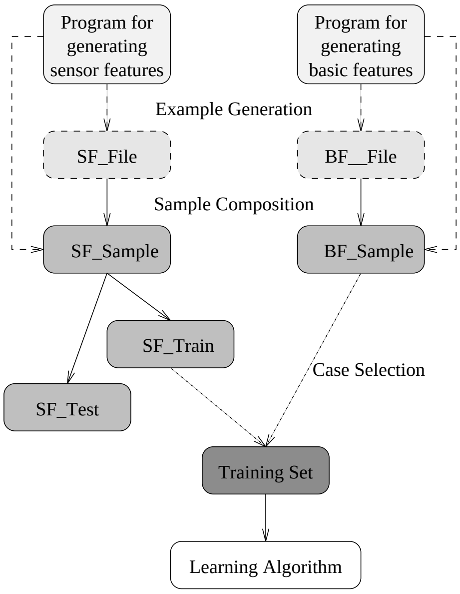

Figure 6: Data preparation

Data Structuring Once, the user has decided on the learning set-up, she has to go through the same sequence of data preparation steps for each learning run. These data preparation steps are illustrated in Figure 6 for the special case of learning sensor features from basic features. The user has to define, which predicates constitute the examples, E, and which predicates constitute the background knowledge, B. The tool supports the user in doing this by the definition of *abstraction levels*, which contain a group of predicates, which are similar in the following sense:

- the predicates have the same arity, n, and
- the argument structure of the predicates is the same, i.e., the i-th argument, $1 \leq i \leq n$, of each of the predicates is of the same sort.

In our application domain, each level of the abstraction hierarchy in Figure 1 represents concepts, which are described by a group of predicates, which have such a structure. Basic features are represented by predicates of arity 6

<bf>(<tr>,<o>,<s_id>,<start>,<end>,<gr>),

where the predicate name, <bf>, is in the set BF={increasing, stable, decreasing, ...}.The arguments are of specific sorts, denoting traces, <tr>, sensor orientations, <o>, sensor identifications, <s_id>, start and end points of time intervals, <start> and <end>, and gradients , <gr>, of successive measurements. Sensor features are described by predicates of arity 5

<sf>(<tr>,<s_id>,<start>,<end>,<rel_or>),

where <rel_or> represents the relative orientation of the robot towards the edge grouping, which is denoted by the predicate name, <sf>, which is an element of SF={s_line, s_concave,s_convex,s_jump}. In this way the user can define two abstraction levels, **bf** and **sf**, for basic features and sensor features, respectively.

Sample Composition When learning to derive sensor features from basic features, **bf** contains the defining predicates, and **sf** the target predicates. The next step is to generate instances of both groups of predicates (example generation), which are then used to compose samples (see Figure 6). A *sample* contains positive and negative instances of predicates, which are specified by a *reference class*, e.g., an abstraction level. The instances of the sample stem from a given *source*, which is

- a program, which generates the instances (example generation),
- a data file, containing the instances, or
- another already existing sample.

Samples can be composed according to two types of criteria: statistical criteria and criteria, which refer to features of the objects, represented by the predicates. At the moment, features are specified by constraints on values of arguments at certain positions of the predicates. Assume the situation, that a user has generated a data file, SF_File, which contains all sensor features for 20 traces (see Figure 6). Furthermore, the program for generating basic features has been used to generate instances of those predicates for the same traces, which are stored in the data file, `BF_File`. The goal is to use the sensor features of traces `t1,...,t15` for learning, and the rest for testing. The syntax of constraints is `arg(ArgPosition,ValueList)`. The operation for composing a sample according to certain constraints is

```
select(Source,Reference,Constraints,SampleId).
```

Then, the samples `SF_Sample`, `SF_Train`, `SF_Test`, and `BF_Sample` are produced by the following sequence of operations

```
select(file('SB_File'),alevel(bf/6),all,BF_Sample)
select(file('SF_File'),alevel(sf/5),all,SF_Sample)
select(sample(SF_Sample),alevel(sf/5),[arg(1,[t1,...,t15])],SF_Train)
select(sample(SF_Sample),alevel(sf/5),[arg(1,[t15,..,t20])],SF_Test)
```

An alternative way of composing samples is to select instances according to statistical criteria. In this case, the user can specify to draw randomly from a given sample

- a number/percentage of positive instances, and/or
- a number/percentage of negative instances,

which constitute a new sample. This requires the specification of what a negative example is. Negative examples could be given explicitly. Alternatively, e.g., in the case of learning a single sensor feature, `s_line`, the positive instances of the predicates `s_concave`, `s_convex`, and `s_jump` could be used as negative examples. An operation has to be provided, that accomplishes the task of selecting positive and negative examples according to the specifications of the user.

Sample composition according to statistical criteria also requires the specification of the distribution, according to which the random drawing is to be done. Distributions can be, in the simplest case, the uniform distribution, or one, which reflects the frequency distribution of a feature or a feature combination in the data. Thus, operations which determine the frequency distribution for a given sample and a user-specified feature combination have to be provided.

Another alternative, which we will offer for composing samples, is to split up the sample containing the target predicates into subsamples, which can then be used for cross validation.

Case Selection In order to put together a training set, the samples containing the instances of target and predicate instance have to be merged and organized in data structures, which are required as input for the respective learning algorithms (see Figure 6). We have implemented a method, which, given

- a sample with instances of the target predicate(s),
- a sample with instances of the defining predicates,

- a list of relations, which express, how one or several argument values of a target predicate instance restrict or determine the value of an argument of a relevant defining predicate instance,

determines the list of cases from these two samples. A case is represented by a list

`[target instance | defining instances]`,

whose first element is the target predicate instance, and whose rest is the list of relevant defining predicate instances. Consider, e.g., learning sensor features from basic features: An instance of a basic feature is *relevant* for a sensor feature, if it refers to the same trace, to the same sensor, and to the same time interval, respectively. This domain-dependent knowledge can be expressed in a more general setting in terms of relations between values of certain arguments (see Figure 7). The transformation can be realized by using the

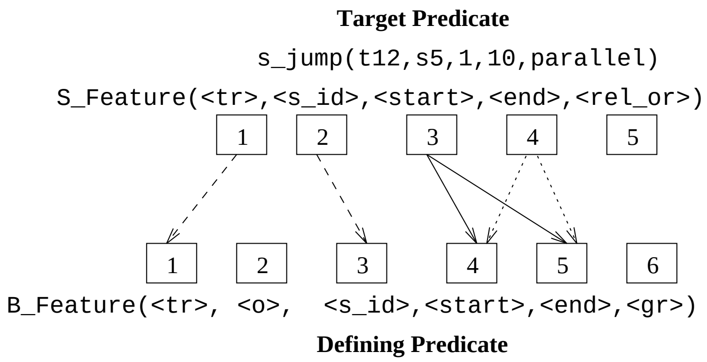

Figure 7: Case selection

declarations of the abstraction levels for sensor features and basic features, respectively.

The algorithm for inferring automata, presented in [7], uses as training data sequences of observations in temporal order, which have or do not have to be accepted by the learned automaton. Thus, if the goal is to infer an automaton, which recognizes sensor features from basic features, the defining instances of the cases constitute the training sequences, and thus have to be sorted according to temporal order. This operation is also provided by the tool.

The application of one of the ILP-algorithms of the MOBAL-system does not necessarily require case selection. With reference to Figure 6, the instances of the samples `SF_Train` and `BF_Sample` could be merged in a flat data set. This way of proceeding involves the risk of overloading the learning algorithm with irrelevant data, which, in view of the enormous amount of data in the domain, is an essential question. Thus, the effort for generating first the cases, and then coverting them to a flat data set, although cumbersome, is worthwhile, as it reduces the time for learning, enormously.

6 Discussion

We have presented a case study of data engineering in a real-world application, i.e., data preparation for learning operational concepts in a robot navigation domain. We have illustrated the specific problems concerning

- the transformation of numerical real-world data to logic-based descriptions of abstract concepts,
- the choice of appropriate representations of concepts,
- data structuring, and
- data preparation for putting together training sets for logic-based learning algorithms.

Given the representational issues and the requirements of the inductive logic-based learning algorithms, we illustrated the decisions, which the user has to make with respect to a learning set-up.

We have presented a software tool, which supports the user with preparing training sets according to the decisions made. It assists the user with structuring the data. By defining abstraction levels, the user can group together similar predicates, which are to be put into a sample. The tool provides several operations for sample composition according to statistical criteria and criteria referring to features of the data. Furthermore, it uses the samples to generate training sets, which can be represented in different ways, as required by different logic-based learning algorithms.

Clearly, the tool is under development. Its presentation made obvious the components, which still have to be implemented. Operations, which are implemented, have to be improved: The composition of samples according to features of objects, represented by the predicates, is realized by specifying constraints on argument values. On one hand, this way of proceeding makes the composition of samples more independent of the domain, to which learning is applied. On the other hand, it requires the user, to know which information is represented by which argument. Data encapsulation may offer an alternative to free the user from specifying argument positions.

In the view of the enormous amount of data, from which the right one has to be selected for putting together a specific training set, the question is, whether the access to the data cannot be made more efficient by using databases and by coupling the tool with the RDT/DB-method developed in [3].

Nevertheless, the tool has already been successfully applied, in order to prepare training sets for two different types of logic-based learningx algorithms. Within this application domain, which is characterized by the enormous amount of data, the most noticeable effect has been, that the tool allowed to put together relevant information for training sets in a structured way, thus preventing the learning algorithms to be overloaded with irrelevant data.

The fact, that the tool works on a restricted first-order logic representation formalism, does not confine its use to a specific domain nor to the two learning algorithms, which we have used. In principle, it can be used for any ILP-algorithm (as for example those presented in [6] and [2]), which require training sets consisting of examples E and background knowledge B, which contain ground literals. It is a known fact, that for all learning algorithms the composition of appropriate training and test sets is crucial. The point is, that users, when applying a learning algorithm, make a lot of choices with respect to the traing sets, which remain hidden in most cases, and which are rarely documented.

However, as the success of applying machine learning algorithms depends critically on the set of choices made, there is a need for making them more explicit. The tool, which has been represented in this paper, forces the user to make her choices explicit in a uniform framework, thus making them amenable to analysis.

Given the two types of learning algorithms we have used, one aspect of future work will be the analysis of the effect of data engineering on the induced models. This is not straightforward, as the induced models - flat rule sets versus automata models, which reflect the structure inherent in the rule set - complement each other. The comparison can only be made by evaluating the performance of the robot, which uses the learned concepts in order to navigate in known as well as unknown environments. The evaluation tool, sketched in Figure 1 and 2, also has not been implemented yet, and thus will be also a topic of future work.

References

[1] V. Klingspor, K. Morik, and A. Rieger. Learning operational concepts from sensor data of a mobile robot. (submitted to Machine Learning Journal), September 1994.

[2] N. Lavrac and S. Dzeroski. *Inductive Logic Programming - Techniques and APplications.* Ellis Horwood, New York, 1994.

[3] G. Lindner. Application of the learning algorithm RDT to a relational database. Master's thesis, Universität Dortmund, 1994. in German.

[4] K. Morik and A. Rieger. Learning action-oriented perceptual features for robot navigation. In *Proc. of the 1st European Workshop on Learning Robots*, 1993. also available as Research Report 3, FB Informatik LS 8, Universität Dortmund.

[5] K. Morik, St. Wrobel, J. U. Kietz, and W. Emde. *Knowledge Acquisition and Machine Learning: Theory, Methods, and Applications.* Addison Wesley, 1993.

[6] St. Muggleton, editor. *Inductive Logic Programming.* Academic Press, 1992.

[7] A. Rieger. Inferring probabilistic automata from sensor data for robot navigation. In *Proc. of the 3rd European Workhop on Learning Robots*, 1995.

[8] St. Sklorz. Representing and learning operational concepts. Master's thesis, Universität Dortmund, 1995. in German.

[9] St. Wessel. Learning qualitative features from numerical robot sensor data. Master's thesis, Universität Dortmund, 1995. in German.