Projektgruppe 543:
Galaxy Quest

Teil 1: Endbericht

2. November 2010

Hayati Bice Fritz Boekler Tobias Brinkjost
Davud Colakovic Malte Isberner Martha Klois
Denis Kurz Nicolai Parlog Thomas Siwczyk
Hendrik Spiegel Georg von der Brüggen

Betreuer:
Dipl.-Inf. Fabian Gieseke
Prof. Dr. Jan Vahrenhold
Dr. Oliver Kramer
Dipl.-Inf. Andreas Thom
PD Dr. Dominik J. Bomans
Dipl.-Inf. Kai Polsterer

Fakultät für Informatik
Algorithm Engineering (Ls11)
Technische Universität Dortmund
http://ls11-www.cs.tu-dortmund.de

Vorwort

In der *Diplomprüfungsordnung Informatik* der TU Dortmund ist in der Fassung von 2001 [3] für jeden Studierenden des Fachs Kerninformatik oder Angewandte Informatik im Hauptstudium die Teilnahme an einer Projektgruppe (PG) vorgesehen. Eine PG besteht dabei in der Regel aus acht bis zwölf Teilnehmern und mindestens einem Betreuer. Unter dessen bzw. deren Anleitung werden über zwei Semester Methoden der Informatik vermittelt, vertieft und angewandt. Ziel der PG ist neben der Umsetzung des von den Betreuern im Projektgruppenantrag vorgegebenen Projektziels die selbstständige Organisation der Teilnehmer, was eine Vorbereitung auf das in der späteren Berufspraxis wichtige Arbeiten in – auch heterogenen – Teams bieten soll.

Da es zu den Aufgaben einer PG gehört, ihre Organisation und Planung sowie auftretende Probleme und Fortschritte zu dokumentieren, sollen ein Zwischenbericht, ein Abschlussbericht und gegebenenfalls ein Handbuch angefertigt werden. Dieser Abschlussbericht ist Teil der Dokumentation der Projektgruppe 543 der TU Dortmund und beinhaltet eine zusammenfassende Übersicht über den gesamten Projektablauf. Das während dieser Zeit entwickelte System Brick ist im dazugehörigen Handbuch ([43]) umfassend dokumentiert, weswegen in diesem Bericht nur im Rahmen der Ergebnissdarstellung ein Überblick über dessen Architektur gegeben wird.

Zunächst wird in der Einleitung näher auf die Motivation und die Problemstellung eingegangen und die organisatorische Struktur der Projektgruppe vorgestellt. Die folgenden Kapitel decken aufeinander aufbauende Phasen des Projekts ab und geben dabei jeweils einen Überblick über die Aufteilung der Gruppe, die Zielsetzung für diesen Zeitraum und einige der erarbeiteten Ergebnisse. Abschließend erfolgt ein Vergleich der an die PG gestellten Ansprüche mit den erreichten Zielen und ein Ausblick auf eine mögliche Fortsetzung des Projekts und auf dann mögliche Erweiterungen.

Inhaltsverzeichnis

1 Einleitung

In dieser Einleitung wird näher auf die Problemstellung und die Motivation des geplanten Projekts eingegangen. Des Weiteren werden das geplante Ziel der Projektgruppe und deren organisatorischer Rahmen näher erläutert.

1.1 Problemstellung und Motivation

Die Astronomie hat das Ziel, die Vorgänge im Universum zu untersuchen und zu erklären. Die Beobachtung des Himmels stellt also eine wichtige Datenquelle dar, weswegen fortwährend Surveys (systematische Durchsuchungen eines großen Himmelsausschnitts) durchgeführt werden. Die Ergebnisse dieser aufwendigen Projekte werden in Katalogen gesammelt, welche die Grundlage für weitergehende Forschung bilden. Das Internetzeitalter ermöglicht es, verschiedene Kataloge – wenn auch teilweise mit großem Aufwand – zu verbinden und ihre Daten vereinheitlicht als *virtuelle Observatorien* [6] online zur Verfügung zu stellen.

Die digitale Erfassung der Kataloge stellt für die Astronomie eine deutliche Vereinfachung des Alltags dar. Sie ermöglicht den Astronomen mit Hilfe spezieller Programme sowohl eine bessere und deutlich komfortablere Analyse bereits bekannter als auch die Entdeckung neuer Objekte und Phänomene im Universum. Einige der häufig verwendeten Hilfsmittel zur Analyse der gewonnenen Daten sind *Topcat* [21], *VO Enabled Mirage* [11] und *Google Earth* [5] mit Sky-Erweiterung. Diese Programme greifen wiederum auf die Daten der virtuellen Observatorien zu (vgl. [54]). Sie bieten eine gute Basis für eine manuelle Evaluation der Daten.

Eines der Probleme vor dem die heutige Astronomie steht ist es, ein stetig wachsendes Datenaufkommen, welches die immer leistungsstärkeren Teleskope täglich liefern, vernünftig auswerten und verarbeiten zu können. Der Katalog des *Sloan Digital Sky Survey* (SDSS) [17] basiert auf Rohdaten von ca. 30 TB Größe und das *Large Synoptic Survey Telescope* wird bis zum Jahr 2025 Daten im Umfang von etwa 200 PB katalogisieren [54].

Diese Datenmengen *manuell* zu verarbeiten (d.h. z.B. mit den oben genannten Programmen zu klassifizieren oder nach ungewöhnlichen Objekten zu durchsuchen) ist unmöglich; hier bieten sich die Verfahren aus dem Bereich des Data Mining als mögliche Lösung an. *Data Mining* wendet systematisch Methoden auf eine Datenmenge an, um bestimmte Muster zu erkennen. Diese Verfahren besitzen kein in Code umgesetztes „Verständnis“ des Problems und sind deswegen bei Lösungen nicht auf existierendes Vorwissen festgelegt. Außerdem wurden sie gezielt entwickelt, um auch auf großen Datenbeständen schnell zu arbeiten. Beides prädestiniert sie zur Verwendung in der Astronomie und dies ist der Punkt, an dem die Projektgruppe ansetzte.

1.2 Ziele der Projektgruppe

Die Ziele der Projektgruppe 543 wurden im entsprechenden Antrag [54] formuliert. Der Fokus lag darauf, ein System zu entwickeln, welches als Hilfsmittel für Astronomen dienen soll. Es

soll mit dem System möglich sein, die Daten des SDSS-Katalogs effizient zu betrachten und analysieren zu können. Bei der Verarbeitung der vorliegenden großen Datenmengen sollen unter anderem Verfahren aus dem Bereich des Data Mining unterstützt werden.

Weiterhin war im Projektantrag ein erster Schritt vorgesehen, in dem eine Benutzeroberfläche zu entwickeln war, welche eine visuelle Darstellung und Erforschung der Bilddaten des SDSS-Katalogs ermöglicht. Außerdem sollte eine allgemeine Schnittstelle zur Einbindung von Data-Mining-Algorithmen implementiert werden, so dass der Einsatz einiger elementarer Verfahren erprobt und ausgewertet werden kann. Des Weiteren war es in Zusammenarbeit mit dem Lehrstuhl für Astronomie der Ruhr-Universität Bochum vorgesehen, forschungsrelevante Fragestellungen aus der Astronomie, wie z.B. das Auffinden seltener Objekte (*Ausreißerdetektion*) und die automatische Einteilung in Gruppen (*Klassifikation*), zu identifizieren. Im letzten Schritt des Projektverlaufs sollten hierfür mögliche Lösungsverfahren entwickelt werden.

Im Anschluss an eine Seminarphase, welche als Einführung am Anfang der Projektgruppe stattfand, arbeiteten die Teilnehmer in Selbstorganisation daran, das Projektgruppenziel zu erreichen. Aspekte wie Kommunikation, Planung der Umsetzung des Projektes, gemeinsame Treffen und Aufteilung der Teilaufgaben und der Verantwortlichkeiten waren ebenfalls Teil der Aufgabe und sollten von den Teilnehmern der Projektgruppe selbstständig geregelt und durchgeführt werden.

1.3 Die Projektgruppe

Die PG 543 wird an der *TU Dortmund* vom *Lehrstuhl für Algorithm Engineering (LS11)* der *Fakultät für Informatik* durchgeführt. Der Durchführungszeitraum erstreckt sich dabei über zwei Semester, und zwar das Wintersemester 2009/10 und das Sommersemester 2010. Die Projektgruppe hat einen Umfang von 16 Semesterwochenstunden (SWS), also 8 SWS pro Semester.

1.3.1 Veranstalter

Die Veranstalter der PG sind:

- Dipl.-Inf. Fabian Gieseke / LS11
- Prof. Dr. Jan Vahrenhold / LS11
- Dr. Oliver Kramer / LS11 (nur Wintersemester 2009/10)
- Dipl.-Inf. Andreas Thom / LS11 (nur Sommersemester 2010)

Als Kooperationspartner ist der von Prof. Dr. Ralf-Jürgen Dettmar geleitete *Lehrstuhl für Astronomie* an der Ruhr-Universität Bochum beteiligt.

1.3.2 Studentische Teilnehmer

Die Projektgruppe besteht aus elf studentischen Teilnehmern mit unterschiedlichen Spezialisierungen und Vorkenntnissen:

- Hayati Bice
  Studiengang: Kerninformatik (Nebenfach: Medizin)
  Schwerpunktgebiet: 3 - Verteilte Systeme

- Fritz Boekler
  Studiengang: Kerninformatik (Nebenfach: Mathematik)
  Schwerpunktgebiet: 4 - Algorithmen, Komplexität und formale Modelle

- Tobias Brinkjost
  Studiengang: Kerninformatik (Nebenfach: Psychologie)
  Schwerpunktgebiet: 7 - Intelligente Systeme

- Georg von der Brüggen
  Studiengang: Kerninformatik (Nebenfach: Physik)
  Schwerpunktgebiet: 4 - Algorithmen, Komplexität und formale Modelle

- Davud Colakovic
  Studiengang: Kerninformatik (Nebenfach: Deutsch)
  Schwerpunktgebiet: 3 / 5 - Verteilte Systeme / Sicherheit und Verifikation

- Malte Isberner
  Studiengang: Kerninformatik (Nebenfach: Physik)
  Schwerpunktgebiet: 4 - Algorithmen, Komplexität und formale Modelle

- Martha Klois
  Studiengang: Angewandte Informatik (Anwendungsfach: Maschinenbau)

- Denis Kurz
  Studiengang: Kerninformatik (Nebenfach: Mathematik)
  Schwerpunktgebiet: 4 - Algorithmen, Komplexität und formale Modelle

- Nicolai Parlog
  Studiengang: Kerninformatik (Nebenfach: Mathematik)
  Schwerpunktgebiet: 4 - Algorithmen, Komplexität und formale Modelle

- Thomas Siwczyk
  Studiengang: Angewandte Informatik (Anwendungsfach: Logistik)

- Hendrik Spiegel
  Studiengang: Kerninformatik (Nebenfach: VWL)
  Schwerpunktgebiet: 4 - Algorithmen, Komplexität und formale Modelle

2 Seminarphase

Die Projektgruppe begann mit einer Seminarphase, zu der jeder PG-Teilnehmer die Aufgabe hatte, je einen Themenbereich aus der Astronomie und dem Data Mining durchzuarbeiten und während einer Kompaktphase Anfang Oktober vorzustellen. Die Themen wurden dabei von den Betreuern ausgewählt und bestmöglich gemäß den Wünschen der Teilnehmer verteilt. Ziel war es, der gesamten Gruppe Arbeitsweisen und Problemfelder der Astronomie näher zu bringen und einen gemeinsamen Wissensstand über die wichtigsten Data-Mining-Algorithmen zu erreichen.

2.1 Seminarthemen Data Mining

Die zum Thema Data Mining gehaltenen Vorträge dauerten jeweils etwa eine halbe Stunde und konnten deswegen natürlich nur einen Überblick über das jeweilige Thema geben. Von und für Informatikstudenten im Hauptstudium gehalten, bewegten sie sich aber dennoch nicht auf trivialem Niveau, sondern statteten die Zuhörer unter anderem mit Informationen zur Implementierung, zu Laufzeiten und zu verschiedenen Varianten aus.

Data Mining und Astronomie

In der Astronomie wächst dank moderner Teleskope die Menge der zu erfassenden, zu verarbeitenden und zu analysierenden Daten in Größe, Komplexität und Qualität exponentiell an, wobei das typische von aktuellen Teleskopen generierte Datenaufkommen bei ca. 2 Terabyte pro Nacht liegt [42]. Diese effizient zu verwalten und zu untersuchen stellt eine große Herausforderung dar, die nur mit Hilfe von interdisziplinärer Zusammenarbeit zwischen Informatikern und Astronomen bewältigt werden kann.

Mit dem Ziel, die Daten effektiv managen und durchsuchen zu können wurden sogenannte *Virtuelle Observatorien* eingerichtet, in denen die Daten archiviert und mithilfe von Datenbankabfragen durchsucht werden können. Des Weiteren bieten sie Visualisierungstools für die Daten an, die sich jedoch auf relativ einfache Bildanzeigen mit Scroll- und Suchfunktion beschränken.

Da es aber essentiell ist, Daten und Data-Mining-Ergebnisse vernünftig darzustellen, gibt es einige Visualisierungspakete. Sie bieten jedoch keine Lösung für das prinzipielle Problem der Darstellung von Daten in mehr als 3 Dimensionen und skalieren nicht gut mit großen multidimensionalen Datenmengen [58]. Dies sind zwei Kernprobleme, die bei zukünftigen *Visualisierungskonzepten* im Vordergrund stehen sollten.

Eine weitere wichtige Aufgabe der astronomischen Datenverarbeitung ist das *automatische Auffinden und Klassifizieren von Objekten* in den Aufnahmen mittels Pattern Recognition, zum Beispiel als Stern oder Galaxie. Hier führen vor allem inhomogene Datensätze, in denen Objekte mehrfach und unter verschiedenen Aufnahmebedingungen erfasst worden sind, zu Problemen. Auch das Klassifizieren der bereits identifizierten und als Datenvektor in der Datenbank abgespeicherten Objekte mittels Machine-Learning- oder Clustering-Verfahren stellt

ein wichtiges Forschungsthema dar. Objekte in *Echtzeit aufzufinden und zu klassifizieren* ist wissenschaftlich besonders interessant, da einige interessante Phänomene wie Supernovae oder Gamma-Strahlen-Ausbrüche nur sehr kurzlebig sind und es daher notwendig ist, möglichst zeitnah Folgebeobachtungen einleiten zu können.

Die Multidimensionalität der Daten führt dazu, dass interessante Strukturen auftreten können, die sich nicht auf wenige Dimensionen projizieren lassen. Diese können jedoch von *Clustering-Algorithmen* aufgespürt und untersucht werden, da die Form, Anzahl und Größe der Cluster sowie die Anzahl der enthaltenen Objekte Informationen über diese Strukturen enthalten können. Auf Grund der immensen Größe der Datenbanken sind „normale“ Data-Mining-Algorithmen oft nicht geeignet, weil eine hohe I/O-Effizienz benötigt wird.

Auch *Ausreißerdetektion* mit Hilfe von Data-Mining- und Machine-Learning-Algorithmen ist ein interessanter Ansatz, da die gefundenen Objekte potentiell seltene und damit interessante Objekte darstellen. Allerdings stellt selbst ein prozentual sehr kleiner Anteil von fehlerhaften Daten bei der Ausreißerdetektion auf Grund der immensen Größe der Datenbanken ein Problem dar.

k-Means

Der Data-Mining-Algorithmus *k-Means* gehört zu den *unüberwachten Clustering-Verfahren*, wobei k für die (vorgegebene) Anzahl der zu ermittelnden Cluster steht. Im Gegensatz zu den hierarchischen Verfahren, die agglomerativ oder diversiv arbeiten, ist k-Means ein *partitionierendes Clustering-Verfahren*. Die Daten werden iterativ in k disjunkte Teilmengen zerlegt, wobei die Entscheidung welchem Cluster ein Datum zugeordnet wird von der Entfernung zu den Zentren der bestehenden Cluster abhängt. Als Distanzmaß wird in der Praxis oft der Euklidische Abstand verwendet.

Zu Beginn des Algorithmus sind der Datensatz, die Anzahl der zu bestimmenden Cluster k und das Abstandsmaß gegeben. Des Weiteren werden k *Clusterzentren*, die sogenannten *Zentroide*, gewählt. Dies geschieht meist durch zufällige Wahl von k Datenpunkten. Die *Kosten* (welche es zu minimieren gilt) einer Zentroidenwahl hängen meist von den Abständen der Punkte zu den Clusterzentren ab.

Danach werden *Iterationen* durchgeführt, die aus folgenden Schritten bestehen:

- Jeder Punkt wird dem nächsten Zentroid zugeordnet
- Die Zentroide werden neu berechnet, indem für jedes Cluster der Schwerpunkt bestimmt wird. Diese Punkte sind die neuen Zentroide

Diese Schritte werden solange wiederholt, bis das *Abbruchkriterum* erfüllt ist. Häufig wird dies entweder durch eine feste Obergrenze für die Iterationszahl oder durch Abbruch, wenn die Zentroiden sich nicht mehr verändern, realisiert.

k-Means wird in der Praxis oft eingesetzt, da er einfach zu implementieren ist und auch bei größeren Datenmengen schnell ein Ergebnis liefert (*quick'n'dirty*), wobei im *Best Case* eine lineare Laufzeit erreicht werden kann. Allerdings kann es passieren, dass der Algorithmus in ein lokales, nicht-globales Minimum bezüglich der Kostenfunktion konvergiert. Daher werden oft mehrere Iterationen mit zufälligen Startpunkten durchgeführt und dann die Partition mit minimalen Kosten ausgewählt.

k-Nächste-Nachbarn

Mit der k-nächste-Nachbarn-Klassifikation sollen Objekte in disjunkte Klassen eingeteilt werden. Hierbei handelt es sich um eine parameterfreie Methode zur Schätzung von Wahrscheinlichkeitsdichtefunktionen. Dabei werden Objekte $x \in \mathbb{R}^n$ den einzelnen Klassen unter Berücksichtigung seiner k nächsten Nachbarn zugeordnet. Im einfachsten Fall erfolgt dies, indem x der Klasse zugeordnet wird, der die meisten bereits klassifizierten Objekte dieser k Nachbarn angehören (Mehrheitsentscheidung). Dabei wird eine bereits klassifizierte Menge von Trainingsdaten vorgegeben. Als Abstandsmaße werden z. B. der Euklidische Abstand und die Manhattan-Metrik verwendet.

Ein einfacher (aber nicht praktikabler, s.u.) Fall ist die Wahl von $k = 1$: In diesem Fall ergibt sich ein *Voronoi-Diagramm* der Punkte in der Trainingsmenge, und ein zu klassifizierender Punkt wird der Klasse des Punkts, in dessen Voronoi-Region er liegt, zugeordnet.

Die richtige Wahl von k stellt das entscheidende Problem dar: Ein zu klein gewähltes k kann zu einer schlechten Klassifikation durch Ausreißer in den Trainingsdaten führen, da diese – ungeachtet einer großen Anzahl an Punkten anderer Klassen in ihrer Umgebung – in ihrer unmittelbaren Nähe alleinig bestimmend sind. Wird k zu groß gewählt, können Objekte mit zu großem Abstand in die Klassifikationsentscheidung mit einbezogen werden. Demnach spielt auch die richtige Anzahl und die Verteilung der Trainingsdaten eine wichtige Rolle.

Wenn die Trainingsdaten nicht gleichverteilt sind, kann eine gewichtete Abstandsfunktion verwendet werden, die näheren Punkten ein höheres Gewicht zuweist als weiter entfernten. Ein praktisches Problem ist auch der Speicher- und Rechenaufwand des Algorithmus bei hochdimensionalen Räumen und einer großen Menge an Trainingsdaten.

C4.5

C4.5 [44] ist ein Algorithmus zur *Klassifizierung* von Daten, d.h. zur Einordnung von Datensätzen (Tupeln) in Klassen anhand ihrer Attributwerte. Der Algorithmus basiert auf Entscheidungsbäumen: Aus einer Menge von Daten mit bekannter Klassifikation (Trainingsmenge) wird ein Entscheidungsbaum konstruiert, anhand dessen sich die Klasse neuer Datensätze mit unbekannter Klassifikation voraussagen lässt. Die inneren Knoten des Entscheidungsbaums sind hierbei mit Attributnamen, die Blätter mit Klassennamen und die ausgehenden Kanten mit Attributwerten beschriftet.

Bei der Konstruktion des Entscheidungsbaumes aus der Trainingsmenge bedient sich C4.5 des informationstheoretischen Konzepts der *Entropie*. Diese ist ein Maß für die Unsicherheit einer Verteilung; illustrativ sind hier die beiden folgenden Extremfälle: Eine Verteilung, in der es ein sicheres und ansonsten nur unmögliche Ereignisse gibt, hat eine (minimale) Entropie von 0; eine *Gleich*verteilung mit n möglichen Ereignissen hat eine (maximale, bzgl. der Gesamtanzahl möglicher Ereignisse) Entropie von $\log n$. Als Verteilung werden bei C4.5 die relativen Häufigkeiten der jeweiligen Attributwerte/Klassen in der Trainingsmenge herangezogen.

Das Vorgehen von C4.5 ist bemerkenswert intuitiv. Die Konstruktion des Entscheidungsbaumes findet *top-down* statt. An jeder Stelle, an der ein Knoten eingefügt werden soll, betrachtet man nur noch die Teilmenge der Trainingsdaten, welche dem zu dieser Stelle führenden Pfad genügen. Die Entscheidung über die (Attribut-)Beschriftung des neuen Knotens wird danach getroffen, welche Wahl die *erwartete* (bzgl. der relativen Häufigkeiten der Attributwerte) *Unsicherheit/Entropie* (bzgl. der Klassifikation) am meisten reduziert (Informationsgewinn, *information gain*). Um eine Dominanz von Attributen mit vielen möglichen Ausprägungen zu

vermeiden – je mehr mögliche Ausprägungen, desto größer die Unsicherheit –, wird der *information gain* zur Korrektur durch die Entropie bzgl. der Ausprägung des jeweiligen Attributs dividiert. Der sich ergebende Wert heißt *gain ratio*.

Neben diesem Grundalgorithmus gibt es noch weitere Aspekte, die eine Beschäftigung wert sind [59]; bspw. das Beschneiden (*pruning*) der Bäume, um das Problem des *overfitting* (zu starke Anpassung des Klassifikators auf die Trainingsmenge) zu mindern; oder auch die Anwendung von C4.5 auf extern gespeicherten Datenmengen.

Dichtebasiertes Clustering

Beim Clustering (engl. *cluster*: Gruppe, Anhäufung) geht es um die Identifikation einer endlichen Menge von Kategorien, Klassen oder Gruppen. Dabei soll gelten, dass Objekte im gleichen Cluster möglichst ähnlich und Objekte aus verschiedenen Clustern hingegen möglichst unähnlich zueinander sind. Die einzelnen Cluster haben eine unterschiedliche Größe, Form und Dichte.

Es wird zwischen hierachischen, partitionierenden und dichtebasierten Clusteringverfahren, sowie Graph-theoretischen Verfahren, neuronalen Netzen und *Fuzzy Clustering* unterschieden. Ein Beispiel für Clusteringverfahren sind die Algorithmen *DBSCAN* (dichtebasiert) und *CLARANS* (partitionierend).

Clustering-Algorithmen sind besonders gut geeignet um Klassen in räumlichen Datenbanken zu identifizieren. Daher richtet sich das Augenmerk der Projektgruppe auf solche Verfahren, um sie nach Möglichkeit angepasst einsetzen zu können.

Bei räumlichen Datenbanken werden folgende Anforderungen an die Clustering-Algorithmen gestellt: Ein Minimum an Wissen über die Domäne, um Eingabeparameter bestimmen zu können, die Entdeckung von Clustern beliebiger Form und Effizienz bei der Arbeit auf großen Datenbanken. Bereits bekannte Clustering-Algorithmen bieten jedoch keine vernünftige Lösung, wenn die Kombination der o.a. Anforderungen vorausgesetzt wird.

Aus diesem Grund wurde mit DBSCAN [29] ein neuer Algorithmus entwickelt, welcher sich für ähnliche Problemstellungen wie der bekannte Algorithmus CLARANS einsetzen lässt, dabei aber sehr viel effizienter arbeitet. Außerdem benötigt DBSCAN nur einen Eingabeparameter, wobei der Benutzer bei dessen Bestimmung unterstützt wird.

Naive Bayes

Naive Bayes [59] ist ein Algorithmus zur Klassifikation von Objekten, der – wie der Name schon sagt – auf der Anwendung des Satz von Bayes beruht. Seine leichte Implementierbarkeit und Nachvollziehbarkeit haben ihm den Spitznamen „Idiot's Bayes“ eingebracht, was jedoch nicht darüber hinwegtäuschen sollte, dass er in vielen Fällen gute Ergebnisse liefert. Zudem ist er in der Ausführung relativ schnell.

Der Algorithmus konstruiert einen Schätzer $\hat{P}$, welcher die Wahrscheinlichkeit einer bestimmten Klassenzugehörigkeit unter der Bedingung des Vorliegens der konkreten Attributausprägungen abschätzt. Ist der geschätzte Wert größer als ein Schwellenwert $\delta \in (0, 1)$, so wird das Objekt der entsprechenden Klasse zugeordnet.

Technisch gesehen betrachtet man die Klassenzugehörigkeit eines Objekts als Zufallsvariable I (Wertebereich: Menge der Klassen K, zur Vereinfachung $K = \{0, 1\}$) und dessen Attributausprägungen als Zufallsvariable X (Wertebereich: Menge der möglichen k-dimensionalen Ausprägungsvektoren V). Die Wahrscheinlichkeit, dass ein Objekt mit Wertevektor $v \in V$

zur Klasse $i \in K$ gehört, ist $P(I = i \mid X = v)$, was unter Verwendung des **Satz von Bayes** geschrieben werden kann als

$$P(I = i \mid X = v) = \frac{P(X = v \mid I = i)P(I = i)}{P(X = v)}.$$

Um den Term $P(X = v)$ schätzen zu können, wendet man den **Satz von der totalen Wahrscheinlichkeit** an, womit sich ergibt:

$$P(X = v) = P(X = v \mid I = 0)P(I = 0) + P(X = v \mid I = 1)P(I = 1).$$

Es müssen also lediglich die Werte für $P(I = j)$ und $P(X = v \mid I = j)$ geschätzt werden. Ersteres erfolgt durch die relativen Anteile in einer betrachteten Menge von Stichproben (Trainingsmenge), für letzteres bedient man sich einer Vereinfachung: Man nimmt *Unabhängigkeit* der k Komponenten des Zufallsvektors $X = (X_1, \ldots, X_k)$ an. Dann ergibt sich:

$$P(X = v \mid I = j) = \prod_{l=1}^{k} P(X_l = v_l \mid I = j).$$

Die einzelnen Faktoren lassen sich jetzt wieder leicht durch Betrachtung von Stichproben schätzen.[1]

Die angenommene Unabhängigkeit ist natürlich nicht immer (bzw., in den seltensten Fällen) gegeben. Aus diesem Grund gibt es mit der sog. *Schrumpf-Faktor-Methode* eine Vorgehensweise, die Effekte von fälschlich angenommener Unabhängigkeit abzuschwächen. Neben dieser Erweiterung existieren noch einige andere, welche allerdings – neben einer eventuellen Verschlechterung der Performanz – den Algorithmus seiner bestechenden Einfachheit berauben.

Support Vector Machines

Support Vector Machines werden dazu verwendet, Punkte des $\mathbb{R}^d$ mit Hilfe einer trennenden Hyperebene zu klassifizieren. In einem Vorverarbeitungsschritt wird zu einer Trainingsmenge eine Hyperebene bestimmt, die den maximalen Abstand zu beiden Klassen maximiert. Zur eigentlichen Klassifizierung muss dann nur noch überprüft werden, auf welcher Seite der Hyperebene der zu klassifizierende Punkt liegt. Durch den maximalen Abstand soll eine möglichst fehlerfreie Klassifizierung neuer Punkte erreicht werden.

Sind die zu klassifizierenden Objekte keine Punkte des $\mathbb{R}^d$ oder nicht durch eine Hyperebene trennbar, wird der sogenannte *Kernel-Trick* verwendet. Ein Kernel ist eine Abbildung, die die Objekte passend in den $\mathbb{R}^d$ abbildet, der *Merkmalsraum* genannt wird. Sowohl die Berechnung der trennenden Hyperebene als auch die Klassifizierung können dann in diesem Merkmalsraum durchgeführt werden. Die Transformation vom Eingabe- in den Merkmalsraum kann dabei implizit geschehen. Dieser Umstand wird als Kernel-Trick bezeichnet.

Wenn die Objekte auch mit Hilfe des Kernel-Tricks nicht durch eine Hyperebene trennbar sind oder man von einem „Rauschen“ in der Trainingsmenge ausgehen kann, können bei Berechnung der Hyperebene auch *Schlupfvariablen* benutzt werden. Diese erlauben zwar, dass einzelne Punkte der Trainingsmenge falsch klassifiziert werden. Jeder falsch klassizierte Punkt wirkt sich allerdings negativ auf die Bewertung der Hyperebene aus.

[1] Dies ist natürlich nur bei diskreten Attributen sinnvoll. Bei kontinuierlichen Attributen behilft man sich mit der – häufig anzutreffenden – Vorgehensweise, sie durch Einführung von „vernünftigen“ Schwellenwerten wie diskrete Attribute zu behandeln.

PageRank

Bei *PageRank* handelt es sich um ein iteratives Verfahren, um eine Sammlung verlinkter Objekte anhand ihrer Stellung im Gesamtverbund zu bewerten.

Das Verfahren stammt aus dem Bereich der Websuche und wurde an der Universität Stanford von Larry Page und Sergej Brin entwickelt.

PageRank betrachtet das WWW als einen Graphen, dessen Knoten die Seiten und dessen Kanten die Links zwischen den Seiten darstellen. Die grundlegende Idee besteht darin, eingehende Kanten eines Knotens als eine gewichtete Stimme für diesen zu werten. Die Summe der eingehenden Stimmgewichte ist dann der *PageRank* des Knotens bzw. der Seite.

Das Gewicht einer ausgehende Kante ist der *PageRank* des Elter-Knotens geteilt durch die Zahl seiner ausgehenden Kanten. Auf diese Weise ist auch sichergestellt, dass der aufsummierte *PageRank* aller Knoten konstant bleibt.

Eine weitere gängige Analogie für das *PageRank*-Verfahren ist das sog. *Zufallssurfer*-Modell. Hierbei wird ein (fiktiver) Benutzer angenommen, der zufällig durch das Internet surft und mit einer bestimmten Übergangswahrscheinlichkeit, beeinflußt durch die ausgehenden Links einer Seite, eine beliebige andere Seite im Internet ansteuert. Dieses Modell impliziert eine Betrachtung des *PageRank*-Verfahrens als Markovkette.

Die Berechnung des *PageRank* erfolgt iterativ. Zunächst werden alle Knoten mit dem gleichen Wert initialisiert, dann erfolgt schrittweise die Bestimmung des *PageRanks*. Die Berechnung bricht ab, wenn die Differenz zwischen den Ergebnissen der i-ten und $i-1$-ten Runde einen vorgegebenen Schwellenwert unterschreitet.

k-d-Bäume und Ausreißerdetektion

Ein *k-d-Baum* (ein k-dimensionaler Baum) ist ein binärer Suchbaum zur Darstellung von Punkten in einem k-dimensionalen Raum. Der Algorithmus zur Erstellung eines solchen Baumes teilt die Punktmenge rekursiv auf jeder Tiefe anhand einer anderen Dimension auf. Dabei wird als Trennungspunkt jeweils der *Median* der betrachteten Dimension verwendet.

Eine typische Operation für eine solche Datenstruktur ist eine *Bereichsabfrage*, deren Ziel es ist, alle Punkte in einem vorgegebenen Suchbereich auszugeben. Hierzu wird der Baum *top-down* durchlaufen. Beim Durchlaufen des Baumes gibt es drei Fälle zu beachten:

- Der durch den Knoten repräsentierte Bereich ist ganz im Suchbereich enthalten
  $\Rightarrow$ Alle durch diesen Knoten repräsentierten Punkte werden ausgegeben
- Der durch den Knoten repräsentierte Bereich liegt außerhalb des Suchbereiches
  $\Rightarrow$ Suche für diesen Knoten abbrechen
- Der durch den Knoten repräsentierte Bereich schneidet den Suchbereich
  $\Rightarrow$ Rekursiver Aufruf der Bereichsabfrage für die Kindknoten

Bei der *Ausreißerdetektion* mittels eines k-d-Baumes [28] wird nach untypischen oder seltenen Objekten innerhalb des Baumes gesucht. Dazu versucht man Gruppen von Punkten zu finden, die ein ähnliches *Ausreißerverhalten* vorweisen, welches sich z.B. anhand der spärlich besetzten Region um ein Objekt messen lässt.

Allerdings ist die zuvor beschriebene *pure* Medianteilung des Raumes für die Ausreißerdetektion ungeeignet und es werden sog. *Special Cuts* eingeführt. Diese Heuristiken sollen, im Falle einer ungleichmäßigen Verteilung der Punkte im Raum, den jeweiligen Cluster von den

Ausreißern trennen. Abschließend findet eine Bewertung der einzelnen Regionen anhand des Inversen der Dichte $\frac{\#Objekte}{Volumen}^{-1}$ statt.

Insgesamt lässt sich dieses Verfahren in linearer Laufzeit bei linearem Platzbedarf ausführen und bietet dennoch, im Vergleich zu „normalen“ Clusteringalgorithmen, eine gute Ausgabequalität.

BSP-Bäume

Ein BSP-Baum ist eine Datenstruktur für eine binäre Unterteilung des Raums (eine *Binary Space Partition*). Diese enthält zu einer Menge beliebiger Objekte im $\mathbb{R}^d$ eine Menge $\mathbb{R}^{d-1}$-dimensionaler Hyperebenen, die den $\mathbb{R}^d$ so unterteilt, dass je zwei Objekte durch mindestens eine Ebene voneinander getrennt werden.

Eine solche Menge kann als binärer Baum modelliert werden. Dazu interpretiert man die Wurzel als Repräsentation des gesamten Raumes und weißt ihr eine Hyperebene zu (diese wird im Knoten gespeichert). Die beiden durch die Ebene definierten Halbräume werden dann durch die Kinder der Wurzel repräsentiert und auf diesen wird rekursiv mit der Zuweisung weiterer Hyperbenen und erneuten Teilungen des Raumes fortgefahren. Die so entstehenden Teilräume sind konvex und werden von je einem inneren Knoten repräsentiert. Das Prozedere endet, wenn ein Teilraum nur noch ein Objekt enthält, welches dann in einem Blatt gespeichert wird.

Wenn bei diesem Vorgehen ein Objekt geteilt wird, werden seine Teile im Folgenden als zwei getrennte Objekte behandelt. Deswegen ist es vernünftig, die Größe eines BSP-Baumes als die abschließende Anzahl der Objekte zu definieren und zu versuchen, diese zu minimieren.

Die Anzahl der Objekte einer BSP hängt offensichtlich von der Wahl der Hyperebenen ab, so dass diese Auswahl das zentrale Element bei der Konstruktion eines möglichst kleinen BSP-Baumes ist. Für die Wahl der Hyperebenen existieren verschiedene Techniken, von denen allerdings viele – insbesondere für $d > 3$ – bis jetzt nicht ordentlich analysiert wurden, d.h. es existieren keine Beweise zur Laufzeit der Berechnung oder Größe des erzeugten Baumes.

Als Beispiel wird hier eine einfache Auswahlmethode beschrieben, die ordentlich analysiert ist. Um dieses Verfahren zu wählen, müssen die Objekte aber eine besondere Eigenschaft haben: Es muss sich um beschränkte Ausschnitte von $\mathbb{R}^{d-1}$-dimensionalen Hyperebenenen handeln. Eine *Auto-Partition* verwendet dann ausschließlich durch die Objekte definierte Hyperebenen zur Teilung des Raumes. Dieses Verfahren ist offensichtlich eingeschränkter als die Wahl beliebiger Ebenen, aber das ermöglicht zumindest im $\mathbb{R}^2$ eine Untersuchung von Laufzeit ($O(n^2 \log n)$) und Größe (im Erwartungswert $O(n \log n)$) und stellt in der Praxis keinen entscheidenden Nachteil dar.

Kompliziertere Methoden arbeiten z.B. mit Segmentbäumen oder Quadtrees, wobei letztere zusammen mit der *Dichte* δ einer Objektmenge eine parametrisierte Laufzeitanalyse ermöglichen.

I/O-effiziente Algorithmen und Datenstrukturen

Datenvolumina in der Astronomie erreichen oft Größenordnungen von Giga- und Terabytes. Deshalb ist bei der Bearbeitung oder Analyse solcher Daten – beispielsweise durch Data-Mining-Algorithmen – die Verwendung *I/O-effizienter Algorithmen und Datenstrukturen* unumgänglich.

Bei der Betrachtung und Analyse dieser Algorithmen und Datenstrukturen wird das *I/O-Modell* von Shriver und Vitter (1994) [39] zugrunde gelegt. Dieses besteht im Wesentlichen

aus drei Komponenten:

- CPU
- Interner Speicher (Größe M, schnell)
- Externer Speicher (D Festplatten, langsam)

Entscheidend für die *I/O-Komplexität* eines Algorithmus oder einer Datenstruktur sind die ausgetauschten Blöcke zwischen internem und externem Speicher. Dieses System lässt sich ebenfalls zwischen CPU und internem Speicher ansetzen (Cache – Interner Speicher).

Weiter gehören zur I/O-Komplexität die interne Laufzeit (sollte vergleichbar mit internen Algorithmen sein) und die benötigten Ressourcen des externen Speichers.

Im Vortrag wurden folgende Beispiele bearbeitet:

- Algorithmen: *Suffix Arrays*
- Datenstrukturen: *Stacks*, *Queues*, *Externer Mergesort*

2.2 Seminarthemen Astronomie

Bis auf wenige Teilnehmer, die durch das Nebenfach Physik oder privates Interesse Wissen aus der Astronomie mitbrachten, besaß die Projektgruppe keine astronomischen Vorkenntnisse und dementsprechend einfach waren die Vorträge aus diesem Bereich gehalten. Sie dauerten jeweils eine Viertelstunde und waren nur dazu gedacht, einen kurzen Einblick in einige Themen und Verfahren der Astronomie zu gewähren, und so zumindest eine Vorstellung vom Zusammenspiel mit der Informatik zu ermöglichen. Hierzu wurden im Rahmen des SDSS-Projekts bereitgestellte Tutorials [52] genutzt.

Sonnensystem und Koordinaten

Das Sonnensystem umfasst das gesamte gravitative Feld der Sonne inklusive den 8 Planeten und ihren Satelliten, allen Kometen, Asteroiden und Meteoriden. Die Planeten umkreisen die Sonne auf meist elliptischen Orbiten.

Die Bestimmung der Position von Himmelskörpern erfolgt anhand der *Rektaszension* (engl. Right Ascension, RA) und der *Deklination* (engl. Declination, Dec). Ihre irdischen Gegenparts sind der Längengrad und der Breitengrad. Rektaszension bezeichnet den Winkel zwischen dem von Nord- zu Südpol verlaufenden Längenkreises durch den Frühlingspunkt und dem Längenkreis über dem der Himmelskörper steht, wenn man den Himmel als Sphäre betrachtet. Als Deklination wird dann der Winkelabstand zwischen dem Himmelsäquator und dem Objekt bezeichnet.

Durch die Rotation der Erde bewegen sich zum einen die Sterne und zum anderen die Sphäre mit ihren Koordinatenlinien. Da sich aber beide im gleichen Verhältnis bewegen, bleiben die Koordinaten der Sterne konstant und jeder erscheint jede Nacht an der selben Stelle der Sphäre.

Scavengerhunt

In diesem Thema wird sowohl kurz auf die einzelnen Objekte im Universum und ihre Eigenschaften eingegangen, als auch auf die zu diesen im SDSS-Katalog enthaltenen Informationen.

Farben: Hervorzuheben bei den angegebenen Farbinformationen im SDSS-Katalog ist, dass höhere numerische Werte zu den einzelnen Objekten so zu interpretieren sind, dass diese für den Betrachter dunkler erscheinen. Die Helligkeitswerte werden in der Regel im Verhältnis von zwei Farbwerten zueinander betrachtet. Im SDSS-Katalog sind zu den einzelnen Objekten fünf Farbwerte gespeichert.

Spektren: Als die aussagekräftigsten Informationsquellen des SDSS besitzen Spektren eine besondere Relevanz in der Analyse von Objekten. Von besonderer Bedeutung in diesem Zusammenhang ist, dass jedes Objekt auf jeder Wellenlänge in charakteristischer Stärke strahlt, womit das Spektrum eine Art individuellen *Fingerabdruck* für das jeweilige Objekt darstellt.

Sterne: Sterne sind riesige Bälle aus Gas, welche in ihren Kernen Wasserstoff verbrennen (bzw. fusionieren). Die Fusionsprodukte können zwar wiederum zu anderen Elementen fusioniert werden; da die Energiebilanz der Kernfusion jedoch nur bei den leichten Elementen positiv ist, ist die Lebensdauer eines Sterns begrenzt.

Sternhaufen: Unter einem Sternhaufen versteht man die örtliche Anhäufung einer größeren Anzahl an Sternen.

Galaxien: Galaxien sind eine Ansammlung von einer großen Menge an Sternen, welche durch Gravitationskräfte zusammengehalten werden. Hierbei sei angemerkt, dass Galaxien deutlich größer als Sternhaufen sind; Sternhaufen sind oft in Galaxien enthalten. Als Beispiel sei unsere Milchstrasse mit Ihren ca. 1 Billionen Sternen genannt.

Quasare: Quasare sind die am weitesten entferten Galaxien mit sehr aktiven Zentren. Optisch erscheinen sie punktförmig, also wie Sterne.

Asteroiden: Asteroiden sind kleine Gesteinsbrocken innerhalb des Sonnensystems. Auf Grund ihrer im Verhältnis zum irdischen Betrachter hohen Geschwindigkeit erscheinen sie auf Aufnahmen häufig als Linien.

Universum

Der Begriff „Universum“ bezeichnet die Gesamtheit aller Dinge. Die deutsche Bezeichnung *Weltall* wird seit dem 17. Jahrhundert verwendet.[2] Das Universum umfasst unter anderem die Galaxien, Sterne und Planeten, sowie alle andere Materie.

In der Astronomie werden Entfernungen in Lichtjahren ($9{,}5 \cdot 10^{12}\,\mathrm{km}$) gemessen. Durch den Einsatz modernster Teleskope können heutzutage Objekte beobachtet werden, die über 10 Milliarden Lichtjahre entfernt sind.

[2] Zu beachten ist, dass der Begriff Welt*raum* nur den Raum außerhalb der Erdatmosphäre bezeichnet.

Expansion und Rotverschiebung

Gemäß dem aktuellen Standardmodell der Kosmologie expandiert unser Universum seit dem Urknall vor ca. 13,7 Milliarden Jahren. Expansion bedeutet in diesem Kontext, dass alle Objekte, so sie nicht durch Gravitation zusammengehalten werden, immer weiter auseinander driften, da sich das Universum in seiner Gesamtheit ausdehnt. Die relative Bewegung zweier Objekte zueinander bleibt davon unberührt.

Ein wichtiges Phänomen, welches dadurch verursacht wird, ist die so genannte „kosmologische Rotverschiebung“. Damit bezeichnet man den Effekt, dass sich die Wellenlänge von elektromagnetischen Wellen über die Zeit vergrößert, wenn diese sich in einem sich ausdehnenden Raum bewegen.[3]

In den zwanziger Jahren des letzten Jahrhunderts entdeckte Edwin Hubble diese Zusammenhänge und entwickelte im Laufe seiner Forschungen das nach ihm benannte *Hubble-Diagramm*, welches den linearen Zusammenhang von Entfernung und Röte des Lichts einer Galaxie aufzeigt.

Ursprünglich wurde die Rotverschiebung durch Vergleiche von Helligkeit und Größe verschiedener Objekte miteinander bestimmt. Heutzutage verwenden Forscher dazu vorwiegend Spektren der einzelnen Objekte, da diese einen umfassenden Blick auf das von einem Objekt emittierte Strahlungsspektrum erlauben.

Da Galaxien und Sterne in Abhängigkeit ihrer Zusammensetzung auf charakteristischen Frequenzen besonders stark oder schwach strahlen (diese erschienen als Linien im Diagramm und heißen deswegen *Spektrallinien*) und sich diese Objekte untereinander recht ähnlich sind, kann man aus der Verschiebung der Spektrallinien die Rotverschiebung ermitteln.

Asteroiden

Asteroiden, auch Kleinplaneten genannt, sind kleine, sich bewegende Gesteinsbrocken mit einem zu Planeten vergleichsweise sehr geringen Durchmesser. Dieser beträgt zwar bis zu 900 km (Ceres, größter bekannter Asteroid), allerdings sind Asteroiden mit einem Durchmesser von über 200 km sehr selten. Asteroiden stammen vermutlich aus der Entstehungszeit des Sonnensystems und bewegen sich seither in festen Umlaufbahnen um die Sonne. Die meisten befinden sich im Asteroidengürtel zwischen Mars und Jupiter.

Bedingt durch das Aufnahmeverfahren (Teilaufnahmen werden sequentiell nacheinander gemacht und später übereinander gelegt) und die Anordnung der Farbfilter der SDSS-Teleskope erscheinen Asteroiden auf den entsprechenden Aufnahmen als gelbe und blaue Punkte oder als rote und grüne Linien – wobei die Erscheinungsform davon abhängt mit welcher (Winkel-)Geschwindigkeit relativ zum Teleskop der Asteroid sich bewegt.

Klassifizierung von Sternen

Das *Spektrum* eines Sterns, eine Darstellung der Intensität der Strahlung in Abhängigkeit von der Wellenlänge, enthält zwei Merkmale, die in jedem Spektrum vorhanden sind und mit deren Hilfe Sterne klassifiziert werden können:

- Das *Intensitätsmaximum* der abgestrahlten Energie liegt bei einer Wellenlänge, die umgekehrt proportional zur Temperatur des Sterns ist

[3] Im Unterschied zu einer durch einen Dopplereffekt ausgelösten Rotverschiebung, bei der sich Objekte relativ zueinander bewegen.

- *Absorptionslinien* sind schmale Täler, die im Spektrum des Sterns auftreten. Sie entstehen, wenn gebundene Elektronen Energie absorbieren. Da sich diese nur auf festen diskreten Energieniveaus befinden können, kann nur Strahlung bestimmter Wellenlängen (nämlich solcher, bei denen die Energie eines Photons der Energiedifferenz zweier Niveaus entspricht) aufgenommen werden. Aus dem Vorhandensein und der Stärke der Absorptionslinie lässt sich daher ermitteln, welche Elemente in welcher Konzentration im Stern vorhanden sind

Zunächst wurde ein Klassifizierungssystem angewendet, welches die Sterne nach der Stärke der ersten Absorptionslinie des Wasserstoffatoms, der sogenannten H-α Linie, absteigend in die mit Buchstaben gekennzeichneten sogenannten *Spektralklassen* sortiert. Nach der Eliminierung unnötiger Klassen, die meist durch Messfehler entstanden, blieben die Spektralklassen **A**, **B**, **F**, **G**, **K**, **M** und **O** übrig.

Da die Klassifizierung nach der Temperatur des Sterns naheliegender ist als nach der Stärke der H-α Linie und die bisherigen Klassifikationen weiterhin gültig bleiben sollten, wurden die Spektralklassen in der heute üblichen Reihenfolge **O B A F G K M** angeordnet, wobei O die heißesten und M die kühlsten Sterne bezeichnet.

Farben

Farben sind eine Eigenschaft des sichtbaren Lichts, welches wiederum ein Teil der elektromagnetischen Strahlung eines Objektes (z.B. eines Sterns) ist. Diese Strahlung hat Wellencharakter und die bestimmende Eigenschaft eines Strahls ist neben seiner Intensität seine Wellenlänge. Im Falle des für den Menschen sichtbaren Lichts liegen diese zwischen 380 und 780 nm. Die unterschiedlichen Farben entsprechen der Wahrnehmung verschiedener Wellenlängenbereiche (z.B. Blau bei $\approx 470\,\text{nm}$, Rot bei $\approx 610\,\text{nm}$). Die Intensität wird in Magnituden gemessen, wobei aus historischen Gründen eine geringe Strahlung einen höheren Wert erhält. (Sterne bis zu 6 mag sind auf dem Land mit bloßem Auge am Nachthimmel erkennbar; die Sonne hat $-26,8\,\text{mag}$.)

Durch verschiedene Methoden ist es möglich, die Strahlungsintensität eines Lichtstrahls in verschiedenen Wellenlängenbereichen zu messen. Selbstverständlich ist dies umso aufwändiger, je detaillierter das Ergebnis sein soll. Im SDSS-Katalog liegen ausführliche Spektren nur für etwa 4000 der 6 Millionen erfassten Sterne vor. Für alle weiteren existieren lediglich schnelle Aufnahmen, die die Intensität nur an 5 festgelegten Positionen des Spektrums messen:

$$\textbf{u:} \approx 350\,\text{nm},\ \textbf{g:} \approx 475\,\text{nm},\ \textbf{r:} \approx 625\,\text{nm},\ \textbf{i:} \approx 770\,\text{nm},\ \textbf{z:} \approx 920\,\text{nm}$$

Aber selbst diese groben Messungen lassen sich zur Identifikation interessanter Sterne nutzen. Dazu muss allerdings etwas zur Entstehung der Strahlung gesagt werden.

Elektromagnetische Strahlung entsteht durch die Bewegung der Atome. Dabei gilt: je schneller die Bewegungen sind, desto mehr Energie hat die Strahlung und desto kürzer ist ihre Wellenlänge. Wird die Strahlung eines Körpers ausschließlich durch seine Wärme erzeugt, spricht man von Wärmestrahlung. Diese beschränkt sich aber nicht nur auf eine Wellenlänge, sondern ist – dem Planckschen Strahlungsgesetz folgend – auf das gesamte Spektrum verteilt:

$$I_T(\lambda) = \frac{2hc^2}{\lambda^5 \cdot (e^{\frac{hc}{\lambda kT}} - 1)}$$

Allerdings strahlen Sterne auch aus anderen Gründen Licht ab und sind somit keine perfekten Quellen für Wärmestrahlung.

Eine Einschätzung, ob ein Stern nur Wärmestrahlung abgibt, ist mit Hilfe von zweidimensionalen Farb-Farb-Diagrammen möglich. Die beiden Achsen dieses Diagramms werden mit je einer Differenz (z.B. $g-r$ und $r-i$) beschriftet. Jeder Stern wird anschließend als Datenpunkt in dieses Diagramm eingetragen. Wegen der Regelmäßigkeit der Wärmestrahlung gibt es in dem erhaltenden Diagramm einen Bereich, in dem Sterne liegen, deren Strahlung eben jener gleicht. Sterne außerhalb dieses Bereiches strahlen noch aus anderen Gründen und sind deswegen für die Astronomie von besonderem Interesse, so dass von ihnen häufig ein detailliertes Spektrum angefertigt wird.

Galaxien

In der Astronomie wird eine Galaxie (altgr.: galaxías, „Milchstraße“) als eine gewaltige, dichte Ansammlung von Materie wie Sternen, Sternhaufen und Planetensystemen verstanden, die durch gegenseitige Gravitation dicht beieinander bleibt. Den Großteil der Masse einer Galaxie machen jedoch Staubwolken, Gas, Gasnebeln und dunkle Materie aus. Zwar besitzen verschiedene Galaxien nur selten die gleiche Größe oder Morphologie aber sie lassen sich dennoch als Spiralgalaxien, elliptische, lentikuläre oder irreguläre Galaxien klassifizieren, wobei sich die einzelnen Galaxietypen wiederum ggf. in Unterklassen gliedern, was im Folgenden beschrieben wird:

Spiralgalaxien: Die erste Klasse von Galaxien ist die der Spiralgalaxien, welche die Unterklassen Sa, Sb und Sc besitzt. Die Unterklassen gruppieren die Spiralgalaxien aufgrund ihrer verschiedenen Spiralarme. Dabei beschreibt die Unterklasse Sa die eng anliegenden Arme und einen ausgeprägten sphäroidischen Kern – den sogenannten Bulge – und Sb dicht anliegende Spiralarme und einen mittleren galaktischen Kern. Die Unterklasse Sc der Spiralgalaxien beschreibt hingegen einen relativ schwachen galaktischen Kern und äußerst locker gewundene Spiralarme, welche fast die Gestalt eines in sich verschlungenen „S“ haben.

Elliptische Galaxien: Die Klasse der elliptischen Galaxien wird in die Unterklassen E0 (kreisförmig) bis E7 (stark elliptisch) eingeteilt – dies geschieht nach ihrer *numerischen Exzentrizität*. Dabei handelt es sich um ein Maß für die Abweichung eines Kegelschnittes von der Kreisform welche auf den Umriss der Galaxie angewandt wird.

Lentikuläre Galaxien: Lentikuläre (linsenförmige) Galaxien gehören der Klasse S0 an, da sie einen Kern haben, jedoch keine Spiralarme.

Irreguläre Galaxien: Die letzte Klasse ist die der irregulären (unregelmäßigen) Galaxien, welche keine Unterklassen besitzt. Diese haben weder Spiralarme noch eine elliptische Form, zudem sind sie im Mittel leuchtschwächer als elliptische und spiralförmige Galaxien.

Spektralklassen

Spektralklassen bilden ein Klassifikationsschema für Sterne, welches sich an deren Spektrum orientiert. Anders als bei der Klassifizierung von Sternen (siehe Seite 20) beschrieben schaut man sich hier jedoch weder das Intensitätsmaximum oder die Stärke bestimmter Absorptionslinien an (dies ist eventuell sowieso nicht möglich, falls die Sterne so kalt bzw. heiß sind, dass

das Intensitätsmaximum außerhalb des beobachteten Wellenlängenbereiches liegt), sondern vielmehr das *Muster* im Auftreten der Absorptionslinien.

Auch hier erfolgt wieder eine Zuordnung zu den bereits erwähnten Klassen **O**, **B**, **A**, **F**, **G**, **K** und **M** sowie, dank mittlerweile verbesserter Teleskope, noch zu einigen neuen Klassen. Das Spektrum von Sternen einer Klasse weist hierbei die Charakteristik des Absorptionsspektrums gewisser Elemente auf; so gleicht etwa das Spektrum von Sternen der Klasse **O** dem Spektrum von ionisiertem Helium (He-II).

Sky Surveys

Die in diesem Beitrag verwendeten Informationen stammen aus dem Sky Survey Online Tutorial, welches auf der SDSS-Webseite zu finden ist. (Vgl. [16])

Der bereits im Vorwort dieses Berichts erwähnte Sloan Digital Sky Survey ist nicht der erste Versuch, den Himmel zu kartografieren. Diese reichen bis ins Jahr 350 v.Chr. zurück [17]. Erste moderne Anstrengungen wurden in den fünfziger Jahren vom *Palomar Observartory* unternommen: der *Palomar Observatory Sky Survey* (POSS I), gefolgt vom *Second Palomar Observatory Sky Survey* (POSS II) in den achtziger Jahren.

Die unterschiedlichen Sky Surveys liefern Himmelskarten, auf denen das Licht in verschiedenen Wellenlängen und in unterschiedlichen Himmelsabschnitten aufgenommen wurde. Um diese auszuwerten und somit logische Schlussfolgerungen ziehen zu können, sind bereits verschiedene Software-Tools entwickelt worden, unter anderem auch das *SDSS Navigation Tool*, welches es erlaubt, die Daten des SDSS-Katalogs zu betrachten und zu evaluieren.

Da die Aufnahmen der Sky Surveys zu verschiedenen Zeitpunkten stattfanden, bieten sich Vergleiche zwischen den zeitlich verschiedenen Aufnahmen an, um Phänomene wie Supernovae zu entdecken.

Der im SDSS betrachtete Wellenlängenbereich deckt den Bereich des sichtbaren Lichts sowie infrarote und ultraviolette Strahlung ab. Andere Surveys betrachten oft einen eingeschränkteren Wellenlängenbereich, bspw. der Two Micron All Sky Survey (2MASS), ein Survey speziell für infrarote Strahlung; oder der Röntgenstrahlen messende ROSAT PSPCC Survey.

Der SDSS ist also einer von vielen bekannten Sky Surveys. Neben neuen Erkenntnissen, die durch separates Evaluieren einzelner Surveys errungen wurden bzw. noch zu erringen sind, entsteht durch den Vergleich der Daten verschiedener Surveys eine erhebliche Menge an neuem Wissen.

Quasare

Quasare sind die am weitesten entfernten Objekte im Universum, welche noch beobachtbar sind. Dies ist auf ihre enorme Strahlungsintensität zurückzuführen. Energie für diese Strahlung liefern wahrscheinlich sog. Supermassive Schwarze Löcher (*supermassive black holes, SMBH*). Diese werden von einer Wolke aus Gas umkreist, welches durch die enorme Gravitationswirkung beschleunigt und somit erhitzt wird, was zur Strahlungsemission führt.

Aufgrund der großen Entfernung erscheinen Quasare auch durch moderne Teleskope betrachtet punktförmig, d.h. wie Sterne. Der erste Quasar wurde durch seine Strahlung im Radiobereich entdeckt, ein Bereich des elektromagnetischen Spektrums, in dem Sterne normalerweise nur sehr schwach strahlen. Darauf geht auch der Name „Quasar“ zurück: **Qua***si*-**s***tell***ar** **r***adio source*, etwa „sternartige Radioquelle“. Dieser Name ist allerdings nicht ganz korrekt,

da die überwiegende Anzahl der bereits gefundenen Quasare nicht im Radiobereich strahlt. Deswegen wird auch der Name *QSO* (*quasi-stellar object*) verwendet.

Zum Auffinden von Quasaren gibt es zwei mögliche Vorgehensweisen. Ausgangspunkt sind jeweils Objekte, welche optisch als Sterne erscheinen.

- Man untersucht diese Objekte auf Radiostrahlung. Da SDSS nur Aufnahmen im Bereich des sichtbaren Lichts (plus Infrarot und Ultraviolett) macht, liefert es also keine Daten über das Radiospektrum. Hierzu greift man auf andere Surveys wie bspw. FIRST zurück.
- Man analysiert das Spektrum der Objekte, um deren Rotverschiebung zu berechnen. Objekte mit einer hohen Rotverschiebung (also Entfernung) können keine Sterne sein.

Beide Methoden haben Nachteile, da bei der einen nur im Radiobereich strahlende Quasare gefunden werden können, bei der anderen eine – aufwendige – Spektralanalyse vonnöten ist. Daher gibt es Forschungsansätze, die sich mit der Berechnung der Rotverschiebung aus den Farbwerten statt aus dem Spektrum befassen.

Image Processing

Die SDSS-Teleskope verwenden *Charge-coupled Devices* (CCDs) zur Umwandlung von Licht (bzw. von elektromagnetischer Strahlung) in elektrische Signale. Das Licht wird in fünf unterschiedlichen Wellenlängen aufgenommen, die kurz mit u, g, r, i und z bezeichnet werden. Diese Komponenten werden im FITS-Format gespeichert und können durch Programme wie *IRIS* [7] visualisiert werden. Durch verschiedene Visualisierungsmodule (z.B. Graustufen, 3D, Isolinien), Parameter (z.B. minimal und maximal dargestellter Helligkeitswert, lineare oder logarithmische Interpolation) oder die Komposition dreier Komponenten zu einem RGB-Bild können gewünschte Objekte oder Eigenschaften (z.B. Sterne, Galaxiezentren oder deren Arme) stärker hervorgehoben werden.

3 Planungsphase

Nach Ende der Seminarphase begann der reguläre Semesterbetrieb und damit auch die Arbeit am Projekt. Zunächst mussten viele Abläufe geplant werden. Dies schloss auf der organisatorischen Seite zum Beispiel die Erstellung einer Infrastruktur zur Kommunikation innerhalb der PG und die Aufstellung eines Projektplans ein. Auf der inhaltlichen Seite mussten unter anderem existierende Informationen aufgearbeitet und zentrale Aspekte der Problemstellung identifiziert werden. Es folgt die Auflistung der einzelnen Aufgaben, bevor in Abschnitt 3.2 (ab Seite 27) die wichtigsten Ergebnisse festgehalten werden.

3.1 Arbeitsgruppen und ihre Aufgaben

Die Identifikation der anliegenden Aufgaben fand zumeist in der regelmäßigen PG-Runde statt, an der alle Mitglieder teilnehmen. Ihre Lösung wurde dann Arbeitsgruppen aufgetragen, die ihre Ergebnisse wiederum in der Runde präsentierten. So konnten alle Teilnehmer an grundlegenden Entscheidungen teilhaben, ohne sich im Detail mit allen Themen auseinander zu setzen. Da zu Beginn des Semesters viele Aufgaben anfielen, die aber dafür meist nicht sehr umfassend waren, fanden sich die meisten PG-Teilnehmer in mehreren Gruppen wieder.

Die Arbeit der Gruppen lässt sich grob in die Beschäftigung mit organisatorischen oder inhaltlichen Aufgaben aufteilen.

3.1.1 Aufgaben zur Organisation

Selbstverständlich gab es gerade zu Beginn der PG die Notwendigkeit eine organisatorische Infrastruktur zu erstellen. Diese Aufgabe – die nicht direkt mit der zu entwickelnden Software in Bezug stand – wurde von einem kleinen Personenkreis übernommen. Ihre wichtigsten Teilaufgaben sind hier aufgelistet.

Kommunikation: Um eine reibungslose Kommunikation zu ermöglichen (und bestenfalls sogar zu fördern) und – wo benötigt – gleichzeitig eine Dokumentation der kommunizierten Inhalte zu vereinfachen, musste eine Infrastruktur verschiedener Dienste (Messenger, E-Mail, Forum, Wiki,...) mit klaren Nutzungsregeln erstellt werden. Dazu wurden nach Rücksprache mit der Gruppe verschiedene Systeme eingerichtet – in 3.2.1 (ab Seite 27) werden sie erklärt.

SVN: Um verteiltes Arbeiten am Projekt zu ermöglichen, wurde ein SVN-Server benötigt. Dieser wurde auf einem Server des Lehrstuhls 11 eingerichtet und passend konfiguriert.

Projektplan: Die an eine Projektgruppe gestellte Aufgabe bezog nicht nur die Lösung des konkreten Problems ein, sondern bestand auch in der Organisation des Weges zu eben dieser Lösung. Ein Projektplan ist elementarer Bestandteil dessen und war während der gesamten Planungsphase ein häufig wiederkehrendes Gesprächsthema.

3.1.2 Aufgaben zum Inhalt

Andere Gruppen bearbeiteten Themen, die sich direkt auf die inhaltlichen Aspekte des Projekts bezogen. Die unten gewählte Reihenfolge ihrer Auflistung entspricht in etwa dem Zeitpunkt der Ergebnisvorstellung der einzelnen Gruppen.

Entwurfsmuster: Zu Beginn der Arbeit an der zentralen Aufgabenstellung wurden von drei PG-Teilnehmern bekannte Entwurfsmuster recherchiert und vorgestellt. Obwohl dies selbstverständlich nicht in aller Tiefe erfolgen konnte, sollte so die umfassende Arbeit auf diesem Gebiet zumindest angeschnitten werden, um sie bei der kommenden Problemlösung im Hinterkopf zu haben.

SDSS: Da der SDSS-Katalog zentraler Bestandteil der Aufgabenstellung war, wurden entsprechend großzügig Ressourcen dazu eingeteilt: Fünf Personen bekamen drei Wochen Zeit, um sich intensiv damit auseinander zu setzen. Ihre Ergebnisse sind in Abschnitt 3.2.2 (ab Seite 29) zu finden.

Astronomietools: Die PG entschied schnell, dass die aktuell verwendeten Programme zu kennen eine gründliche Anforderungsanalyse erleichtern würde. Um also einen Überblick über die der Astronomie bereits zur Verfügung stehenden – und eventuell von unserem Projekt ebenfalls erwarteten – Funktionalitäten zu bekommen, wurden einige Personen damit beauftragt, häufig verwendete Tools vorzustellen. Als Tools wurden *Aladin* [27], *Vizier* [40] und *Topcat* [21] ausgewählt.

Data Mining in der Astronomie: Um einen Überblick über den Stand der Dinge im Schnittbereich von Data Mining und Astronomie zu bekommen, bekamen zwei Teilnehmer die Aufgabe, die entsprechende Litratur nach Ergebnissen zu durchforsten. Ihre Ergebnisse (im Abschnitt 3.2.3 ab Seite 31 dargestellt) prägten die Richtung des Projekts deutlich.

UML: Zu Beginn der Konzeptionierung stand die Frage im Raum, ob eine Gestaltung und Dokumentation in UML erfolgen sollte. Da sie nicht umgehend beantwortet werden konnte, wurden zwei Personen abgestellt, um sich mit ihr zu beschäftigen. In Abschnitt 3.2.5 (siehe Seite 32) ist das Ergebnis dokumentiert.

Programmiersprache: Die Wahl der Programmiersprache stellte sich als nicht einfach heraus, denn es galt viele verschiedene Ansprüche gegeneinander abzuwägen. Wegen der Wichtigkeit dieser Entscheidung wurde keine Teilgruppe mit der Recherche beauftragt, sondern alle Teilnehmer bekamen zwei Wochen Zeit, sich mit verschiedenen – und ihnen potenziell unbekannten – Sprachen auseinander zu setzen. Abschließend wurde die Entscheidung gemeinsam getroffen. Im Abschnitt 3.2.6 (siehe Seite 32) wird die Problematik und der Entschluss wiedergegeben.

Programmierkonventionen: Nachdem die Wahl der Programmiersprache getroffen war, wurden zwei programmiererfahrene PG-Teilnehmer mit der Erstellung von Programmierkonventionen beauftragt. Mit der Festlegung auf einen Programmierstil sollte – aus Gründen der Lesbarkeit – ein einheitliches Erscheinungsbild gewährleistet werden. Durch die Erstellung eines Templates für die Eclipse-Entwicklungsumgebung wurde eine den aufgestellten Konventionen entsprechende automatische Formatierung ermöglicht.

3.2 Ergebnisse der Arbeitsgruppen

Die wichtigsten Ergebnisse der verschiedenen Arbeitsgruppen sind im Folgenden in der Reihenfolge der Fertigstellung dokumentiert.

3.2.1 Kommunikation

Die Kommunikation innerhalb einer größeren Gruppe mit zum größten Teil örtlich verteilten Arbeitsplätzen ist ein wichtiger Faktor für den Erfolg eines Projektes. Die Projektgruppe hat daher frühzeitig verschiedene Kommunikationsmittel eingerichtet und genutzt.

Trac

Trac [20] ist eine Web-basierte Projektmanagement-Software, die als Open-Source-Projekt von der Online-Community *edgewall.org* entwickelt wird. Laut eigenem Anspruch verfolgt es dabei einen „minimalistischen Ansatz“ und hat das Ziel „Entwicklern zu helfen, großartige Software zu schreiben und sich dabei im Hintergrund zu halten“ (ebenda).

Es verbindet dabei zahlreiche nützliche Tools:

SVN-Anbindung: Trac kann mit einem SVN-Server verbunden werden und bietet dann unter anderem die Möglichkeit, den Quelltext im Browser zu betrachten sowie eine Timeline, die alle Änderungen im SVN-Repository nachhält.

Wiki: Der perfekte Ort, um alle bedeutsame Entwicklungen fest zu halten. Was dazu alles gehören kann, wird im nächsten Abschnitt erläutert.

Ticketsystem: Das Ticketsystem erlaubt auf einfache Art und Weise, die Dokumentation von Veränderungswünschen am Quellcode. Ob es sich dabei um zentrale Bestandteile des Programms, mögliche Erweiterungen oder fehlerhaften Code handelt, ist dabei irrelevant. Wurde ein Ticket erstellt, kann z.B. eine Zuweisung zu einer Person erfolgen oder in dessen Anhang – ähnlich wie in einem Forum – eine Diskussion geführt werden.

Wiki Markup: Diese und weitere Möglichkeiten zur Ansicht und Dokumentation des Codes werden durch die integrierte Wiki Rendering Engine verbunden. Sie wird in allen textbasierten Tools (also z.B. dem Wiki und dem Ticketsystem) verwendet und ermöglicht nicht nur auf einfache Weise die Formatierung und Strukturierung der Texte, sondern bietet insbesondere eine simple Mechanik an, mit der all diese Teile untereinander verlinkt werden können.

Alle oben genannten Möglichkeiten wurden im Rahmen der Entwicklung umfassend genutzt und so erwies sich die Wahl von Trac (die durch die PG-Betreuer angestoßen wurde) als sehr gut.

Wiki

Ein Wiki ist mittlerweile unumgänglich, um vorhandene Ergebnisse zu protokollieren und allen zur Verfügung zu stellen. Anfänglich wurde ein vom Lehrstuhl zur Verfügung gestelltes Wiki verwendet, aber nach kurzer Zeit zog die PG in das zu Trac gehörige Wiki um. Gründe hierfür waren u.a. eine stärkere örtliche Bündelung der Inhalte, sowie die Trac-Integration,

welche bspw. einfache Verlinkungen auf Meilensteine oder Dateien im Subversion-Repository ermöglicht.

Im Wiki wurden vor allem folgende Informationen dokumentiert:

- Profile der einzelnen PG-Mitglieder, inklusive Kontaktdaten
- Vorträge der Seminarphase und entsprechende Literatur
- Protokolle und Termine der einzelnen PG-Sitzungen
- Informationen zur Arbeit der Astronomen, verwendeter Software und Verfahren
- Dokumentation externer Quellen (Programmiersprachen, Bibliotheken, Anforderungen und externe Programme)
- Konventionen und Ratschläge zur Programmierung und Verwendung von LaTeX
- Dokumentation der einzelnen Gruppenergebnisse und entwickelter Konzepte

E-Mail

Ein weiterer Kommunikationsweg wurde durch die Erstellung zweier E-Mail-Verteiler erschlossen:

- **pg543a@ls11.cs.tu-dortmund.de**
  Empfänger dieses Verteilers sind alle Studenten, Betreuer und Astronomen.
- **pg543s@ls11.cs.tu-dortmund.de**
  Empfänger dieses Verteilers sind nur die Studenten.

www.pggq.de

Während der ersten Wochen der Projektarbeit standen verschiedene Kommunikationsplattformen wie beispielsweise Trac nicht zur Verfügung, weswegen eine eigene Webseite eingerichtet wurde. Die Projektgruppe beschloss, dass diese nur Studenten zur Verfügung stehen soll, aber die grundlegenden Funktionen wurden den Betreuern präsentiert. Zu diesen Funktionen zählten:

Kalender- und Aufgabenübersicht: Es wurde eine Kalenderübersicht erstellt, mit der einzelne Studenten ihre Aufgaben als Balken innerhalb des Kalenders überblicken konnten. Verschiedene Farben markierten den Status der einzelnen Aufgaben.

Eigenes Forum: Schnell wurde erkannt, dass für gewisse Abstimmungen, Fragen und andere Themen der E-Mail-Verteiler ein ungeeignetes Medium darstellt, weswegen ein Forum eingerichtet wurde. Hier wurden insbesondere die Tagesordnungspunkte für bevorstehende Treffen gesammelt.

Chat: Die Webseite beinhaltete unter anderem einen Chat, dessen Verlauf von Zeit zu Zeit archiviert und zur Verfügung gestellt wurde. Auf Grund der im November einsetzenden Teilgruppentreffen und Messengerkommunikation wurde er nur kurze Zeit verwendet.

Subdomain-Weiterleitungen: Es wurden Subdomain-Weiterleitungen eingerichtet, um einen einfachen Zugriff auf das Trac-System und das Forum zu gewährleisten.

RSS-Feed/Startseite: Die Startseite wurde so gestaltet, dass sie die neuesten Informationen als RSS-Feed einband, und so eine schnelle Übersicht über die letzten Änderungen in der Trac Timeline und in der Kalender- und Aufgabenübersicht, sowie neue Beiträge im Forum bot.

Sonstige

Neben den von der PG eingerichteten Kommunikationswegen werden natürlich noch andere verwendet. Hier steht vor allem der private Kontakt via Instant Messenger im Vordergrund. Zu diesem Zweck hat die Projektgruppe gleich zu Beginn des Semsters persönliche Kontaktinformationen gesammelt und ins Wiki gestellt.

3.2.2 SDSS

Da sich die Arbeit der Projektgruppe vorrangig auf die Daten des Sloan Digital Sky Survey (SDSS) beziehen sollte, lag eine eingehende Beschäftigung mit diesem Katalog nahe. So wurden die gespeicherten Daten selbst sowie die zur Verfügung stehenden technischen Möglichkeiten zu ihrer Betrachtung und Verarbeitung untersucht. Anschließend wurden die Ergebnisse ausführlich präsentiert, um auch den anderen PG-Mitgliedern die Arbeit mit den Daten zu ermöglichen.

Die Daten

Der Sloan Digital Sky Survey ist eine Durchmusterung eines Viertels des Himmels, wobei Aufnahmen in fünf Wellenlängenbereichen gemacht werden und von einzelnen interessanten Objekten eine Spektroskopie durchgeführt wird.

Die Daten werden in Streifen (*stripes*) von $2,5°$ Breite und 30° Länge aufgenommen, die wiederum in Sequenzen (*runs*) unterteilt sind. Während einer Sequenz werden verschiedene Felder (*fields*) aufgenommen, die sich aus den Intensitäten in fünf Wellenlängenbereichen (ultraviolett u, grün g, rot r, naher Infrarotbereich i, weiter Infrarotbereich z) zusammen setzen. Für Menschen sichtbare Objekte erscheinen jedoch nicht in der Datenbank, da ihre große Helligkeit dazu führt, dass die sehr empfindlichen CCD-Sensoren des Teleskops nur unscharfe Bilder aufnehmen können.

Die Bilddaten werden in der *Datenverarbeitungspipeline* auf die auf ihnen erkennbaren Objekte hin untersucht. Jedem dieser Objekte werden 454 Attribute zugewiesen, die in der Datenbanktabelle `PhotoObjAll` gespeichert werden (ca. 2 KB pro Objekt). Da sich etwa jeweils 10 % der Streifen überlappen, werden einige Objekte mehrfach erfasst, wobei eine der Objektinstanzen als primäre (*primary*) ausgezeichnet wird – dennnoch erscheinen alle in der Datenbank. Wenn ein Objekt hinter einem anderen liegt, wird das vordere als dessen Elter (*parent*) deklariert. Daten werden oft mehrfach verarbeitet, da die Pipeline laufend weiterentwickelt und verbessert wird (sogenannte *reruns*).

Die Datenbank (aktueller Datenrelease: DR7) basiert auf dem Standard-Datenformat der Astronomie, dem *Flexible Image Transport System* (*FITS*) [57], welches von der NASA und der *International Astronomical Union* entwickelt wurde. Der Header des FITS-Format enthält ASCII-codierte Metadaten, so dass ein Auslesen zumindest der Header selbst durch Menschen möglich ist. In einem FITS-Container können nicht nur Bilder, sondern auch beliebige andere (Binär-)Daten wie Spektren und Tabellen gespeichert werden.

Der Datenarchivserver (*DAS*) ermöglicht den direkten Zugriff auf die Daten, ohne auf die SQL-Datenbank zugreifen zu müssen. Der Zugriff auf die Daten des DR7 erfolgt über Himmelskoordinaten, direkte Angabe der für die Spektralanalyse verwendeten Glasfaserplatten (*plate/fiber*), einer Angabe von *stripe* und *run* oder direkten Verzeichniszugriff.

Werkzeuge

Die Werkzeuge des SDSS lassen sich einteilen in Visualisierungs- und Suchwerkzeuge.

Visualisierungswerkzeuge dienen dem direktem Zugriff auf aufbereitete Bilddaten. Sie zeigen, welche Daten existieren und welche Visualisierungen für Astronomen interessant sein könnten. Zu den gängigsten Visualisierungswerkzeugen gehören:

Finding Chart: Zeigt ein Bild zu eingegebenen Himmelskoordinaten an [48].

Navigate: Ermöglicht es durch Mausklicks durch den Himmel zu navigieren. Hierbei kann man durch das Anklicken von Objekten einige elementare Informationen zu diesen erhalten [50].

Image List: Erzeugt zu einer eingegebenen Positionsliste Miniaturbilder einzelner Objekte des SDSS-Katalogs. Diese Bilder sind dabei mit anderen Tools verlinkt, wodurch ein Weiterarbeiten ermöglicht wird[49].

Explore: Zeigt Informationen zu einem Objekt an und bietet interaktive Links zu weiteren detaillierten Eigenschaften, wie dessen Spektrum oder Nachbarn [47].

Die Suchwerkzeuge stellen vorgegebene SQL-Anfragen dar, welche der Benutzer mittels einiger Parameter anpassen kann. So ist es z.B. möglich, alle Objekte in einem bestimmten Radius um einen Punkt zu ermitteln [51]. Weiter ist es auch möglich, SQL-Anfragen von Hand einzugeben [53]. Anfragen sind dabei zeitlich und in der Anzahl an Rückgabezeilen begrenzt. Das Maximum sind 60 Minuten und 500.000 Zeilen. Beim Werkzeug *Radial/Rectangular Search* sind es nur 10 Minuten. Es besteht jedoch die Möglichkeit, Anfragen als sog. *Batch Jobs* abzugeben, wobei die maximale Laufzeit dann 1000 Minuten beträgt. Hierfür ist allerdings eine Anmeldung notwendig, da eine eigene Datenbank für die Ergebnisse erstellt wird.

Technische Details

Die im vorigen Abschnitt vorgestellten Tools können größtenteils über eine Programmierschnittstelle (*API*) angesprochen werden, wobei als Träger meistens das HTTP verwendet wird. Anfragen werden mittels *GET*- oder *POST*-Methoden gestellt, deren Rückgabe – je nach Tool – als Bild (JPEG), Textdatei (XML, HTML, CSV) oder Verlinkung zu einem weiteren Tool erfolgt.

Die Daten sind in unterschiedlichen Tabellen gespeichert, wobei es eine Vielzahl von vordefinierten Sichten (*views* – Ergebnis einer Anfrage, welches dem Nutzer als *eine* Tabelle dargestellt wird) gibt, welche die Arbeit mit den großen Datensätzen[1] vereinfachen.

Um die Möglichkeiten von Anfragen zu erweitern und diese zu beschleunigen, bietet die Datenbank unterschiedliche Funktionen, Prozeduren und Indizes an. Die Indizes dienen dazu, Anfragen zu beschleunigen (im Vergleich zum linearen Durchlauf) und greifen immer dann,

[1] 585.634.220 Datensätze in der Tabelle `PhotoObjAll`, 175.265.029 Datensätze in der dazugehörigen Sicht `Galaxy`

wenn eine Bedingung bzgl. eines Indizes in der Anfrage enthalten ist. Eine Tabelle kann dabei mehrere Indizes enthalten, untergliedert in drei Kategorien, den Primärschlüssel (*primary key index*), Fremdschlüssel (*foreign key index*, Primärschlüssel einer anderen Tabelle) und den *Covering Index*. Der *Covering Index* ist ein zusammengesetzter Index, der aus mehr als einer Spalte besteht und sofort greift, wenn ein Index in der Bedingung vorkommt.

Spektraldaten

Die Spektrogramme aus dem SDSS-Katalog liegen im Binärformat FITS vor, wobei jedes Spektrum eine Größe von circa 57 MB besitzt. Die Spektraldaten werden im Virtual Observatory (VO) vorgehalten und können auch als CSV oder XML abgerufen werden.

Suchanfragen beim VO funktionieren ähnlich wie bei SDSS: Es kann über eine SQL-Anfrage (Datenbankschema des VO oder des SDSS) nach den Himmelskoordinaten, der Objekt-ID, dem Radial (Ra, Dec, Radius), in selbst definierten Regionen, nach den Eigenschaften des Objekts (z.B. Name, Klasse, usw.) oder der Rotverschiebung gesucht werden. Auch können Objekte mit Ähnlichkeiten zu einem Referenzspektrum gesucht werden.

Suchanfragen lassen sich dank Umkodierung in das *Simple Object Access Protocol* (SOAP) [55] automatisieren. Bei diesem Protokoll handelt es sich um eine XML-basierte Anfragesprache, welche über das HTTP kommuniziert.

Das CSV-Format (*Comma-Separated Values*) ist, da es ein ASCII-Format ist, gut lesbar. Ebenfalls verfügbar ist das eigens für das VO entwickelte *VOTable*-Format [34], welches auf XML basiert. In allen Formaten ist die diskrete Zuordnung der Wellenlänge zu einer Flussdichte (Einheit: Jy, Jansky) enthalten.

3.2.3 Data Mining in der Astronomie

Die Aufgabe der Algorithmen-Gruppe bestand darin, in Erfahrung zu bringen, welche *Machine-Learning-* und *Data-Mining-*Techniken in der Astronomie bereits bekannt sind und in welchem Umfang sie eingesetzt werden. Außerdem sollte konstatiert werden, ob – und wenn ja, welche – besonderen Anforderungen an Algorithmen zur Verarbeitung von Daten aus astronomischen Datenbanken gestellt werden.

Da die zu verwendenden Datenbestände in der Regel in gut dokumentierten und standardisierten Formaten vorliegen, schien eine spezielle Anpassung der Daten nicht vonnöten.

In der Astronomie werden bereits seit einigen Jahren Machine-Learning- und Data-Mining-Techniken angewandt, wobei es dort vier Hauptanwendungsgebiete gibt:

Ausreißerdetektion: In einer großen Datenmenge sind seltene, ungewöhnliche Objekte potenziell interessant. Dies ist ein Bereich, in dem Data-Mining-Algorithmen eine große Rolle spielen können.

Pattern Recognition: Die mit den Teleskopen aufgenommen Objekte sollen anhand ihres Aussehens einer Objektklasse zugeordet werden (also zum Beispiel als Stern oder Galaxie klassifiziert werden). Da sich die Projektgruppe auf die Verarbeitung von Daten in Form von Tabellen spezialisieren wollte, war Bildverarbeitung von eher geringerem Interesse.

Klassifizierung von Daten: Die Daten einer Datenbank in verschiedene Klassen zu unterteilen (also z.B. Sterne anhand der Farbinformationen zu klassifizieren), ist ein typischer Anwendungsfall. Auch hier können Data-Mining-Algorithmen (z.B. aus den Bereichen des Clustering oder Machine-Learning) vielseitig eingesetzt werden.

Echtzeitverarbeitung zum Auffinden kurzfristiger Veränderungen: Einige aus astronomischer Sicht besonders interessante Ereignisse wie zum Beispiel Supernovae treten relativ plötzlich auf und sind auch nur über einen kurzen Zeitraum gut zu beobachten. Ziel wäre es, diese Ereignisse aus dem Nacht für Nacht auftretenden Datenstrom schnell heraus zu filtern, um so eine gezielte Beobachtung des Ereignisses zu ermöglichen.

Zu der Vielzahl an eingesetzten Data-Mining- und Machine-Learning-Techniken zählen unter anderem *Artificial Neural Networks*, *Decision Trees*, *Support Vector Machines*, *Nearest Neighbors* und *Expectation Maximization* [24].

Auf Grund dieser Informationen entschied sich die Projektgruppe, den Fokus der Arbeit auf die Entwicklung eines flexiblen Frameworks (siehe Abschnitt 4.3.1, Seite 37) zu legen und nicht zu versuchen, primär Lösungen für einige einfache Problemstellung zu entwickeln. Diese sollten jedoch im Sommersemester verwendet werden, um die Funktionalität des Frameworks zu überprüfen.

3.2.4 Projektplan

Nachdem Ende Oktober genug Informationen gesammelt waren, konnte ein Projektplan für das erste Semester erstellt werden. Dieser berücksichtigte bereits einige Ideen zur Architektur und richtete die Planung dementsprechend aus. Er deckte das Wintersemester 2009/10 ab und enthielt somit die zeitlichen Abläufe der Meilensteine 1.0 und 2.0.

3.2.5 Planung und Dokumentation mit UML

Zu Beginn der Planungsphase wurde diskutiert, ob es sinnvoll sei, UML zu Dokumentations- und Entwurfszwecken zu verwenden. Im Rahmen dessen wurden als Werkzeuge das *Netbeans UML Plugin* [12] und *MagicDraw* von der Firma NoMagic [10] in die engere Wahl gezogen, um auf dieser Basis zu einer Entscheidung zu kommen.

Nach Evaluation der beiden Werkzeuge wurden sowohl diese selbst, als auch UML als Hilfsmittel verworfen. Ausschlaggebend für die Entscheidung war vor allem, dass kein Teilnehmer bislang ausreichend positive Erfahrungen mit UML-Werkzeugen gemacht hatte und dass es vorgesehen war, Planung und Konzeption in Gruppenarbeit mit Tafel- bzw. Whiteboardanschrieb durchzuführen.

Die so entstehenden Ergebnisse sollten dann fotografiert oder als nachträglich erstellte Diagramme ins Wiki gestellt werden, wo auch zum Verständnis notwendige Kontextinformationen mit festgehalten werden sollten.

3.2.6 Wahl der Programmiersprache

Bevor mit der Implementierung begonnen werden konnte, musste sich die Projektgruppe auf die zu verwendende(n) Programmiersprache(n) einigen. In die engere Auswahl gefasst wurden *C++*, *Python* sowie *Java*. Zudem wurde darüber nachgedacht, verschiedene Programmiersprachen zu verwenden und so für spezielle Teilbereiche auch „exotischere“ Sprachen, wie etwa die im Bereich statistischer Anwendungen weit verbreitete Sprache *R* [19], einzusetzen.

C++

Für C++ sprachen in erster Linie die sehr gute Performance sowie die Möglichkeit, im I/O-Bereich auf einem niedrigen Level operieren zu können. Dies war wünschenswert, um die

Effizienzansprüche an die zu entwickelnde Datenschnittstelle zu erfüllen. Ferner existieren Anbindungen für quasi sämtliche anderen Programmiersprachen.

Als ein großer Nachteil erschien jedoch der sehr hohe Verwaltungsaufwand: Vom Einrichten einer Entwicklungsumgebung und eines Compilers über notwendige Anpassungen auf verschiedenen Plattformen bis hin zum Debuggen bestand die Befürchtung, sich zu sehr mit Aufgaben, die bei anderen Programmiersprachen unnötig bzw. unkomplizierter sind, befassen zu müssen.

In Anbetracht dessen sowie der Tatsache, dass viele Mitglieder der Projektgruppe keine Programmiererfahrung in C++ hatten, wurde die Verwendung verworfen. Es sollte jedoch die Möglichkeit offen gehalten werden, einzelne Teile des Programms, welche besonders hohe Effizienzansprüche stellen, bei Bedarf nachträglich in C++ zu implementieren.

Python

Bei den Überlegungen zu Python stand die leichte Erlernbarkeit sowie die wachsende Verbreitung im wissenschaftlichen (insb. numerischen) Bereich [15, 14] im Vordergrund. Jedoch hatte keines der Mitglieder der Projektgruppe Erfahrung in der Entwicklung größerer Python-Projekte.

Java

Die Wahl fiel also schlussendlich auf Java. Da es in Vorlesungen wie DAP I gelehrt sowie Voraussetzung für das von allen Teilnehmern absolvierte Software-Praktikum war, war die Situation bzgl. der Vorkenntnisse optimal. Ein weiterer wichtiger Aspekt war, dass es mit einer IDE wie *Eclipse* [4] möglich ist, ohne weitere Konfiguration „out of the box“ zu arbeiten.

Hinzu kommen die allgemein bekannten Vorteile von Java wie Plattformunabhängigkeit, automatische Speicherverwaltung und gute Performance. Ebenso eignet sich Java dank Sprachmitteln wie Reflection sehr gut für modulare, erweiterbare Frameworks. Die Anbindung von nativem (d.h. C- oder C++-Code) ist über das *Java Native Interface* (JNI) [41] möglich.

4 Meilenstein 1.0: Konzeptionierung und erste Demos

Anfang November waren die meisten organisatorischen Entscheidungen getroffen und die Vorstellung der Ergebnisse jener Gruppen, die sich mit inhaltlichen Fragen auseinandergesetzt hatten, leiteten die Diskussion um die Lösung der eigentlichen Aufgabenstellung ein. Nach einer gemeinsamen Anforderungsanalyse gab es drei Phasen, in der sich Arbeitsgruppen mit der Analyse und Implementierung von Teilproblemen beschäftigten und die durch eine gruppenübergreifende Verwertung der gewonnenen Erkenntnisse verbunden wurden.

Die Zeitplanung für diesen Meilenstein wird im Folgenden dargestellt. Sie ist Teil eines zu Beginn der Projektgruppe angefertigten Projektplans, der das Wintersemester 2009/10 abdeckte. Anschließend werden die Zusammensetzung und Aufgaben der Gruppen kurz beschrieben und die wichtigsten Ereignisse dieser Phase erläutert.

4.1 Zeitplan

Der folgende zeitliche Überblick über die Planung des ersten Meilensteins bezieht sich auf die Kalenderwochen der Jahre 2009 bzw. 2010.

KW 44 *Anforderungsanalyse und Gruppenaufteilung*
Zu Beginn sollte von der gesamten PG eine allgemeine Anforderungsanalyse durchgeführt werden, um eine sinnvolle Aufteilung des Projekts zu ermöglichen. Anschließend sollten Gruppen zusammengestellt werden, die sich mit jeweils einem Teil beschäftigen würden.

KW 46 *Anforderungsanalyse für Teilprobleme*
Diese Gruppen sollten dann ihrerseits eine Anforderungsanalyse ihres Themengebietes erstellen. Diese gruppenspezifische Analyse hatte insbesondere die Aufgabe, die Schnittstellen zwischen den einzelnen Komponenten des Programms zu definieren – beispielsweise „Welche Daten benötigen die Algorithmen?“ und „Auf welche Weise werden sie an die GUI weitergereicht?“.

KW 48 *Abstimmung und Entwicklung – erste Iteration*
Ende November sollten die Ergebnisse und Entwürfe der einzelnen Gruppen vorgestellt und aufeinander abgestimmt werden, damit die Gruppen anschließend mit den daraus gewonnenen Erkenntnissen mit der Erstellung erster Demos, welche die zuvor gemeinsam formulierten Anforderungen erfüllen würden, beginnen konnten.

KW 51 | *Abstimmung und Entwicklung – zweite Iteration*
Die in der ersten Realisierungsphase gewonnenen neuen Erkenntnisse sollten in einer erneuten gruppenübergreifenden Phase vorgestellt werden, um eine Anpassung der entwickelten Kompomenten an die neuen Ansprüche der jeweils anderen Gruppen zu ermöglichen ohne jedoch bereits ein gemeinsames Funktionieren umzusetzen.

KW 02 | *Vorstellung der Demos*
Der Abschluss des Meilensteins lag in der Vorstellung der Stand-Alone-Demos.

4.2 Arbeitsgruppen

Da im Laufe des Meilensteins fortlaufend neue Anforderungen identifiziert und erarbeitet wurden, blieb die Einteilung in Gruppen nicht konstant, aber ähnlich. Auch deren Aufgaben wandelten sich und entwickelten sich im Groben von Recherche über Konzeptionierung zur Implementierung. An dieser Stelle seien die beiden unterschiedlichen Gruppenaufteilung beschrieben.

4.2.1 November 2009

Die erste Aufteilung war stark vom Projektgruppenantrag geleitet, der eine Auseinandersetzung mit den großen Datenmengen der Astronomie forderte.

Datenschnittstelle: Dementsprechend wurden fünf Personen damit betraut, eine Schnittstelle, welche effizientes Arbeiten auf großen Datenmengen ermöglicht, zu entwickeln.

Data-Mining-Algorithmen: Diese bereits während der Planungsphase existente Gruppe von zwei Personen setzte ihre Arbeit fort.

GUI/Visualisierung: Neben der Verarbeitung war auch die Visualisierung großer Datenmengen ein Auftrag an die Gruppe, und so setzten sich hiermit und mit ersten Entwürfen zu einer grafischen Benutzeroberfläche vier Mitglieder auseinander.

4.2.2 Ab Dezember 2009

Erste Zwischenergebnisse und architektonische Entscheidungen (siehe 4.3.1 und 4.3.2) bewirkten eine Fokussierung auf das generische Zusammenstellen verschiedener Algorithmen zu einer Anwendung durch den Benutzer. Dementsprechend wurde ein neuer Schwerpunkt auf diese Aspekte gelegt:

Datenschnittstelle: Für die Entwicklung der Schnittstelle waren im Folgenden nur noch drei Personen zuständig.

ApplicationCenter: Das Konzept der automatischen Prüfung und Ausführung einer Anwendung wurde nun von zwei Teilgruppen zu je zwei Personen bearbeitet.

GUI/Visualisierung: Diese Gruppe blieb unverändert.

4.3 Umsetzung & Ergebnisse

Es ist zu vermerken, dass die Planung des Meilensteins ohne Verzug eingehalten werden konnte und am 12.01.2010 mit der Vorstellung der einzelnen Demos endete. Wichtiger als diese waren jedoch die zu Beginn des Semesters getroffenen Entscheidungen zur Architektur von Brick. Bereits zu diesem Zeitpunkt enthielt das Konzept alle wichtigen Merkmale des späteren Programms und der gründlichen Planung und Umsetzung dieser Phase ist es zu verdanken, dass das Projekt ohne konzeptionelle Neuorientierung auskam.

Die wichtigsten Ergebnisse werden hier kurz beleuchtet.

4.3.1 Konkrete Anwendungen in einem allgemeinen Framework

Zur Entwicklung eines Systems, das mit Hilfe von Data-Mining-Algorithmen große Datenbestände untersucht, ist die Semantik der Daten kein zentraler Aspekt. Deswegen entschied die Projektgruppe schon früh, Brick nicht auf astronomische Daten einzuschränken, sondern eine vielseitige Data-Mining-Suite zu entwickeln. Diese sollte modular implementierte Datenadapter, Algorithmen (zusammen allgemein *Knoten*) und Visualisierungen anbieten und einem Benutzer erlauben, daraus eine auf sein Problem angepasste *Anwendung* (bzw. *Application*) zu konstruieren.

Eine solche Anwendung würde einem gerichteten Graphen (dem sog. *Prozessgraphen*) entsprechen, in dem die Knoten entsprechend dem Datenfluss miteinander verbunden sein würden. Außerdem wurde beschlossen, Anwendungen in einem XML-Format zu speichern. Dies bietet den Vorteil, dass zum Auslesen und Verarbeiten zahlreiche Bibliotheken existieren und es als *Plain-Text*-Format ohne Hilfsmittel verarbeitbar und z.B. per Skript automatisiert erstellbar ist.

Voraussetzung zur erfolgreichen Benutzung von Brick würden zwar Grundkenntnisse über die implementierten Algorithmen[1] sein, aber darüber hinaus sollte der Benutzer kein Wissen aus dem Bereich der Informatik benötigen.

4.3.2 Freie Erweiterbarkeit durch Plugins

Um durch Anwendung von Data-Mining-Algorithmen die Bearbeitung astronomischer Fragestellungen zu ermöglichen, hätte eine Vielzahl von Algorithmen, Visualisierungsmöglichkeiten und Adaptern für verschiedene Datenformate implementiert werden müssen. Da außerdem das Ziel formuliert wurde, Brick als generische – also nicht auf astronomische Problematiken eingeschränkte – Data-Mining-Suite zu entwerfen, wuchs diese Menge weiter an. Zur Bewältigung dieser Aufgabe fehlte sowohl das Hintergrundwissen im astronomischen Bereich als auch die Zeit für die Implementierung einer ausreichend großen Menge von Algorithmen. Auch aus diesem Grund forderte der Projektgruppenantrag ein „modular aufgebautes System“ und daher wurde beschlossen, das System in diesem Bereich auf Erweiterbarkeit auszulegen.

Daher wurde ein Plugin-System konzipiert, das es ermöglichen sollte, jederzeit neue Knoten zur Kapselung von Algorithmen und Datenquellen/-senken und neue Austauschformate zwischen solchen Knoten hinzuzufügen. Auf diese Weise sollte jede Installation eines Servers

[1] Von Bedeutung ist dabei insbesondere, Eigenheiten der Algorithmen zu kennen, wie bspw. dass bei k-Means die Anzahl der Cluster vorbestimmt werden muss oder was die Parameter ε und *minPoints* bei DBSCAN bedeuten.

individuell an unterschiedlichste Aufgaben angepasst werden können, ohne etwas an der Implementierung des eigentlichen Systems zu ändern oder diese zu kennen.

Der Ansatz der Projektgruppe war also sehr allgemein, da aber die Bearbeitung astronomischer Fragestellungen im Fokus lag, sollten – sobald die Rahmenbedingungen hergestellt sein würden – Plugins integriert werden, welche dazu notwendige Funktionalitäten realisieren würden. Ein Beispiel dafür ist ein Adapter für den SDSS-Katalog.

4.3.3 Visualisierung des Prozessgraphen

Schon kurz nach dem Entschluss, das freie Zusammenstellen von Knoten aus verschiedenen Plugins zu einer Anwendung zu erlauben, wurde das Vorhaben formuliert, dies in einer grafischen Benutzeroberfläche in Form des sogenannten „Klickgraphen“ zu ermöglichen. Dabei sollte dem Benutzer die Möglichkeit gegeben werden, aus den zur Verfügung stehenden Plugins per *Drag and Drop* eine durch ihren Prozessgraphen repräsentierte Anwendung zu erstellen und auf einfache Weise alle nötigen Parameter einzustellen.

Da eine vollständig selbst entwickelte Graphvisualisierung allein schon aus Gründen des Zeitmanagements nicht sehr sinnvoll erschien, gab es Überlegungen zur Wahl einer möglichst umfassenden externen Bibliothek. Die zuständigen PG-Mitglieder haben sich schließlich für die *Jung*-Bibliothek entschieden. Dabei haben Faktoren wie Open-Source-Verfügbarkeit, leichter Zugang und vorhandene Beispielapplikationen, welche die Möglichkeiten der Bibliothek offenkundig zeigten und eine relativ schnelle Einarbeitung erlaubten, eine wesentliche Rolle bei der Entscheidung gespielt.

Bis zum Ende dieses Meilensteins wurden bereits einige Funktionalitäten implementiert. Es war möglich, Knoten frei zu positionieren. Außerdem konnte der Graph in eine XML-Datei geschrieben und von dort auch wieder ausgelesen werden.

4.3.4 Prüfung einer Anwendung

Bei einer vom Benutzer zusammengestellten Anwendung muss selbstverständlich kontrolliert werden, ob deren Aufbau sinnvoll ist, d.h. es muss geprüft werden, ob die Knoten richtig parametrisiert und verbunden sind. Dazu wurde das Konzept des *Sanity Checks* entwickelt. Dieser sollte eine Application, die sich in einem beliebigen Stadium der Konstruktion befindet, analysieren und dem Benutzer alle gefundenen Fehler mitteilen. Dabei handelt es sich um eine reine Analyse, d.h. es sollte kein Teil der Anwendung tatsächlich ausgeführt werden. So sollte verhindert werden, dass potentiell langwierige Berechnung vergebens erfolgen.

Während einige Teile der Prüfung vom System selbst erledigt werden können, muss ein Großteil auf Ebene der Knoten erfolgen. Dazu gehört z.B. die Prüfung aller Parameter sowie die Kontrolle, ob die Formate der an den Eingängen anliegenden Daten die Anforderungen des Algorithmus erfüllen. Aufgrund der freien Erweiterbarkeit war klar, dass diese Prüfung im Wesentlichen dem Knoten selber zugeteilt werden mussten, da Brick nicht alle existierenden und zukünftigten Formate kennen kann. Es wurden zwei Phasen konzipiert: In der ersten soll der Knoten seine Eingaben (Parameter und Formate der Eingabedaten) prüfen, bevor er in der zweiten das Format seiner Ausgabe(n) vorhersagt. Der Sanity Check besteht auf Anwendungsebene also darin, sämtliche Knoten in topologischer Reihenfolge zu durchlaufen und beide Phasen direkt nacheinander ausführen zu lassen. So kann die Prüfung für den gesamten Prozessgraphen erfolgen.

Außerdem wurde beschlossen, den Sanity Check von der GUI nach *jeder* Eingabe anstoßen zu lassen. So sollte sichergestellt werden, dass der Benutzer zu jedem Zeitpunkt weiß, welche seiner Einstellungen fehlerhaft sind und welche noch fehlen. Dies ist vergleichbar mit der Anzeige von Fehlern in modernen IDEs, bspw. in *Eclipse*.

4.3.5 Datenschnittstelle

Ein Hauptaugenmerk der Projektgruppe sollte auf dem Umgang mit Datenmengen im Terabyte-Bereich liegen. Dies war einer der Hauptgründe, C++ bei der Wahl der Programmiersprache (siehe Abschnitt 3.2.6, Seite 32) so intensiv zu bedenken. Nachdem die Wahl auf Java gefallen war, gab es wenig Spielraum auf I/O-Ebene an der Effizienz der Datenschnittstelle zu arbeiten.

Stattdessen wurde versucht, eine effiziente Datenanbindung durch andere Konzepte zu ermöglichen. Insgesamt folgt der Aufbau einer Application einer *Pipes-and-Filters*-Architektur, wobei die Daten durch die vom Benutzer zusammengestellte Anwendung fließen. Die von Entwicklern zur Verfügung gestellten Knoten stellen dabei die Filter dar. Sie definieren Anschlussstellen (Kanäle bzw. *Channels*), welche verbunden werden können. Man beachte, dass der Begriff *Pipe* bei Brick eine über die einer einfachen Verbindung hinausgehende Bedeutung hat: Sie realisiert den Datentransport zwischen zwei Knoten. So gibt es z.B. verschiedene Pipes, je nachdem, ob der nachfolgende Knoten sequentiellen oder wahlfreien Zugriff auf die Daten benötigt. Pipes werden, entsprechend ihrer Notwendigkeit, automatisch eingefügt.

Eine von einem Knoten angeforderte Datenquelle für sequentiellen Zugriff würde einmal gelesene Daten nicht mehr zur Verfügung stellen und deswegen nicht mehr vorrätig halten müssen. Zusammen mit der möglichst parallelen Abarbeitung der Knoten sollte so verhindert werden, dass die Knoten einer Application nacheinander alle Daten vollständig von der Festplatte lesen und auf diese auch wieder schreiben, was zwangsläufig zu hohem I/O-Aufwand führen würde. Stattdessen könnte eine etwa hauptspeichergroßes Fenster der Daten von vielen Knoten und im Optimalfall gar von der gesamten Application verarbeitet werden, bevor es wieder abgelegt werden müsste.

Außerdem erlaubt das Konzept der Plugins (siehe Abschnitt 4.3.2, Seite 37) eine effizientere Implementierung des Datentransports (auch Pipes sollten über Plugins eingebunden werden können) einfach zu integrieren.

4.3.6 Aufteilung in Client und Server

Die Verarbeitung von Daten mit einem Umfang im Terabyte-Bereich ist nur mit darauf ausgelegten Hochleistungsrechnern oder Rechenclustern (wie z.B. Universitätsservern) sinnvoll. Auf eben solchen Servern werden typischerweise auch die Datenbanken gespeichert, welche diese Datenmengen hergeben. Da in diesen Systemen also im Normalfall sowohl die benötigte Speicherkapazität als auch hohe Rechenleistung vorhanden ist, hat die Projektgruppe schon früh beschlossen, Anwendungen dort ausführen zu lassen.

Auf der anderen Seite findet die alltägliche Bedienung eines Werkzeugs, wie Brick es sein soll, üblicherweise an einem Arbeitsplatzrechner statt. Zudem ermöglichen Hochleistungssysteme nicht-Administratoren oftmals keinen direkten Zugriff. Es bestand also die Notwendigkeit, Brick von einem Arbeitsplatz aus bedienen zu können.

Die Vereinigung dieser Ansätze bestand in der Aufteilung des Systems in *Client* und *Server*. Ersterer sollte z.B. auf einem Arbeitsplatzrechner laufen und über eine GUI oder ein

Kommandozeilen-Interface die Konstruktion sowie die Beauftragung der Ausführung von Anwendungen erlauben. Auf dem System, das auch die Daten enthält auf denen gearbeitet werden soll, würde der Server laufen und von verschiedenen Clients Anwendungen erhalten, die geprüft oder ausgeführt werden sollen.

Der Teil des Servers, der sich mit Anwendungen befasst, wurde als *Framework* bezeichnet und kapselt einen Großteil der gesamten Funktionalität von Brick. Weitere Teile sind neben der GUI und einem CLI insbesondere die Client/Server-Schnittstelle, welche die Kommunikation übernimmt. Deren Umsetzung erfolgte im folgenden Meilenstein 2.0 (siehe Abschnitt 5.2.1, Seite 42).

5 Meilenstein 2.0: eine zusammenhängende Demonstration

Der zweite Meilenstein hatte das Ziel, die Stand-Alone-Demos, die Anfang Januar fertig gestellt wurden, zu einem Programm zu verbinden und so zu zeigen, dass das entwickelte Konzept tragfähig ist. Dabei sollten nur solche neuen Funktionalitäten umgesetzt werden, die zur Verbindung der Einzelteile nötig waren. Deswegen bestand die Arbeit an BRICK in dieser Phase – abgesehen von der GUI (siehe Abschnitt 5.2.3, Seite 43) – größtenteils aus technischen Modifikationen bzw. Erweiterungen.

Dazu standen vier Wochen zur Verfügung und der Meilenstein sollte mit einer Vorstellung der Demo am 09.02.2010, zu der auch Besucher vom Lehrstuhl für Astronomie der Ruhr-Universität Bochum eingeladen waren, beendet werden.

5.1 Arbeitsgruppen

Um dem Lehrveranstaltungscharakter Rechnung zu tragen, sollte durch eine Neuaufteilung der Gruppen verhindert werden, dass die Teilnehmer nur an den von ihnen entworfenen Programmaspekten arbeiten und so keinen Überblick über das gesamte Projekt erlangen. Aus diesem Grund wurde eine neue Aufteilung gewählt, die in jeder Gruppe möglichst nur eines der ursprünglichen Mitglieder beließ (welches dann den Einstieg der neuen Gruppenmitglieder in die vorhandenen Ergebnisse erleichtern sollte) und die restlichen PG-Teilnehmer umverteilte.

GUI: Diese Gruppe wurde mit drei Personen besetzt und sollte die Entwicklung der grafischen Benutzeroberfläche fortführen.

Framework: Nachdem sowohl die Datenschnittstelle als auch die Prüfung und Ausführung von Anwendungen als Teile eines übergeordneten Frameworks verstanden wurden, wurden diese Aufgaben zusammengelegt. Fünf Personen wurden damit beauftragt, das Zusammenspiel dieser und neuer Komponenten zu entwickeln.

Netzwerk: Die Kommunikation zwischen dem Framework und der GUI war als Client/Server-Architektur modelliert worden. Drei Mitglieder bekamen die Aufgabe, diese umzusetzen.

5.2 Umsetzung & Ergebnisse

Die Koordination der einzelnen Programmteile erforderte einen deutlich größeren Aufwand, als zuvor abgeschätzt worden war. Die abschließende Version war zwar funktionstüchtig, aber an vielen Aspekten des Programms war der Status einer Alpha-Version noch deutlich zu erkennen. Dennoch konnte dieser Meilenstein als Erfolg gewertet werden, denn er zeigte, dass das Konzept sowohl den gestellten Ansprüchen als auch nötigen Erweiterungen gewachsen war.

Neben vielen rein technischen Arbeiten wurden auch während dieses Meilensteins wichtige Konzepte entworfen und/oder umgesetzt. Einige davon seien hier dokumentiert.

5.2.1 Umsetzung der Client-Server-Schnittstelle

Zunächst war die Frage zu klären, ob die Schnittstelle komplett selbst entworfen, oder auf unterstützende, fertige Bibliotheken zurückgegriffen werden sollte. Die Verwendung von *Remote Method Invocation* (kurz *RMI*), die Java-eigene Lösung für die Kommunikation zwischen Virtual Machines, wurde direkt zu Anfang geprüft. Die Vorteile davon, eine Schnittstelle selbst zu konzeptionieren, überwogen nicht den Nachteil eines erheblich höheren Zeitaufwandes. Daher wurde RMI als Lösung für die Schnittstelle gewählt.

Für die Verwendung der Client/Server-Schnittstelle wurde zum einen eine ausführbare Anwendung, welche den Server realisiert und die über RMI empfangenen Daten an das Framework weitergibt, zum anderen eine Bibliothek, welche es auf Client-Seite dem Programmierer ermöglicht, mit dem Server zu kommunizieren, angelegt.

Besonderes Augenmerk lag natürlich auf der Übertragung einer Anwendung vom Client zum Server, sowie der Ergebnisse der Prüfung und der durch die Ausführung erhaltenen Daten vom Server zum Client.

XML

Als Speicherformat für eine Application wurde ein XML-Format gewählt, welches in einer eigenen *Document Type Definition* (*DTD*) beschrieben wurde. Der Grund für die Wahl dieses Formats lag vor allem in der guten Lesbarkeit und der einfachen Realisierung eines Parsers in Java. Ein solcher wurde implementiert und erlaubte es fortan, gespeicherte Anwendungen auszulesen und in Java-Objekte umzusetzen, auf denen dann die Weiterverarbeitung erfolgte.

Neben der eigentlichen Speicherfunktion wurde dieses Format auch bei der Übertragung von Anwendungen an das Framework benutzt, wo diese Dateien validiert und anschließend ausgeführt werden sollten.

Reports

Die Ergebnisse des Sanity Checks (siehe Abschnitt 4.3.4, Seite 38) werden vom Framework in einem hierarchisch organisierten System von Berichten (*Reports*) abgelegt. Es existieren Berichte zu Knoten, ihren Parametern und Kanälen, Kanten sowie zur gesamten Anwendung. Alle Ergebnisse der Prüfung werden hier abgelegt und stehen dem Client nach der Übertragung zur Verfügung.

Während Berichte zur Anwendung und ihren Kanten und Knoten vom Framework angelegt und mit Informationen versehen werden, müssen die Knoten die Berichte zu ihren Parametern und Kanälen selbstverständlich selber verwalten, denn aufgrund der Modularität der Plugins (siehe Abschnitt 4.3.2, Seite 37) kennt das Framework deren internen Aufbau nicht. Der Client erfährt also nur über die Berichte von den Kanälen und Parametern eines Knotens und kann daraus unter anderem Informationen wie Name, Beschreibung und gefundene Probleme auslesen. Insbesondere enthält ein Parameter-Report auch den Wert eines Parameters nach der Prüfung durch den Knoten (bspw. Default-Werte für Parameter, deren Wert noch nicht spezifiziert wurde).

Hooked Channels

Es war erforderlich, für den Client eine Möglichkeit zu schaffen, Ergebnisse der Ausführung einer Anwendung abzurufen. Das Problem hierbei war grundsätzlich, dass die Senkenknoten (wie alle anderen Knoten auch) komplett auf dem Server ausgeführt werden und demzufolge auch nur dessen System zur Verfügung haben. Ein Knoten, der die Ergebnisse der Ausführung eines Algorithmus in bspw. eine VOTable-Datei auf dem Server schreibt, ist ohne entfernten Zugriff auf das Dateisystem des Servers uninteressant.

Aus diesem Grund wurde die Möglichkeit geschaffen, bestimmte Ausgabekanäle als *hooked* zu markieren (sog. *Hooked Channels*). Dies war grundsätzlich nur möglich bei Kanälen, an denen Tabellendaten ausgegeben werden. Während bzw. nach der Ausführung der Anwendung konnte der Client – unter Angabe der Identifikatoren des jeweiligen Knotens und Kanals – die an diesem Kanal ausgegebenen Daten abfragen.

Mit Hooked Channels wurde die strikte Trennung zwischen Framework und konkreten Daten verletzt, was in Konflikt mit der Forderung nach leichter Erweiterbarkeit stand. Um auf äquivalente Art und Weise die Daten anderer Kanäle, bspw. Entscheidungsbäume, auf den Client übertragen zu können, hätte die Server-Schnittstelle geändert werden müssen, was eine über die Plugin-Ebene hinausgehende Modifikation bedeutet hätte. Daher stellten Hooked Channels von Anfang an eher eine Übergangslösung dar, welche aufgrund der Zielvorgabe des Meilensteins notwendig war.

Ersetzt wurde sie während des Meilensteins 3.0 durch sogenannte *GUI-Knoten* (siehe Abschnitt 6.4.1, Seite 53).

5.2.2 Autoparameter

Die – z.B. in der GUI vom Benutzer eingestellten – Parameter eines Knotes werden in einer XML-Datei (siehe oben) festgehalten und so an das Framework übertragen, welches daraus eine Abbildung der Form *Parametername* $\mapsto$ *Wert* erstellt. Da die Attribute eines Knotens dem Framework nicht bekannt sind, müssen in erster Instanz die Knoten selber das Auslesen der Parameterwerte aus dieser Abbildung übernehmen. Um dem Knotenprogrammierer diesen immer gleichen Vorgang zu ersparen, wurde das Konzept der *Autoparameter* entwickelt und umgesetzt.

Es gibt dem Knotenprogrammierer die Möglichkeit, Java-Attribute mittels einer Annotation zu kennzeichnen. Dies erlaubt dem Framework, einen Zusammenhang zwischen dem Parameternamen aus der XML-Datei (der zunächst ja nichts weiter als ein String ist) und den Attributen der Knoten herzustellen, und ermöglicht so eine automatische Zuweisung. In der Annotation kann der Parameter weiterhin noch als „verpflichtend“ (*mandatory*) deklariert werden. Ist dann kein Wert für diesen Parameter angegeben, wird ein entsprechender Fehlerbericht erzeugt.

Selbstverständlich bleibt dem Knotenprogrammierer die Möglichkeit, die Abbildung selber auszulesen und somit Parameterwerte manuell zuzuweisen. Dies ist insbesondere von Bedeutung, wenn die Menge der Parameter dynamisch ist, d.h. von den Werten anderer Parameter abhängt.

5.2.3 Neu konzipierte GUI

Aufgrund der zu Beginn dieses Meilensteins geänderten Zusammensetzung der GUI-Gruppe kamen in diesem Bereich eine Fülle neuer Ideen und Anforderungen auf. In diesem Rahmen

wurde entschieden, von der Swing-basierten auf eine neue Oberfläche umzuschwenken, die auf der *Netbeans Rich Client Platform* (*RCP*) [13] basieren sollte. Die alte GUI, welche bis dahin der Erprobung von Darstellungskonzepten gedient hatte, wurde nicht weiterentwickelt.

Gründe für den Umstieg

Ausschlaggebend waren vorrangig die von der Netbeans-Plattform bereitgestellten Funktionalitäten:

Generische Plattform für GUI-Anwendungen: Die Netbeans RCP stellt Komponenten wie Menüleisten und Unterfenster zur Verfügung. Unterfenster können z.B. innerhalb des Fensters frei platziert oder sogar herausgelöst werden. Alle GUI-Komponenten basieren auf Standard-Swing-Klassen und können dementsprechend durch Vererbung erweitert werden.

Plugin-Management: Funktionalitäten innerhalb der Netbeans RCP werden ausschließlich durch Module bereitgestellt. Diese können unabhängig vom restlichen Gerüst entwickelt und ausgeliefert werden. Dadurch wird auch konzeptionell eine modulare Softwareentwicklung unterstützt.

Des Weiteren bietet die Plattform die Möglichkeit, einzelne oder gebündelte Module zu aktualisieren, indem sie deren Status mit einem externen Server abgleicht und bei Bedarf eine neue Version herunterlädt und installiert.

***Visual Library*:** Die *RCP*-Umgebung stellt standardmäßig diese Bibliothek zur Verfügung, die es ermöglicht, Graphen – wie den Prozessgraphen einer Anwendung – mittels Drag and Drop zu erzeugen und zu manipulieren.

Die Verwendung der Netbeans Rich Client Platform ermöglicht es, den Entwicklungsprozess auf die eigentliche Funktionalität der grafischen Schnittstelle zu fokussieren und den beteiligten Entwicklern Arbeiten an den übrigen Komponenten abzunehmen.

Programmteile der GUI

Obwohl die GUI zu Beginn des Meilensteins 2.0 auf der Softwareseite eine komplette Neukonzeption war, lassen sich in ihr viele Parallelen zum ersten Entwurf finden. So bleibt als zentraler Teil der Benutzeroberfläche der Prozessgraph bestehen, der sowohl zur Darstellung als auch zum Erstellen einer Anwendung benötigt wird. Im Folgenden sollen einzelne Bestandteile der neuen GUI beschrieben werden.

Prozessgraph

Um den Prozessgraphen darzustellen, fiel die Wahl auf die Visual Library. Gründe hierfür waren sowohl die gute Integration in die Netbeans-Plattform als auch eine sehr gute Dokumentation anhand von Anwendungsbeispielen. Zudem bot die Bibliothek bereits eine Art von Knotendarstellung inklusive großem Funktionsumfang an, die schon „out of the box“ der Wunschdarstellung eines Prozessgraphen entsprach.

Projekte

Ein *Projekt* dient als eine Art Container für alle Informationen, die für die Ausführung und den Bau einer Anwendung nötig sind. Die Netbeans-Plattform bietet dazu eine API an, die von der Projektverwaltung zu großen Teilen genutzt wird. Ein Projekt hat dabei zwei Ebenen: Zum einen ist es ein Objekt, welches zur Laufzeit erzeugt wird und die entsprechenden Daten enthält; zum anderen ist es eine Kombination von Dateien und Ordnern, welche die entsprechenden Daten enthalten.

Lokaler Server

Um das lokale Arbeiten – ohne jegliche Verbindung „nach außen“ – zu ermöglichen, wurde eine lokale Version des Servers integriert. Er wird automatisch zusammen mit der GUI gestartet.

Visualisierungsmodule

Da erwartet wurde, dass die Ein- und Ausgabedaten in der Regel mehr als drei Dimensionen haben, mussten Wege gefunden werden, diese adäquat zu visualisieren. Da einfache 2D- oder 3D-Plots nicht ausreichten, wurden zwei alternative Lösungsansätze ausgewählt.

Tabellendarstellung

Die naheliegendste Darstellung hochdimensionaler Daten ist vermutlich eine Tabelle. Hierbei wurde auf die Java-Standardbibliothek zurückgegriffen. Das verwendete Modell wurde so erweitert, dass Spalten eine Farbe zugewiesen werden kann, deren Intensität zellenweise die Position im Wertebereich der Dimension der jeweiligen Spalte repräsentiert.

Durch Sortieren eine Spalte lässt sich so schnell erkennen, ob zwischen den Werten der sortierten und den Werten einer anderen Spalte Korrelationen gegeben sind. Hierbei wurde darauf geachtet, dass die Sortierung der Spalten stabil ist.

5D-Plot

Um 5 auswählbare Dimensionen (Spalten) zu visualisieren wurde ein dreh- und zoombarer 3D-Plot, basierend auf der *Java-Easy-3D*-Bibliothek [8], entwickelt. Diese Bibliothek stellt eine Abstraktionsschicht für *OpenGL* bzw. dessen Java-Implementierung *JOGL* [9] dar und profitiert damit von dessen Leistungen und Unterstützung durch Grafiktreiber und -hardware.

Die ersten drei Dimensionen werden in einem dreidimensionalen Koordinatensystem durch ihre Positionierung bezüglich der Koordinatenachsen dargestellt, während die vierte durch die Größe und die fünfte durch die Farbe (bzw. deren Intensität) der Objekte repräsentiert wird.

6 Meilenstein 3.0: eine lauffähige Minimalversion

Während der ersten beiden Meilensteine bestand die Arbeit am Programm aus sehr vielen stark miteinander verwobenen Aufgaben, bei denen eine Aufteilung in kleine Gruppen mit separaten und gut definierbaren Teilzielen schwierig bis unmöglich war. Die fortschreitende Fertigstellung des Frameworks zum Ende des Wintersemesters 2009/10 hatte diese Umstände geändert. Alle wichtigen Elemente waren implementiert und neue Funktionalitäten konnten gezielt eingebaut werden und ließen sich so wesentlich einfacher planen.

Außerdem war die Schnittstelle zur Einbindung von Knoten soweit gereift, dass von keinen bedeutenden Änderungen mehr ausgegangen werden musste und die Integration weiterer Datenquellen, -senken und Algorithmen begonnen werden konnte. Auch die Ausarbeitung der Schnittstelle für Visualisierungsplugins sollte noch im Mai abgeschlossen werden. Die durch die Architektur vorgesehene Trennung zwischen Framework und Plugins war somit umgesetzt und es konnte parallel zur Weiterentwicklung mit der Integration diverser Algorithmen begonnen werden.

Insgesamt ermöglichte dies zum Sommersemester 2010 einen Wechsel des Organisationsparadigmas, in dem eine detaillierte Aufgabendefinition, -einteilung und -planung wesentlich gezielter und häufiger eingesetzt werden konnte.

Der erste Meilenstein des neuen Semesters bestand aus zwei großen Aufgabenfeldern: der Weiterentwicklung des Programms auf der einen und der Integration weiterer Plugins auf der anderen Seite. Am Ende sollte eine stabile Version mit den wichtigsten Funktionalitäten und einigen verwendbaren Data-Mining-Algorithmen existieren.

6.1 Arbeitsgruppen

Die beiden identifizierten Aufgabenfelder schlugen sich direkt in der Aufteilung der Mitglieder in verschiedene Gruppen nieder.

Weiterentwicklung: Diese Gruppe befasste sich mit der Weiterentwicklung des Programms und bestand aus den drei Untergruppen **GUI** (zwei Personen), **Netzwerk** (eine Person) und **Framework** (zwei Personen). Diese beschäftigten sich primär mit der Weiterentwicklung ihrer jeweiligen Komponente, wobei bei übergreifenden Aufgaben die betroffenen Untergruppen natürlich zusammenarbeiteten.

Algorithmen: Außerdem gab es eine Algorithmengruppe, die aus sechs Personen bestand und die Integration verschiedener Data-Mining-Algorithmen und Visualisierungsmöglichkeiten ins Framework übernahmen.

Hierbei sollte es sich um *Proof-of-Concept*-Implementierungen handeln, d.h. der Fokus wurde auf die Integration des Algorithmus in Brick und nicht auf eine effiziente Implementierung gelegt. Ziel war es außerdem, anhand dieser konkreten Arbeit mit dem

Framework fehlende Funktionalitäten zu identifizieren, deren Umsetzung durch die Weiterentwicklungsgruppe bei Bedarf in einen späteren Meilenstein erfolgen sollten.

6.2 Ziele

Im Folgenden werden die Ziele der einzelnen Gruppen aufgelistet. Die Angabe **PW** bezeichnet dabei die Anzahl der zur Lösung der jeweiligen Aufgabe kalkulierten Personenwochen.

6.2.1 Weiterentwicklung

Die hierunter zusammengefassten Aufgaben schlossen Lücken, die zu dem Zeitpunkt noch bestanden und realisierten somit eine Version, in der zum ersten Mal alle wichtigen geplanten Funktionalitäten realisiert waren.

Framework

Das Framework beinhaltete zu diesem Zeitpunkt beinahe alle zu seiner Verwendung benötigten Elemente, aber manche davon waren fehlerhaft. Diese Fehler zu korrigieren hatte natürlich oberste Priorität. Gleichzeitig stand die Gruppe jedoch zur Verfügung, um Funktionalitäten zum Framework hinzuzufügen, die zur Verbesserung anderer Komponenten benötigt wurden.

automatische Listenparameter (1 PW): Die per Annotationen erreichte automatische Zuweisung von Werten aus der XML-Datei zu den entsprechenden Parametern der Knoten sollte auf Listenparameter ausgeweitet werden.

GUI-Knoten (2 PW): Der Transportmechanismus von Daten vom Server zur GUI (bzw. allgemein zum Client), der vormals durch *Hooked Channels* (siehe Abschnitt 5.2.1, Seite 43) realisiert war, wurde geändert. Hooked Channels wurden abgeschafft und durch sog. *GUI-Knoten* ersetzt.

Threads (4 PW): Jeder Knoten läuft in einem eigenen Thread, allerdings waren Synchronisationsmechanismen bei der Kommunikation zwischen Knoten nur unzureichend realisiert. Diese mussten also implementiert werden.

Exception-Handling (2 PW): In Zusammenarbeit mit der Netzwerkgruppe sollte verhindert werden, dass Exceptions aus dem Framework bis in die GUI gereicht werden und dort zu Abstürzen führen. Stattdessen sollte das Framework Exceptions fangen und in Fehlerberichte umwandeln.

GUI

Da die Entwicklung der GUI Anfang des Jahres 2010 neu begonnen wurde (siehe Abschnitt 5.2.3, Seite 43), bestand mehr Arbeitsaufwand als bei den anderen Kompenten. Die Aufgaben werden unten ausformuliert, wobei die mit ▶ markierten Punkte Priorität hatten.

Basisfunktionalität (2 PW): Einige Funktionen, die für die Benutzung des Frameworks durch die GUI von elementarer Bedeutung sind, waren zu diesem Zeitpunkt noch nicht implementiert:

Projekte (1 PW): Projekte dienen als Container für einzelne Applications und sollten per Wizard erzeugt werden, welcher noch implementiert werden musste. Zusätzlich sollten diese geöffnet und gespeichert werden können.

- ▶ Erzeugen
- ▶ Speichern/Laden

Servermanagement & Superuser-Funktionen (1 PW): Die GUI sollte einem Benutzer die Möglichkeit bieten, die zur Verfügung stehenden Server zu verwalten und anzusprechen (zu diesem Zeitpunkt passierte war dies noch über Textdateien möglich). Ferner sollten die von der Netzwerkgruppe implementierten Superuser-Funktionen (siehe dort) integriert und grafisch verfügbar gemacht werden.

- ▶ Serverliste verwalten
- • Superuser-Funktionen

Klickgraph (2-4 PW): Der Klickgraph ermöglicht es, eine Anwendung in der GUI als Prozessgraph zu modellieren. Er besteht mit dem Builder und dem Executor aus zwei Teilen:

Builder: Dieser Teil sollte in Zukunft das Erstellen einer vollständigen Application ermöglichen. Hierzu fehlten noch einige wichtige Funktionen, wie das Einstellen von Parametern, sowie das Verbinden von Kanälen mittels einer Kante. Zudem sollten Fehler, die beim „Zusammenklicken“ aufgetreten sind, entsprechend verdeutlicht werden.

Als optionales Feature sollte Brick um den Netbeans XML-Editor erweitert werden, welcher das Bearbeiten von Applications „per Hand“ ermöglicht.

- ▶ Kanten
- ▶ Parameter
- ▶ Fehleranzeige
- • XML-Editor einbinden
- • Undo/Redo

Executor: Die elementare Funktionalität des Executors war in der aktuellen GUI-Version schon gegeben. Wegen der bevorstehenden Änderungen am Framework waren jedoch einige Anpassungen vonnöten, wie z.B. die Umstellung von Hooked Channels auf GUI-Knoten. Zudem sollte zumindest eine elementare Steuerung einer Application möglich sein (mindestens Starten/Stoppen, im Optimalfall Pausieren/Fortsetzen). Um den aktuellen Status einer Application zu signalisieren, sollte eine Anzeige implementiert werden, welche für jeden Knoten zumindest angibt, ob er die Berechnung bereits abgeschlossen hat und ob bei seiner Ausführung Fehler aufgetreten sind.

- ▶ Interaktion mit Framework
- ▶ GUI-Knoten anzeigen
- ▶ Starten/Stoppen
- • Pausieren/Fortsetzen
- ▶ Status

- ▶ Fehlerausgabe

Visualisierung (2-n PW): Im Bereich der Visualisierung mussten zunächst die bestehenden Funktionen modularisiert werden. Dies beinhaltete zum einen eine Anpassung an die Netbeans RCP und ihren modularen Aufbau, zum anderen eine Kapselung dieser Integration, so dass eine Erweiterung basierend auf diesen Bausteinen ohne Kenntnisse ihrer inneren Funktionsweisen und der Netbeans-Plattform erfolgen konnte.

- ▶ Schnittstelle für RCP
- ▶ Schnittstelle für Erweiterungen
- Bausteine
  - Selektionswerkzeuge
  - Plot (Punkte/Objekte, Kurven, Geraden, Ebenen,...)
  - Datenmodell (effizientes Selektieren, Einschränken von Wertebereichen, ...)
  - Tabelle

Sonstiges: Es wurde angedacht, dass ein Plugin für *Aladin* nützlich wäre, mit dem sich das zu einem ausgewählten Datensatz zugehörige Objekt anzeigen ließe.

- Aladin Plugin/Skript

Netzwerk

Als Schnittstelle zwischen der GUI und dem Framework war die Netzwerk-Gruppe in viele der oben genannten Änderungen involviert und hatte dort einen umfangreichen Aufgabenbereich. Außerdem wurden folgende Ziele definiert:

Remote Commands (1 PW): Es sollte eine allgemeine Schnittstelle entwickelt werden, die es einem Client erlaubt, Knoten in einer laufenden Anwendung direkt anzusprechen (und bspw. manche ihrer Methoden aufzurufen). Dies sollte ebenfalls Datentransport vom Server zum Client ermöglichen, und so die Hooked Channels ersetzen.

Dynamisches Laden von JAR-Dateien (1 PW): Die erforderliche Funktionaltität, um Plugins zur Laufzeit aus .jar Dateien laden zu könnnen, sollte implementiert und auch das entfernte Laden vorbereitet werden.

Superuser-Funktionen (1 PW): Im Rahmen des Servermanagements mussten neben User- auch Superuser-Funktionen implementiert werden. Dabei handelt es sich um Funktionalitäten zur Verwaltung von Usern und Plugins.

Exception-Handling (2 PW): Siehe gleicher Eintrag in *Weiterentwicklung – Framework*.

Command Line Interface (CLI – 2 PW): Um den Server über Textkommandos zumindest grundlegend steuern zu können, sollte ein Command Line Interface realisiert werden, das die Ausführung von Superuser-Funktionen ermöglicht.

6.2.2 Erweiterung

Parallel zur Weiterentwicklung von BRICK sollten erste Beispielanwendungen realisiert werden. Dazu mussten Plugins geschrieben werden, die Daten aufbereiten, Data-Mining-Algorithmen implementieren oder eine Visualisierung der Ergebnisse ermöglichen. Diese und andere Aufgaben, die sich mit der Verwendung des Programmes beschäftigen, wurden von der Algorithmengruppe übernommen.

Vorbereitung

Module – Konzeptionierung (2 PW): Als erste mögliche Erweiterungen für das Framework wurden sogenannte Module angedacht, die von verschiedenen Knotentypen benötigte Funktionalitäten (z.B. Spaltenauswahl und Metrikverwaltung) kapseln sollten. Deren Konzeptionierung sollte angedacht und mit der Framework-Gruppe abgesprochen werden.

Module – Umsetzung (3 PW): Die angedachten Module sollten umgesetzt werden.

Verfahrensrecherche zur Quasardetektion über Spektraldaten (2 PW): Die Suche nach Quasaren erfolgt über die Spektraldaten. Mit einem Data-Mining-Algorithmus sollte die Identifikation von Quasaren anhand bestimmter Charakteristika (z.B. Anzahl der Peaks) maschinell erfolgen. Für diese Klassifizierung ist es erforderlich, die Anzahl der Dimensionen eines Spektrums (momentan bei SDSS etwa 2.000) zu reduzieren und die Spektren so zu bearbeiten, dass die entscheidenden Charakteristika erkannt und hervorgehoben sowie störende Informationen (z.B. Rauschen) entfernt werden.

Um diese Aufgaben erfüllen zu können, waren umfangreiche und auch fachübergereifende Recherchen nach möglichen Verfahren notwendig. Diese Verfahren galt es dann zusammen mit den Astronomen der Ruhr-Universität Bochum auf ihre Tauglichkeit zu analysieren. In dieser Arbeitsgruppe wurden im genannten Zeitraum auch die Charakteristika von Quasaren studiert und versucht, diese in ein für einen Algorithmus erkennbares Schema zu überführen.

Datenquellen

TXT-Datenquelle (2 PW): Neben CSV werden in der Astronomie auch andere Formate verwendet, um Daten in Textdateien abzulegen. Es sollte ein Knoten entworfen werden, der diese verschiedenen Formate auslesen und dem Framework als Datenquelle zur Verfügung stellen kann.

Datenaufbereitung

Normalisierer (3 PW): Sowohl zur Visualisierung als auch zur Verarbeitung der Daten ist es unter Umständen nötig, Spalten zu normieren/normalisieren und so vergleichbar zu machen. Dazu sollten einfache Verfahren sowohl als Modul als auch als Knoten umgesetzt werden.

Aufbereitung der Spektraldaten (3PW): Das Format der Spektraldaten in der SDSS-Datenbank war zur Weiterverarbeitung nicht geeignet: Die Intensitäten an den etwa 2.000 gemessenen Stellen des Spektrums wurden durch Kommata getrennt in einem einzelnen

String gespeichert. Außerdem mussten die Wellenlängen, zu denen diese Intensitäten gehören, erst mittels einer Formel berechnet werden. Aus diesen Informationen sollten je ein Array mit x- und ein weiteres mit y-Koordinaten entstehen, um so die Verarbeitung durch Brick zu ermöglichen.

Integrierer (2 PW): Der Integrierer sollte die etwa 2.000 Datenpunkte eines Spektrums reduzieren ohne es dabei zu stark zu verfälschen. Dazu zerlegt es den Messbereich auf verschiedene Arten in Intervalle und ermittelt für jedes Intervall einen repräsentativen Wert.

Sliding Window (2 PW): Der erste Schritt zur Entwicklung einer Pipeline zur Quasardetektion sollte darin bestehen, mit Hilfe eines Sliding-Window-Algorithmus zu den Spektren den jeweiligen Fluss zu bestimmen.

Data-Mining-Algorithmen

DBSCAN (3 PW): Es sollte der Clustering-Algorithmus *DBSCAN* sowie eine zugehörige Knotenklasse implementiert werden, um ihn in Anwendungen einbinden zu können. Zusätzlich sollte eine Autoparameterbestimmung implementiert werden, die einen der beiden Parameter ε oder *minPts* des Algorithmus automatisiert so wählt, dass eine vorgegebene Quote von klassifizierten Punkten zum Hintergrundrauschen erreicht wird.

k-Nächste-Nachbarn (3 PW): Auch dieser Algorithmus sollte implementiert und als Plugin in Brick eingebunden werden. Zusätzlich sollte eine Autoparameterbestimmung implementiert werden, die anhand der Trainingsmenge das k mit dem kleinsten Fehler ermittelt.

Visualisierung

Farbiger 2D-Plot (2 PW): Das erste Visualisierungsmodul sollte zwei vom Benutzer ausgewählte Spalten einer Tabelle als x- bzw. y-Koordinaten interpretieren und so jede Zeile als zweidimensionalen Punkt darstellen. Eine zusätzlich optional auswählbare Spalte sollte als Farbinfomation interpretiert werden, gemäß derer die Punkte eingefärbt werden.

6.3 Zeitplan

Im Folgenden wird ein zeitlicher Überblick über die Planung des Meilensteins gegeben. Dabei wird für jede Kalenderwoche angegeben, welche Ergebnisse bis dahin vorliegen und dann vorgestellt werden sollten.

KW 17	Beginn der Arbeit am Meilenstein 3.0
KW 18	Weiterentwicklung – Framework: *automatische Listenparameter* Weiterentwicklung – Netzwerk: *Remote Commands* Algorithmen: *DBSCAN*

KW 19

Weiterentwicklung – GUI:
- *Visualisierung: Schnittstelle für RCP*
- *Visualisierung: Schnittstelle für Erweiterungen*
- *Klickgraph: Builder*

Weiterentwicklung – Netzwerk: *Dynamisches Laden von JAR-Dateien*
Algorithmen: *Module – Konzeptionierung*
Algorithmen:
- *Normalisierer*
- *k-nächste-Nachbarn*
- *Integrierer*

KW 20

Weiterentwicklung – Framework: *GUI-Knoten*
Weiterentwicklung – Netzwerk: *Hinzufügen von Superuser-Funktionen*
Weiterentwicklung – Framework & Netzwerk: *Exception-Handling*

KW 21

Weiterentwicklung – Framework: *Threads*
Weiterentwicklung – GUI:
- *Basisfunktionalität*
- *Klickgraph: Executor*
- *Visualisierung*

Weiterentwicklung – Netzwerk: *CLI für Superuser-Funktionen*
Algorithmen:
- *Verfahrensrecherche und Umsetzungsanalyse zur Quasardetektion*
- *Module – Umsetzung*
- *Preprocessing der Spektraldaten*
- *TXT-Datenquelle*

6.4 Umsetzung & Ergebnisse

Das Ziel dieses Meilensteins war es, die bisher existente Alpha-Version des Programms in einen Beta-Status zu überführen. Der dazu nötige Aufwand wurde im Vorfeld unterschätzt, denn bei der Abarbeitung der einzelnen Aufgaben taten sich immer wieder kleine Lücken auf, die erst geschlossen werden mussten. Aus diesem Grund musste der Abschluss des Meilensteins um zwei Wochen nach hinten in Kalenderwoche 23 verschoben werden.

Da die Architektur des Programms zu diesem Zeitpunk bereits sehr weit gereift war, war es kaum mehr nötig neue bedeutende Konzepte zu erarbeiten. Die meiste Arbeit im Bereich der Weiterentwicklung bezog sich auf die Umsetzung bereits beschlossener Funktionalitäten oder Verbesserung der Bedienbarkeit. Die wenigen Ausnahmen sind gemeinsam mit einigen Ergebnissen der Algorithmengruppe hier aufgeführt.

6.4.1 GUI-Knoten

Das Konzept der *Hooked Channels* (siehe Abschnitt 5.2.1, Seite 43) erschien wenig zufriedenstellend, da es auf Ebene der Application-Center-Schnittstelle eine Sonderbehandlungen von Tabellendaten implementierte (andere Arten von Daten hätten über Hooked Channels nicht abgefragt werden können). Dies stand im Widerspruch zum grundlegenden Konzept, dass – um eine maximale Flexibilität zu gewährleisten – die Art der kommunizierten Daten dem Framework nicht bekannt sein sollte.

Hooked Channels wurden deswegen durch sog. *GUI-Knoten*, bzw. allgemeiner gefasst (da sie nicht nur über die GUI, sondern bspw. auch über das CLI abgefragt werden können), *RemoteSource*-Knoten ersetzt. Diese werden in der Application-Beschreibung explizit (aus Sicht des Clients – aus Sicht eines Benutzers des Application-Editors könnte dies auch implizit geschehen, wenn es automatisch durch den Application-Designer übernommen würde) platziert. Eine Kommunikation mit diesen Knoten wurde durch Verwendung von *Remote Commands* (siehe Abschnitt 6.4.3, Seite 54) implementiert. Eine Erweiterung dieses Konzepts auf andere Arten von Daten wurde zwar nicht implementiert, ist aber mit dem gleichen Konzept auf Plugin-Ebene, ohne weitere Veränderungen am Application Center, möglich.

6.4.2 Servermanagement

Die Verwaltung von Benutzern und Plugins erforderte ein neues Konzept, das während dieses Meilensteins nur angedacht, aber nicht vollständig entwickelt oder umgesetzt wurde. Die letztendliche Lösung waren *Environments* und wurden im nächsten Meilenstein implementiert (siehe Abschnitt 7.3.1, Seite 62).

6.4.3 Remote-Commands

Zu diesem Zeitpunkt war die Client-Server-Architektur so angelegt, dass einzelne Anfragen des Clients an Daten, welche auf dem Server erzeugt wurden, über das gleiche Interface geregelt wurden, über das auch Anwendungen importiert und verwaltet wurden. Dies erforderte eine Sonderbehandlung von Tabellendaten auf der Ebene der Serverschnittstelle. Im Zuge des Meilensteins 3.0 wurde beschlossen, dass dieser Datenaustausch ein eigenes Konzept der Kommunikation erhalten soll.

Zu diesem Zweck wurde ein allgemeinerer Ansatz entwickelt: Vom Client aus sollten über die Serverschnittstelle Kommandos in Form von Strings an einen bestimmten Knoten oder an alle Knoten (*Broadcast*) einer Anwendung gesendet werden können. Diesen Kommandos können beliebige[1] Parameter mitgegeben werden, die Knoten können für ein solches Kommando außerdem einen Rückgabewert liefern. Mit diesem Konzept konnten dann unter anderem auch solche Befehle implementiert werden, welche – über den Rückgabewert – den Abruf von auf dem Server erzeugten Daten ermöglichen. Mittels eines Broadcast-Kommandos können zuvor alle Knoten, die Daten für den Client bereitstellen, identifiziert werden.

6.4.4 Module

Bereits während Meilenstein 2.0 wurde mit k-Means ein erster einfacher Algorithmus in BRICK integriert. Mit den daraus gewonnenen Erfahrungen wurde schnell klar, dass es Funktionalitäten gibt, die über verschiedene Knoten hinweg (beinahe) identisch verwendet werden. Beispiele dazu sind die Auswahl der Spalten, auf denen ein Algorithmus arbeiten soll und das Berechnen des Abstandes zweier Datenpunkte, wie es für viele Data-Mining-Algorithmen nötig ist.

Es wurde also eine Schnittstelle entworfen, welche es erlaubt, diese Funktionalitäten in *Modulen* zu kapseln. Dabei wurde neben der Umsetzung der funktionalen Ansprüche insbesondere auch die Verwaltung der dazugehörigen Parameter über entsprechende Berichte innerhalb des

[1] Einzige Anforderung ist – aus naheliegenden Gründen – die Serialisierbarkeit der Parametertypen. Dies wird im übrigen auch vom Rückgabetyp gefordert.

Moduls möglich gemacht. Dies bewirkt, dass ein Modul seine Parametrisierung durch Benutzerinteraktion vor dem Programmierer eines Knotens verbergen und so mit einem minimalen Aufwand eingebunden werden kann. Die Metrikauswahl stellt ein solches Modul dar.

Eine besondere Form dieser Schnittstelle wurde für die Arbeit auf tabellenartigen Daten entworfen. Solche Module sollen zeilenweise Daten manipulieren und so z.B. eine Spaltenauswahl oder Normalisierung von Werten realisieren (beides wurde umgesetzt). Um diese Module möglichst komfortabel nutzen zu können, wurde zwei weitere Mechanismen umgesetzt: Eine automatische Kaskadierung vom Programmierer ausgewählter Module erlaubt es, diese hintereinanderzuschalten, um so auch komplexere Manipulationen zu erlauben und eine Maskierung ermöglicht, den Originalinhalt einer Zeile wiederherzustellen und so zu verhindern, dass die Änderungen beim Schreiben einer verarbeiteten Zeile permanent werden.

6.4.5 Spektren: Datenaufbereitung und Datenstruktur

Die Anforderungsanalyse der Quasardetektion bestand einerseits aus der Recherche möglicher Verfahren und der Entwicklung einer entsprechenden Pipeline zur Bewältigung dieser Aufgabenstellung sowie andererseits der Analyse der Spektraldaten des SDSS-Kataloges. Im Gegensatz zu den anderen, für Data-Mining-Algorithmen verwendbaren, Daten (Zahlen, eine je Zelle) liegen die Intensitäten eines Spektrums dort als durch Kommata getrennter String vor. Die entsprechenden Wellenlängen zu diesen Intensitäten müssen anhand einer Formel berechnet werden, deren zwei Koeffizienten von Datensatz zu Datensatz abweichen können.

Um diese Daten für die verschiedenen Algorithmen von Brick zur Verfügung stellen zu können, entschied man sich zunächst dafür, die Wellenlängen in ein einheitliches Raster zu überführen und anschließend für jede eine einzelne Spalte an den jeweiligen Datensatz anzufügen. Der Wert einer Zelle entspräche dann dem Ergebnis einer Interpolation von Punkten, die aus den Koeffizienten und dem String erzeugt wurden.

Nachdem die Recherche verschiedener Verfahren zur Quasardetektion insbesondere in Bezug auf die Peakanalyse erste Ergebnisse geliefert hatte, fiel auf dass durch eine feste Wahl des Rasters einzelne Peaks im Extremfall abgeschnitten oder zumindest stark verfälscht werden können. Daher wurde die Vereinheitlichung der Daten auf ein einheitliches Raster verworfen. Weiter fiel bei den ersten GUI-Tests auf, dass die Handhabung von etwa 2.000 zusätzlichen Spalten je Datensatz nicht zufriedenstellend ist. Zum einen schwindet die Übersicht über die vorhandenen Daten eines Datensatzes, zum anderen wäre eine manuelle Auswahl einer solchen Anzahl von Spalten weder wünschenswert noch zumutbar.

Die Projektgruppe einigte sich stattdessen darauf, die Wellenlängen und die Intensitäten eines Spektrums in je einem Float-Array zu speichern. Zwar wurde damit ausgeschlossen, dass die bisher implementierten Metriken (und damit auch die bisherigen Data-Mining-Algorithmen) direkt auf diesen Daten arbeiten können, andererseits wurde dadurch eine – in jedem Fall nötige – gesonderte Konzeption von Abstandsmaßen auf Spektren mit speziell darauf zugeschnittenen Verfahren motiviert.

7 Meilenstein 4.0: Fertigstellung

Der letzte Meilenstein der Projektgruppe setzte die Arbeit des vorherigen fort. In gleicher Gruppeneinteilung wurde auf der einen Seite Brick um neue Features erweitert und auf der anderen Seite mit weiteren Plugins ausgestattet. Selbstverständlich hätte diese Arbeit noch deutlich weiter geführt werden können und wurde nur wegen des nahenden Endes der Projektgruppenzeit abgeschlossen.

Der Abschluss dieses Meilensteins war die Vorstellung von Brick im DDH-Seminar des Lehrstuhls 11 am 13.07.2010.

7.1 Ziele

Wie zuvor ging der Fortsetzung der Arbeit an Brick eine Phase der Analyse voraus, die neue Aufgaben identifizierte. Nach über sechs Monaten Implementierungsarbeit, die nun langsam ein Ende finden sollte, standen allerdings auch „Aufräumarbeiten“ an. Es wurde nach unbenutzten Methoden, umständlichen Implementierungen und Möglichkeiten zur klareren Strukturierung (z.B. durch Änderungen der Vererbungshierarchie) gesucht. Zusätzlich zur Arbeit am Programm wurde die Präsentation für die abschließende Vorstellung entworfen.

Es folgt eine Auflistung der Ziele der einzelnen Gruppen, wobei **PW** wieder die Anzahl der zur Umsetzung geplanten Personenwochen angibt.

7.1.1 Weiterentwicklung

Framework

Die Weiterentwicklung des Frameworks konzentrierte sich hauptsächlich auf die Umsetzung eines schon länger geplanten Konzepts, den *Environments*. Aufgrund der zentralen Rolle von Environments sowie der Tatsache, dass es sich hier um ein neues und auch für den Benutzer sichtbares Feature handelte, musste in enger Abstimmung mit der Client/Server- und der GUI-Gruppe gearbeitet werden.

Environments (3 PW): Plugins sollen zum Zwecke der Verwendung von mehreren Versionen in Environments eingeteilt werden können. Ein Konzept soll erarbeitet werden und es soll möglich sein, die Environments in der GUI zu verwalten.

Command Line Interface (CLI – 2 PW): Bis zum Meilenstein 3.0 existierte nur ein Client für das Brick-Framework, der auf eine grafische Benutzeroberfläche setzt, was bedeutet, dass er beim Zusammenstellen und Ausführen von Anwendungen und dem Holen von Ergebnissen auf Benutzerinteraktion angewiesen ist. Zur automatisierten Ausführung von – unter Umständen auch automatisch generierten – Anwendungen ist er somit ungeeignet. Zu diesem Zweck sollte ein zustandsloses Kommandozeilenprogramm entwickelt werden, das für die oben genannten Aufgaben benutzt werden kann.

Variablen (1 PW): Es kann wünschenswert sein, die gleiche Anwendung mehrfach auszuführen, wobei einer oder mehrere der Parameter über die verschiedenen Ausführungen variiert werden. Dies wird bei BRICK nicht von Haus aus unterstützt, so dass hier Batch-/Shell-Skripte (in Verbindung mit dem CLI) das Mittel der Wahl sind. Es sollte eine Möglichkeit geschaffen werden, als Parameterwerte Variablen zu verwenden, welche bspw. bei der Ausführung den Wert von Umgebungsvariablen annehmen.

Ausweitung der Autoparameterfunktionalität auf ParameterReports (1 PW): Wie in Abschnitt 5.2.1 (siehe Seite 42) beschrieben, enthält ein ParameterReport den Wert des Parameters. Dazu war es bis zu diesem Zeitpunkt nötig, für alle Parameter den vom Benutzer gesetzten Wert in den entsprechenden Bericht zu schreiben. Durch Ausweitung der Autoparameterannotation sollte dies für Autoparameter vom Framework erledigt werden.

GUI

Probearbeiten (1 PW): Es wurden verschiedene Anwendungen erstellt und Arbeitsabläufe in der GUI ausgeführt, um dort sowohl Fehler zu finden als auch Verbesserungsvorschläge für eine bessere Handhabung zu formulieren.

Datenschnittstelle für Visualisierungsmodule (1 PW): Bis zu diesem Zeitpunkt mussten bei jeder Visualisierung die entsprechenden Daten vom Server zum Client transportiert werden, was bei mehrfacher Anzeige der gleichen Daten zu unnötigem Verkehr führte. Eine Datenverwaltung innerhalb der GUI sollte dies verhindern, indem sie Daten auf ein passendes Modell abbildet, welches es zudem ermöglicht Zusatzfunktionen einzubinden. Da es in BRICK keine Sonderrolle für bestimmte Datentypen gibt, sollte eine Schnittstelle entwickelt werden, welche es erlaubt Datenmodelle für unterschiedliche Daten einzubinden.

Datenmodell für tabellarische Daten (1 PW): Aufgrund der neuen Quellenhandhabung innerhalb der GUI wurde ein Modul benötigt, welches tabellarische Daten auf ein Modell abbildet und zudem die Zugriffe auf die entsprechende Quelle regelt. Das dabei entwickelte Modell sollte zudem noch die Möglichkeit bieten, Daten – bzw. Selektionen von Datenpunkten – zu manipulieren und dies an andere Visualisierungsmodule zu kommunizieren. Dies sollte z.B. erlauben, in einer Tabelle Datenpunkte zu selektieren, diese Selektion in einem Plot darzustellen und dort einige Punkte zu löschen, wobei die Tabellenansicht über diese Änderung informiert wird und die entsprechenden Zeilen nicht mehr anzeigt.

Anpassung / Weiterentwicklung bestehender Visualisierungen (2 PW): Die bereits existenten Plots sollten an die Änderung der Datenschnittstelle angepasst werden. Darüber hinaus sollten im 3D-Plot und der Tabellenansicht neben der Umsetzung einiger Funktionalitäten insbesondere an der Stabilität und Geschwindigkeit gearbeitet werden.

Integration des Servermanagements in die GUI (1 PW): Die Funktionen des im letzten Meilenstein fertig gestellten CLI für das Servermanagement sollten in die GUI integriert werden.

Entfernen von plattform-spezifischen Abhängigkeiten (1 PW): Der Netbeans-Unterbau wurde so modifiziert, dass die vorher benötigten nativen Bibliotheken nicht mehr im

Zielsystem installiert sein müssen, sondern mit BRICK ausgeliefert und automatisch bei Programmstart geladen werden können. Im Rahmen dessen wurde auch die zuvor benötigte Unterscheidung zwischen 32- und 64Bit-basierten Systemen entfernt.

k-d-Baum (2 PW): Zur Beschleunigung der Arbeit mit Plots (z.B. Löschen und Einfügen von Punkten), aber auch zur Verwendung in einigen Algorithmen sollte ein randomisierter k-d-Baum implementiert werden.

Netzwerk

Environments (3 PW): Siehe gleicher Eintrag in *Weiterentwicklung – Framework*.

Server als eigenständige Einheit extrahieren (1 PW): Der Server sollte als eigenständige Einheit zum Zweck der Installation an die Betreuer übergeben werden.

7.1.2 Erweiterung

Durch Absprachen mit den Astronomen der Ruhr-Universität Bochum entstanden verschiedene Anwendungsfälle, die wiederum ein deutlich größeres Angebot an Plugins benötigten. Ziel der Algorithmengruppe war es, davon möglichst viele zur Verfügung zu stellen.

Datenverwaltung

Spaltenfilter (2 PW): Während man bei SQL-Anfragen die Spalten der erzeugten Daten vorgeben kann, ist eine Auswahl bei anderen Quellen nicht so einfach möglich. Ein Spaltenfilter-Knoten sollte dies ebenso ermöglichen, wie das Hinzufügen von Spalten mit vorgegebenen Werten, um dadurch z.B. Daten aus bestimmten Quellen zu markieren.

Union/Join (2PW): Um Daten aus verschiedenen Quellen zusammen zu führen, sollte ein Knoten implementiert werden, der auf beliebig vielen Eingängen die Operationen *Union*, *Inner Join* und *Outer Join* ausführen kann.

Zeilenfilter (2 PW): Um eine Tabelle zeilenweise nach vorgegebenen Kriterien zu filtern, sollte ein Knoten entwickelt werden, mit dessen Hilfe boolesche Ausdrücke über den Tabellenspalten ausgewertet werden können. Dabei sollten zwei Varianten umgesetzt werden: entweder kann eine Ergebnisspalte angehängt werden, welche die Zeilen gemäß der Auswertung markiert, oder es werden nur die Zeilen ausgegeben, welche den Ausdruck erfüllen.

Datenaufbereitung

Konverter für Gradangaben (1 PW): Bei einem Winkel kann ein Teil eines Grads sowohl dezimal als auch in Minuten und Sekunden angegeben werden. Ein Modul soll die automatische Umrechnung ermöglichen.

Anpassung und Erweiterung des Integrators (1 PW): Der Integrator-Knoten sollte so angepasst werden, dass er nicht nur über Tabellenspalten integrieren sondern außerdem mit dem neuen internen Array-basierten Spektraldatenformat arbeiten kann. Außerdem sollten zur Handhabung der Ausreißerwerte an den Rändern zwei Clippingverfahren eingefügt werden.

Spektrensubtrahierer (2 PW): Es sollte ein Modul implementiert werden, das zwei Spektren voneinander subtrahiert. Dabei sollten zunächst fehlende Punkte interpoliert werden, so dass anschließend eine simple Subtraktion der Intensitäten durchgeführt werden kann. Eine Anwendung hierfür ist die Rechnung *Spektrum − Fluss*, welche die Merkmalsextraktion vorbereitet.

Merkmalsextrahierer für Spektren – Modul (3 PW): Um die Leistung von Data-Mining-Algorithmen auf Spektren zu verbessern, sollte ein Modul zur Extraktion wichtiger Merkmale aus einem entsprechend vorbereiteten Spektrum implementiert werden. Anvisierte Merkmale waren dabei die Anzahl und Gesamtfläche positiver und negativer Peaks, sowie deren maximale Höhe und Breite.

Merkmalsextrahierer für Spektren – Knoten (1 PW): Die Module zur Flussbestimmung, Subtraktion und Merkmalsextraktion von Spektren sollten zu einem Knoten verbunden werden.

Data-Mining-Algorithmen

Arbeit an DBSCAN (1 PW): Der bereits implementierte DBSCAN-Algorithmus sollte umfassend auf verschiedene Datenmengen angewendet werden, um auf der einen Seite die Laufzeit und die Autoparameterbestimmung zu testen und auf der andern Seite zu ermitteln, inwieweit Brick momentan schon zur Analyse astronomischer Fragestellungen beitragen kann.

Hierarchisches Clustering (4 PW): Nach k-Means, kNN und DBSCAN sollte mit Hierarchischem Clustering ein viertes Data-Mining-Verfahren in das Framework integriert werden.

Visualisierung

Spektrenvisualisierung (2 PW): Der bereits existente 2D-Plot sollte für die Anzeige von Spektren modifiziert werden. Zu beachten war, dass die bisherigen Visualisierungsmodule immer alle Daten eines GUI-Knotens anzeigen konnten/sollten, während dies bei Spektren nicht sinnvoll ist, so dass zusätzlich eine Auswahl des anzuzeigenden Spektrums ermöglicht werden musste.

Matrix-Scatter-Plot (2 PW): Zusätzlich zum bestehenden 2D-Plot sollte ein *Matrix-Scatter-Plot* implementiert werden. Dieser sollte eine Matrix von 2D-Plots enthalten, in der jede mögliche Kombination von zwei Dimensionen eines Datensatzes visualisiert ist.

Data-Mining-Anwendungen

Quasaridentifikation (2 PW): Zu diesem Zeitpunkt sollten alle Knoten zur Verfügung stehen, die nötig sind, um für Objekte aus dem SDSS-Katalog aufgrund ihrer Spektraldaten zuverlässig entscheiden zu können, ob es sich um Quasare handelt. Dies sollte durchgeführt und ausgewertet werden.

7.2 Zeitplan

Im Folgenden wird ein zeitlicher Überblick über die Planung des Meilensteins gegeben. Dabei wird für jede Kalenderwoche angegeben, welche Ergebnisse bis dahin vorliegen und dann vorgestellt werden sollten. Von vornherein stand fest, dass die Quasaridentifikation durch Spektren nicht bis zur Vorstellung am 13.07.2010 abgeschlossen werden würde.

KW 24 | Beginn der Arbeit am Meilenstein 4.0

KW 25 | Weiterentwicklung – GUI:
- *Datenschnittstelle für Visualisierungsmodule*
- *Entfernen von plattform-spezifischen Abhängigkeiten*

Algorithmen:
- *Anpassung und Erweiterung des Integrators*
- *DBSCAN*

KW 26 | Weiterentwicklung – Framework: *CLI*
Weiterentwicklung – Netzwerk: *Integration des Servermanagements in die GUI*
Weiterentwicklung – GUI: *Datenmodell für tabellarische Daten*
Algorithmen:
- *Konverter für Gradangaben*
- *Spaltenfilter*
- *Spektrensubtrahierer*

KW 27 | Weiterentwicklung – Framework: *Variablen*
Weiterentwicklung – Framework & Netzwerk: *Environments*
Weiterentwicklung – GUI: *k-d-Baum*
Algorithmen:
- *Matrix-Scatter-Plot*
- *Zeilenfilter*
- *Merkmalsextrahierer für Spektren – Modul*

KW 28 | Weiterentwicklung – Framework: *Ausweitung der Autoparameterfunktionalität*
Weiterentwicklung – Netzwerk: *Server als eigenständige Einheit extrahieren*
Weiterentwicklung – GUI:
- *Probearbeiten*
- *Anpassung / Weiterentwicklung bestehender Visualisierungen*

Algorithmen:
- *Union/Join*
- *Merkmalsextrahierer für Spektren – Knoten*
- *Hierarchisches Clustering*

Vorstellung von Brick im DDH-Seminar des Lehrstuhls 11.

KW 29 | Algorithmen: *Spektrenvisualisierung*

KW 30 | Algorithmen: *Quasaridentifikation*

7.3 Umsetzung & Ergebnisse

Die Umsetzung dieses Meilensteins verlief bis auf eine Ausnahme reibungslos und konnte mit einer erfolgreichen Vorstellung am 13.07.2010 abgeschlossen werden. Besagte Ausnahme war der bedauerliche Unfall eines PG-Mitglieds – dies hat den Abschluss der Quasaridentifikation durch Spektren um seine Genesungszeit verschoben.

7.3.1 Environments

Mit *Environments* wurde ein bereits seit Meilenstein 3.0 bekanntes Problem (siehe Abschnitt 6.4.2, Seite 54) gelöst, welches hier zunächst beschrieben wird, bevor ein Überblick über das Konzept zu dessen Lösung folgt.

Problematik

Beim Entwurf des Frameworks wurde sehr viel Wert darauf gelegt, dass Anwendungen portabel sind, d.h. bequem zwischen verschiedenen Servern ausgetauscht werden können. Ein hierbei auftretendes grundsätzliches Problem sind verschiedene Konfigurationen bei verschiedenen Servern, insbesondere im Hinblick auf die vorhandenen Plugins.

Ein Angleichen der Konfiguration kann sich problematisch gestalten: Es ist denkbar, dass verschiedene Versionen des gleichen Plugins sich durchaus signifikant unterscheiden; etwa in Bezug auf die Parameter oder das Verhalten der Knoten. Da Plugins bei Brick als JAR-Dateien abgelegt werden, ist es nicht ohne Weiteres möglich, in einem Server mehrere Versionen eines Plugins bereitzustellen – hierbei kann es leicht zu Problemen durch gleiche Java-Klassennamen, welche aber unterschiedliche Implementierungen referenzieren *sollen*, kommen.

Die Möglichkeit, ein maximales Maß an Abwärtskompatibilität durch das Bereitstellen mehrerer Versionen des gleichen Plugins zu ermöglichen, war die Hauptmotivation hinter der Umsetzung des Environment-Konzepts.

Lösung

Vereinfacht ausgedrückt verwaltet ein Environment eine Liste von Paaren aus Pluginname und Versionsnummer. Eine Anwendung benötigt, gewissermaßen als „Laufzeitumgebung“, immer ein Environment. Dies gilt offensichtlich auch dann, wenn die Anwendung nur geprüft und nicht ausgeführt werden soll, da auch dazu ein Instanziieren der Knoten notwendig ist, welches wiederum erst von einem Environment ermöglicht wird. Der Anwendung stehen dann nur die Plugins des Environments zur Verfügung und auch nur in den dort spezifizierten Versionen.

Die Auswahl, in welchem Environment die Anwendungen geprüft oder ausgeführt wird, obliegt dem Benutzer. Da diese – im Gegensatz zu Anwendungen – auf dem Server gespeichert werden, ist es bei jeder vom Benutzer angestoßenen Aktion notwendig, das dazugehörige Environment zu spezifizieren.

Jeder Benutzer kann selbst eigene Environments erstellen, wobei jedes auf dem Server gespeicherte Environment genau einen Besitzer hat.[1] Eigene Environments können verändert und gelöscht werden, außerdem kann spezifiziert werden, ob sie von anderen Benutzern verwendet werden dürfen. Solche „fremden“ Environments dürfen für eigene Anwendungen verwendet, nicht jedoch verändert oder gar gelöscht werden. Allerdings können sie kopiert und dann den eigenen Anforderungen angepasst werden.

[1] Eine Ausnahme stellt das der Einfachheit halber immer vorhandene *Default-Environment* dar.

7.3.2 Datenschnittstelle innerhalb der GUI

Das bisherige Vorgehen beim Auslesen von Datenquellen zu Visualisierungszwecken sah vor, dass ein Visualisierungsmodul die Quelle, die es anzeigen soll, kennt und die komplette Datenverwaltung selbstständig übernimmt. Dies hatte insbesondere zur Folge, dass Daten bei mehrfacher Visualisierung, mehrfach übertragen werden mussten, was vor allem bei großen Datenmengen zu vermeiden ist. Zudem hatte diese Art der Datenhandhabung den Nachteil, dass Datenmanipulationen, aber auch Selektionen von z.B. Punkten in einem Koordinatenkreuz, anderen Modulen nicht mitgeteilt werden konnten.

Zur Lösung dieses Problems wurde eine Datenschnittstelle innerhalb der GUI entworfen, welche jede darstellbare Quelle einer Application verwaltet und ihre Daten zur Verfügung stellt. Hierbei wurde zudem darauf geachtet, dass Daten unterschiedlicher Art vorkommen können (z.B. Tabellen oder Bäume), was dazu führte, dass die Schnittstelle erlaubt, Module zur Visualisierung aller Datentypen einzubinden.

7.3.3 Hierarchisches Clustering

Als einer von vier Data-Mining-Algorithmen wurde in Brick das Hierarchische Clustering implementiert. In diesem Abschnitt wird es beschrieben, wobei hauptsächlich auf [22] und [25] zurückgegriffen wird.

Verfahren und Funktionsweise des Hierarchischen Clusterings

Die grundsätzliche Idee des Hierarchischen Clusterings ist es, aufgrund der geringsten Distanz zwischen zwei Clustern diese, je nach eingesetztem Verfahren, entweder zu vereinigen oder zu separieren. Das Hierarchische Clustering kann grundsätzlich in zwei Varianten implementiert werden. Die Arbeitsweise der ersten Variante, die des agglomerativen (bottom up) Verfahrens, sieht vor, dass zuerst jedes Objekt o_i aus dem Objektraum $O = \{o_1, ..., o_n\}$ jeweils einem Cluster C_i zugeordnet wird. Sukzessiv werden dann in jedem Iterationsschritt die beiden in Bezug auf die Interclusterdistanz „nächsten“ Cluster zusammengeführt. Im Gegensatz dazu geht das divisive (top down) Verfahren von *einem* Cluster aus, in dem alle Datenobjekte enthalten sind. In jedem Schritt wird das Cluster geteilt, welches den größten Durchmesser besitzt, wobei der Durchmesser anhand der beiden voneinander am weitesten entfernten Punkte im Cluster bestimmt wird. Beide Verfahren sind in Abbildung 7.1 dargestellt.

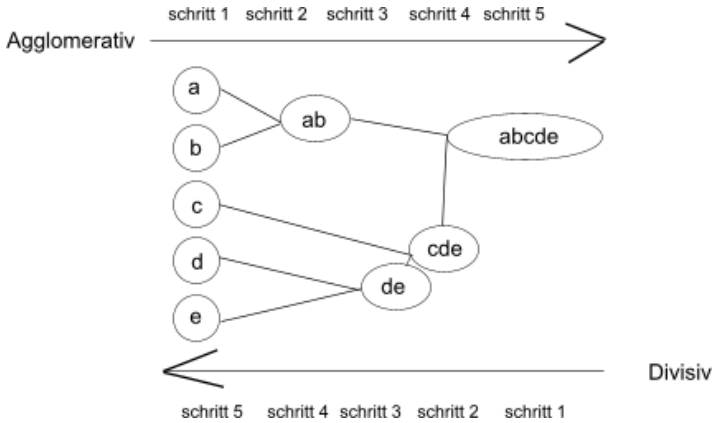


Abbildung 7.1: Agglomeratives und divisives hierarchisches Clustering

In der Praxis hat sich das agglomerative Verfahren gegenüber dem langsameren divisiven Verfahren, das in jedem Schritt die $2^{(k-1)} - 1$ möglichen Aufteilungen eines k-elementigen Clusters in zwei Subcluster untersucht, vor allem auf Grund der besseren Laufzeit durchgesetzt. Aufgrund dieses Vorteils wird in diesem Abschnitt hauptsächlich das agglomerative Verfahren erläutert.

Mathematische Struktur

Während des Hierarchischen Clusterings auf einer Objektmenge $\mathcal{O}$, wird eine Partition erzeugt:

Sei O eine Menge. Eine *Partition* (oder Zerlegung) von O ist eine Menge $\mathcal{P} = \{P_1, ..., P_n\}$ von Teilmengen von O (d.h. für alle $i \in \{1, ..., n\}$ gilt $P_i \subseteq O$), für die gilt:

- Alle Mengen aus $\mathcal{P}$ sind nicht-leer, d.h. $P_i \neq \emptyset$ für alle $i \in \{1, ..., n\}$.
- Alle Mengen aus $\mathcal{P}$ sind paarweise disjunkt, d.h. $P_i \cap P_j = \emptyset$ für alle $i, j \in \{1, ..., n\}$ mit $i \neq j$.
- Die Vereinigung der Mengen aus $\mathcal{P}$ ergibt O, d.h. $P_1 \cup P_2 \cup \cdots \cup P_n = O$

Eine Partition $\mathcal{P}$ ist in eine andere Partition $\mathcal{Q}$ genau dann *eingebettet*, wenn jedes Element aus $\mathcal{P}$ eine echte Teilmenge eines Elements von $\mathcal{Q}$ ist.

Das agglomerative Hierarchische Clustering erzeugt eine Sequenz von m ineinander eingebetteten Partitionen $\mathcal{P}_1, \ldots, \mathcal{P}_m$, wobei m die Anzahl der Objekte in der Gesamtmenge (Eingabe) ist. Dabei entspricht jedes Element einer Partition $\mathcal{P}_i$ einem Cluster, wobei die Cluster der Partition $\mathcal{P}_1$ jeweils ein Datenobjekt enthalten und die Partition $\mathcal{P}_m$ ein Cluster mit allen Datenobjekten aus dem Objektraum enthält. [35]

HACM-Algorithmus

Die Funktionsweise des Hierarchischen Clusterings wird in Algorithmus 1 mittels des grundsätzlichen **HACM**-Algorithmus (*Hierarchical Agglomerativ Clustering Methods*) vorgestellt. Die möglichen Distanzmaße werden im Anschluss im Abschnitt 7.3.3 erläutert.

Algorithmus 1 : Hierarchical Clustering

Eingabe : n Objekte
Ausgabe : Einordnung der n Objekte in Cluster

```
1 ordne jedes Objekt einem einelementigen Cluster zu
2 repeat
3 |   forall Cluster A do
4 |   |   forall Cluster B ≠ A do
5 |   |   |   bestimme die Interclusterdistanz zwischen A und B
6 |   verschmelze jenes Clusterpaar (A, B), dessen Interclusterdistanz minimal ist
7 until Abbruchbedingung erreicht ;
```

Abbruchkriterien

Die im Pseudo-Code auftretende Abbruchbedingung kann zum Beispiel eine der folgenden sein:

Anzahlkriterium / number criterion: Die Anzahl der Cluster wurde auf einen vorgegebenen Wert reduziert.

Distanzkriterium / distance criterion: Die – im Laufe des Algorithmus ansteigende – Interclusterdistanz der beiden zuletzt vereinigten Cluster überschreitet erstmalig einen vorgegebenen Schwellenwert.

Beispiel

In Abbildung 7.2 wird ein Beispiel betrachtet, welches das Single-Link-Distanzmaß, bei dem die Ähnlichkeit zwischen zwei Clustern (Teilmengen) durch die größte Ähnlichkeit bzw. die kürzeste Entfernung zwischen jeweils einem Datenobjekt der beiden Teilmengen definiert wird (siehe Seite 67), verwendet.

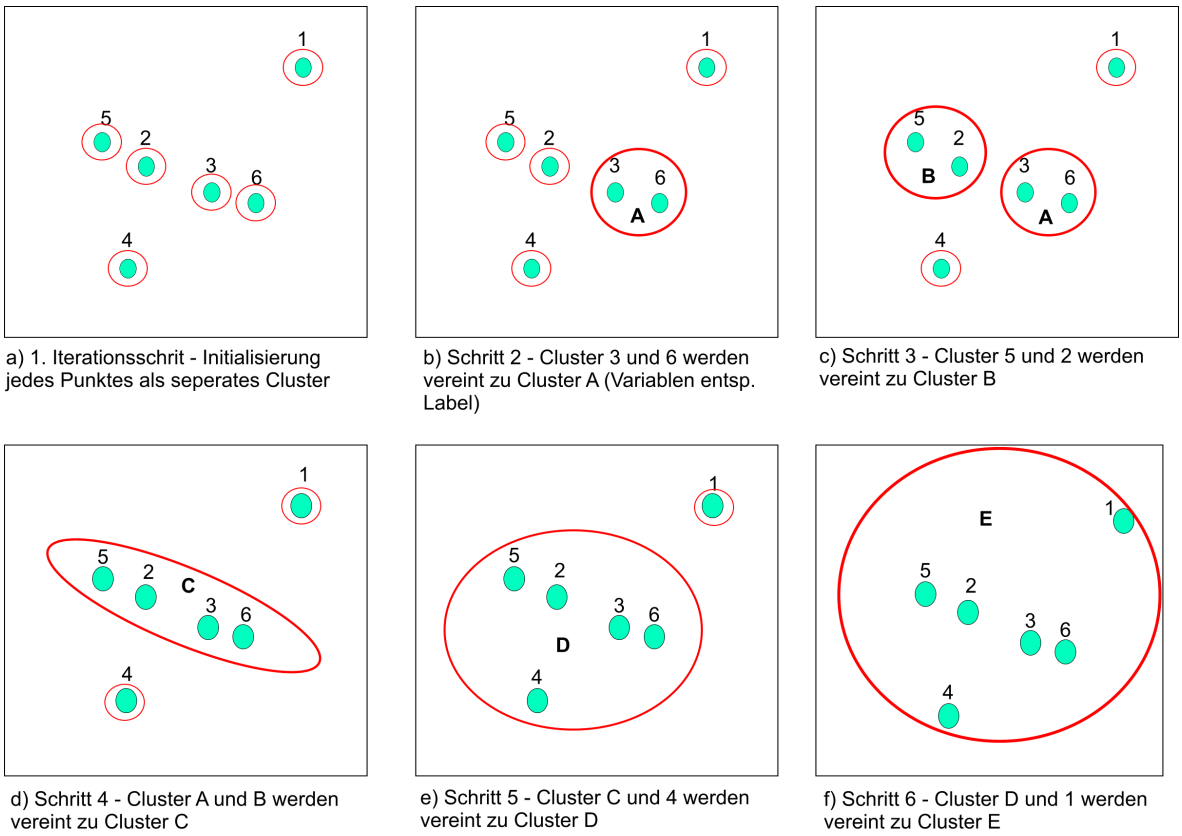


Abbildung 7.2: HACM-Beispiel mit Single-Link

Es ist zu erkennen, dass im ersten Schritt des Clusterings jeder Punkt mit einem separaten Cluster initialisiert und in jedem Schritt genau eine Vereinigung durchgeführt wird. Im zweiten Schritt werden die beiden sich nächsten Cluster zu A vereinigt. Im darauf folgenden, dritten Schritt werden die Cluster 5 und 2 zu B vereinigt. Der Algorithmus endet in diesem Fall, wenn alle Datenpunkte in einem Cluster vereinigt sind.

Eine Darstellung dieses Durchlaufs, in Form eines Dendogramms, ist in Abbildung 7.5a (siehe Seite 69) zu sehen.

Speicher- und Zeitkomplexität

Der HACM-Algorithmus kann in drei Varianten [31] implementiert werden, die sich in Bezug auf Speicherbedarf und Laufzeit sehr unterschiedlich verhalten können. Hier bezeichnet n die Anzahl der Datensätze in der Eingabe. Die Laufzeitangaben betrachten den Fall, dass der Algorithmus erst beendet wird, wenn alle Cluster vereinigt sind.

Verwendung der Datensätze: Die erste Möglichkeit den Algorithmus umzusetzen besteht darin, die Daten ohne vorherige Aufbereitung zu verwenden. In diesem Fall besitzt der Algorithmus eine Zeitkomplexität von $\mathcal{O}(n^3)$, die sich folgendermaßen zusammensetzt:

- Es finden n Clustervereinigungen statt.
- Die Auswahl der zu vereinigenden Cluster A und B benötigt $\mathcal{O}(n^2)$ Rechenschritte.

Der Speicherbedarf beträgt $\mathcal{O}(n)$.

Die Umsetzung des agglomerativen hierarchischen Algorithmus in BRICK wurde vorerst mit dieser Variante implementiert, was jedoch bei der Verarbeitung von Datensätzen mit einem Umfang, wie er für BRICK vorgesehen ist, zu untragbaren Laufzeiten führte. Als logische Konsequenz wurde nach einem Verfahren Ausschau gehalten, das die Laufzeit minimiert, wobei dieses Verfahren im Folgenden betrachtet wird.

Verwendung einer Distanzmatrix: Bei dieser Variante werden in einer Initialisierungsphase alle paarweisen Abstände der Datenpunkte zueinander berechnet und in einer Distanzmatrix gespeichert. Mit Hilfe dieser Matrix ist es dann nicht mehr notwendig, die Abstände der Objekte zueinander immer wieder (bedarfsorientiert) zu berechnen. Diese Aufbereitung der Daten hat jeweils eine Laufzeit und einen Speicherbedarf von $\mathcal{O}(n^2)$.

Da jedoch die n^2 Einträge der Distanzmatrix nach der kleinsten Distanz durchsucht werden müssen (Laufzeit $\mathcal{O}(n^2)$) und dieser Vorgang für n Vereinigungen zu wiederholen ist, wirkt sich diese Variante nicht positiv auf die asymptotische Gesamtlaufzeit aus, die somit bei $\mathcal{O}(n^3)$ bleibt.

Aus Praxissicht ist jedoch Folgendes zu konstatieren: Da die Matrix nur einmal berechnet wird und die Distanzen danach nur ausgelesen werden müssen, kann insbesondere bei hochdimensionalen Datenpunkten ein deutlicher Geschwindigkeitszuwachs gegenüber dem Verfahren ohne die Distanzmatrix verzeichnet werden. Es bleibt jedoch der „Nachgeschmack“, dass bei größeren Datenmengen der zusätzliche Speicherbedarf von $\mathcal{O}(n^2)$ ein Problem darstellen kann. Abschließend sei aus den genannten Gründen noch bemerkt, dass hierbei zwischen besserer Laufzeit und geringerem Speicherverbrauch abgewogen werden muss.

In BRICK wird diese Variante eingesetzt und bietet Freiraum für eigene Erweiterungen, die im nächsten Verfahren erläutert werden.

Verwendung einer sortierten Distanzmatrix: In diesem Fall wird die angelegte Distanzmatrix als Liste betrachtet und zunächst sortiert. Mit entsprechender Verwaltung dieser Datenstruktur während des Algorithmus kann die Suche nach dem Datenpaar mit minimalem Abstand in konstanter Zeit erfolgen. Die Laufzeit dieser Variante ergibt sich aus folgender Betrachtung:

- Anlegen der Distanzmatrix: $\mathcal{O}(n^2)$
- Sortieren der Distanzmatrix: $\mathcal{O}(n^2 log(n^2))$
- Weiterhin werden n Clustervereinigungen durchgeführt.
- Die Auswahl der zu vereinigenden Cluster bzw. die Bestimmung des kleinsten Wertes ($\mathcal{O}(1)$) sowie die Verwaltung der Distanzmatrix ($\mathcal{O}(n)$) kann zusammen in Zeit $\mathcal{O}(n)$ erfolgen.

Somit beträgt die Gesamtlaufzeit: $\mathcal{O}(n^2) + O(n^2 \log(n^2)) + n \cdot O(n) = O(n^2 \log(n^2))$.

Distanzmaße

In diesem Abschnitt werden die drei Verfahren *Single-Link*, *Complete-Link* und *Average-Link* vorgestellt, nach denen der HACM-Algorithmus implementiert werden kann. Diese drei Verfahren unterscheiden sich insbesondere dahingehend, dass sie sehr stark voneinander abweichende Ergebnisse bezüglich des grundsätzlichen Clusterings der Datensätze erzeugen. Dabei sei d ein auf der Objektmenge O definiertes Distanzmaß.

Single-Link

Unter den drei genannten Verfahren ist das Single-Link-Verfahren sowohl das bekannteste als auch das gebräuchlichste. Es berechnet die Distanz zwischen zwei Teilmengen A und B von O wie folgt:

$$d_{SL}(A, B) = \min\{d(a, b) \mid a \in A, b \in B\}$$

Diese Abstandsmessung ist in Abbildung 7.3 dargestellt.

Nach dem Single-Link-Verfahren werden also jene Cluster vereinigt, welche jeweils ein Element des Datenpaares mit dem geringsten Abstand enthalten.

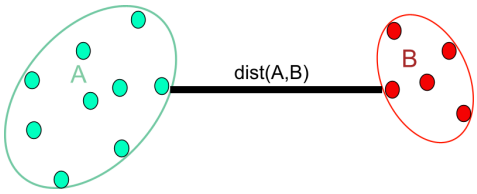
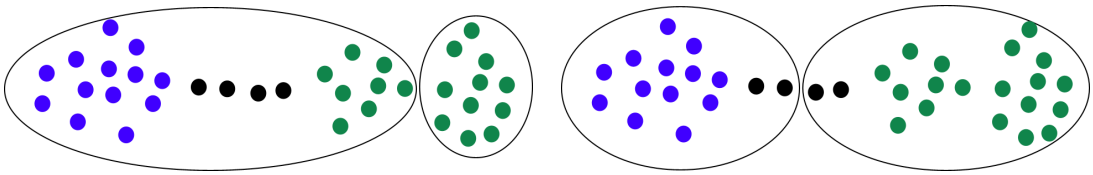


Abbildung 7.3: SingleLink

Trotz des Bekanntheitsgrades und des häufigen Einsatzes dieses Verfahrens in der Praxis sollte das potentielle Problem der *verketteten Cluster* (*chaining*) nicht vernachlässigt werden. Insbesondere bei größeren Datenmengen neigt dieser Ansatz zur Bildung sogenannter „kettenförmiger“ Cluster, da zum Vereinigen von zwei Clustern relativ wenig Datenobjekte notwendig sind. Dabei entstehende Cluster weisen häufig eine auffällig langgezogene Struktur auf (vergleiche Abbildung 7.4).

Nach [22] beträgt die Laufzeit $\mathcal{O}(n^2)$ und der Speicherbedarf $\mathcal{O}(n)$ zum Erzeugen einer hierarchischen Clusterstruktur, vorausgesetzt es wird der effiziente SLINK-Algorithmus, der von *Sibson* in [45] vorgestellt wurde, eingesetzt.



(a) Single-Link – Verkettungsproblem (b) Complete-Link (siehe unten)

Abbildung 7.4: HACM-Ergebnis bei drei intuitiven Clustern (Farben), die zu zwei Mengen (Kreise) zusammengefasst wurden.

Complete-Link

Beim *Complete-Link-Ansatz* wird nicht das nächste sondern das am weitesten entfernte Punktepaar betrachtet:

$$d_{CL}(A, B) = \max\{d(a, b) \mid a \in A, b \in B\}$$

Der Ansatz des Complete-Link-Verfahrens hat die selben Komplexitäten in Bezug auf die Laufzeit und den Speicherbedarf wie das Single-Link-Verfahren. Da es bei diesem Verfahren jedoch nicht möglich ist, zwei Cluster allein durch die beiden nächsten Datenobjekte zu vereinigen, tritt das Problem der Verkettung nicht auf und der Durchmesser der Cluster (also der größte Abstand zweier Punkte innerhalb des Clusters) wird verringert. Insgesamt entstehen dadurch eher Cluster ähnlicher Größe.

Average-Link

Bei diesem Verfahren wird der Abstand zwischen zwei Mengen A und B als der durchschnittliche Abstand zwischen den einzelnen Datenobjekten berechnet.

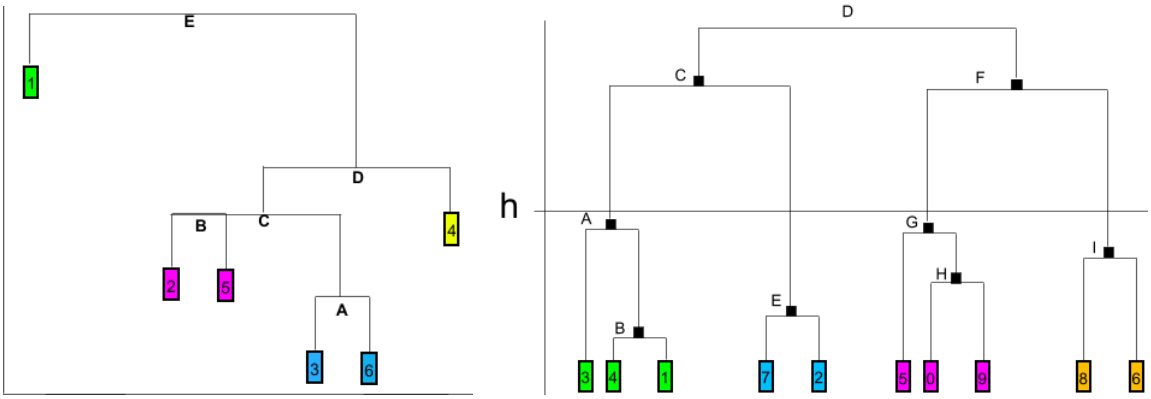
$$d_{AL}(A, B) = \frac{1}{|A| \cdot |B|} \sum_{a \in A, b \in B} d(a, b)$$

Dieses Verfahren erzeugt eine Struktur, die „zwischen“ derjenigen des Single-Link- und des Complete-Link-Verfahrens liegt. Es wird in der Praxis selten eingesetzt, da zur Bestimmung der Interclusterdistanz viele Abstände zwischen Punkten betrachtet werden müssen, was einen Laufzeitnachteil gegenüber den anderen beiden Varianten, bei denen nur jeweils ein Abstand betrachtet wird, bedeutet.

Repräsentation der Lösung

Die Lösung des hierarchischen Clusterings wird in der Regel durch eine Baumstruktur, ein sogenanntes *Dendogramm* dargestellt. Die Blätter entsprechen den Clustern mit jeweils einem Datenobjekt. Ein Knoten im Inneren des Baums entspricht einem Cluster, der die Vereinigung der beiden Kindknoten (Cluster) darstellt, die wiederum selbst Datenobjekte enthalten. Die Wurzel des Dendogramms repräsentiert den gesamten Objektraum.

Um ein Ergebnis mit einer festen Anzahl von Clustern zu erhalten, kann im Dendogramm auf einer beliebigen Ebene h ein Schnitt durchgeführt werden, der zu einem Wald führt, wobei jeder Baum ein Cluster definiert. Dies ist in Abbildung 7.5b veranschaulicht.

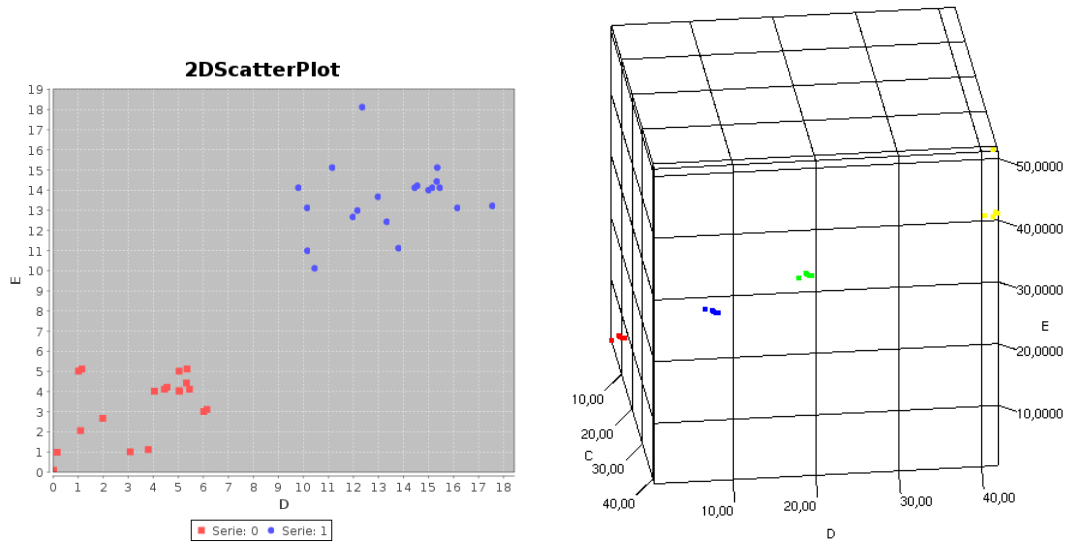


(a) Dendogramm zum Fallbeispiel (b) Dendogramm mit Schnittebene h

Abbildung 7.5: Darstellung zweier HACM-Durchläufe als Dendogramm

Beispiele aus Brick – Hierarchisches Clustering

In Abbildung 7.6 sind zwei Ergebnisse des in Brick realisierten HACM-Algorithmus zu sehen, die mit den bisher integrierten Modulen *2DScatterPlot* bzw. *3DScatterPlot* visualisiert wurden.



(a) Single-Link, 2D-Plot, 2 Cluster (b) Single-Link, 3D-Plot, 4 Cluster

Abbildung 7.6: Agglomeratives Clustering in Brick

7.3.4 k-Nächste-Nachbarn

Die k-Nächste-Nachbarn-Klassifikation, oder kurz kNN-Klassifikation, gehört zu den fundamentalsten und einfachsten Klassifikationsmethoden und bietet eine der besten Erkennungsraten, auch wenn es nur geringe Vorkenntnisse über die Verteilung der Daten gibt (vgl. [26]). In den folgenden Abschnitten wird der kNN-Algorithmus, sowie eine dazugehörige Beispielapp-

lication in BRICK beschrieben. Es sei schon einmal vorweg genommen, dass der ursprüngliche Algorithmus um die Möglichkeit erweitert wurde, den Benutzer frei wählen zu lassen, ob er eine bestimmte Anzahl an „Nachbarn“ vorgeben möchte, oder ob das „beste“ k automatisch bestimmt werden soll.

Algorithmus

Die Aufgabe des kNN-Algorithmus besteht darin, die Eingabedaten (semi-)automatisch so zu klassifizieren, dass Objekte in gleichen Gruppen möglichst ähnlich und Objekte aus verschiedenen Gruppen möglichst unähnlich zueinander sind. Zur Klassifizierung eines Objekts verwendet der kNN-Algorithmus die k nächsten Nachbarn innerhalb einer vorklassifizierten Trainingsmenge.

Algorithmus 2 beschreibt das fundamentale Vorgehen von kNN anhand von Pseudocode (vgl.[59]). Zu beachten ist, dass in diesem Durchlauf nur ein Objekt klassifiziert wird. Bei größeren Mengen muss dieses Verfahren iteriert werden.

Algorithmus 2 : kNN

Eingabe : Trainingsmenge D, Menge der Cluster L, zu klassifizierendes Datum z
Ausgabe : das Cluster $c_z \in L$, dem z zugeordnet wird

1 **forall** $y \in D$ **do**
2 berechne die Distanz $d(z, y)$ zwischen z und y
3 wähle die Menge $N_z \subseteq D$ mit den k Daten, die z am nächsten sind
4 ordne z dem Cluster zu, das in N_z am häufigsten vorkommt

Für ein zu klassifizierendes Objekt $y \in R^n$ werden zunächst mit Hilfe einer beliebigen Distanzfunktion, z.B. dem Euklidischen Abstand oder der Manhatten-Metrik, die Abstände zu allen Trainingspunkten berechnet. Anhand einer Mehrheitsentscheidung, an der sich die k nächsten Nachbarn beteiligen, wird y dann jener Klasse zugewiesen, welche die meisten Vertreter unter dieser Nachbarschaft hat.

Die Lernphase beim kNN-Algoritmus besteht aus dem Abspeichern der Trainingsmenge. Dieser Vorgang wird auch *lazy learning* genannt, da keine weiteren vorbereitenden Schritte unternommen werden.

Wahl von k

Die richtige Wahl von k spielt beim kNN-Algorithmus eine wichtige Rolle. Die Probleme, die bei der Wahl von k auftreten können werden im Folgenden beschrieben.

kleines k: Ein zu klein gewähltes k führt dazu, dass sich Ausreißer und Fehlklassifizierungen in der Traningsmenge sehr stark auf die Klassifikation auswirken. Wird $k = 1$ gewählt, erhält jeder Punkt die gleiche Klassifizierung wie sein nächster Nachbar in der Trainingsmenge.

großes k: Die Wahl eines zu großen k kann dazu führen, dass Punkte mit einem sehr großen Abstand zum klassifizierenden Punkt x betrachtet werden, das bedeutet, dass viele Objekte aus anderen Klassen mit in die Entscheidung einfließen.

Es besteht die Möglichkeit, durch mehrere Durchläufe *(runs)* die *beste Wahl* von k zu bestimmen, dabei wird bei jedem Durchlauf eine beliebige Anzahl von k nächsten Nachbarn getestet.

Automatische Bestimmung von k

Der in BRICK implementierte kNN-Algorithmus besitzt ein verbessertes Verfahren zur automatischen Bestimmung des Parameters k. Damit k automatisch bestimmt werden kann, müssen vorklassifizierte Daten vorliegen, um so pro Durchlauf für jedes k eine Qualitätsmessung durchzuführen.

Für die Qualitätsmessung werden die Daten zweimal geteilt. Die erste Teilung erfolgt auf der Gesamtmenge der vorklassifizierten Daten. Damit erhält man zwei gleich große Teilmengen, die Trainingsdaten und die Testdaten. Die Trainingsdaten werden wiederum in die Trainingsmenge und eine Validierungsmenge geteilt (vgl. Abbildung 7.7). Zur Aufteilung der Trainingsdaten wurde hier die Split-Sample-Methode verwendet ([56]). Hierbei erfolgt die Aufteilung nach einem Aufteilungsprozentsatz p, wobei in der Implementierung $p = 1/2$ gewählt wurde. Die Datenmenge der Trainingsmenge muss so groß sein, dass sie bis auf kleine Fehler die wesentlichen Eigenschaften der Gesamtmenge enthält. (siehe [56]).

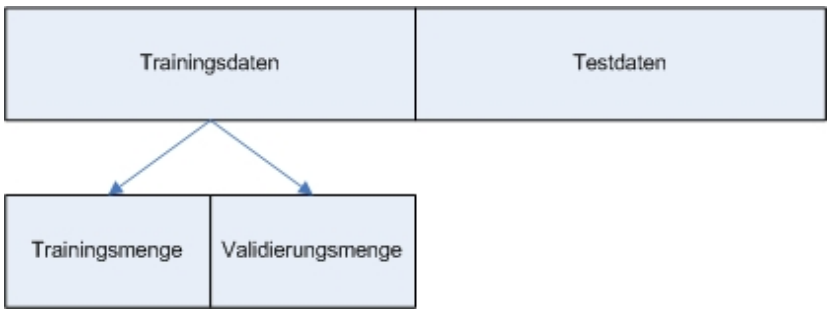


Abbildung 7.7: Aufteilung der Daten

Der kNN-Algorithmus wird dann auf der Trainingsmenge (die er tatsächlich als Trainingsmenge verwendet) und den Testdaten (die er neu klassifiziert) ausgeführt. Anschließend wird über den Testdaten das *Summarische Fehlermaß* berechnet, indem die Anzahl der falschen Klassifikationen durch die Anzahl aller Daten geteilt wird. Dieser Wert wird für jedes gewünschte k ermittelt, womit zum Schluss jenes mit dem kleinsten Fehler bestimmt werden kann. Der ebenso berechnete Fehler der Validierungsmenge dient zur Abschätzung der Generalisierungsfähigkeit des gewählten Klassifikators.

Mit der Auswahl von *autok* legt der Benutzer fest, ob k in der Klassifikation automatisch bestimmt werden soll, wobei dann ein maximales k angegeben werden muss. Der Startwert für die Klassifikation liegt bei $k = 1$. Die Anzahl der Durchläufe *(runs)*, in denen das k von 1 bis k ausgeführt wird, wobei für jeden Run die Daten zufällig neu geteilt werden, wird ebenfalls durch den Benutzer festgelegt. Nachdem diese Einstellungen vorgenommen wurden, wird der Algorithmus für jedes k entsprechend oft durchlaufen.

Laufzeit und praktische Probleme

Die Laufzeit des kNN-Algorithmus auf einer Trainingmenge T und einer Datenmenge D kann durch $\mathcal{O}(|D| \cdot |T| \cdot \Delta)$ abgeschätzt werden, wobei Δ den Aufwand der Abstandsberechnung

angibt, der direkt von der Dimension der Daten abhängt.

Für kleine Dimensionen ist der kNN-Algorithmus sehr effizient. Ein praktisches Problem ist jedoch der Speicher- und Rechenaufwand bei hochdimensionalen Räumen und vielen Trainingsdaten. Des Weiteren entsteht noch das Problem, dass beim Hinzufügen weiterer Dimensionen in einen mathematischen Raum dieser vom Volumen her größer wird und sich die Objekte darin verlieren können. Dies wird auch als *Fluch der Dimensionalität* bezeichnet [32].

Eine Verbesserung des kNN-Algorithmus kann durch k-d-Bäume erfolgen, jedoch bleibt die Berechnung bei großen Datenmengen aufwändig. Ein k-d-Baum ist eine Verallgemeinerung des binären Suchbaums und wird zur Darstellung von Punkten in einem k-dimensionalen Raum verwendet. Die Idee ist, die Punktmenge auf jeder Tiefe anhand einer anderen Dimension zu teilen. Geteilt wird immer an einem Punkt (Median) ([60]). k-d-Bäume dienen zur effizienten Speicherung von k-dimensionalen Punkten.

Anwendungsbeispiele

Als Anwendungsbeispiel aus astronomischer Sicht sei z.B. die automatische Klassifikation von Objekten in Sterne, Galaxien und Quasare genannt. Der kNN-Algorithmus ist auf alle Daten anwendbar, die sich in Klassen einteilen lassen.

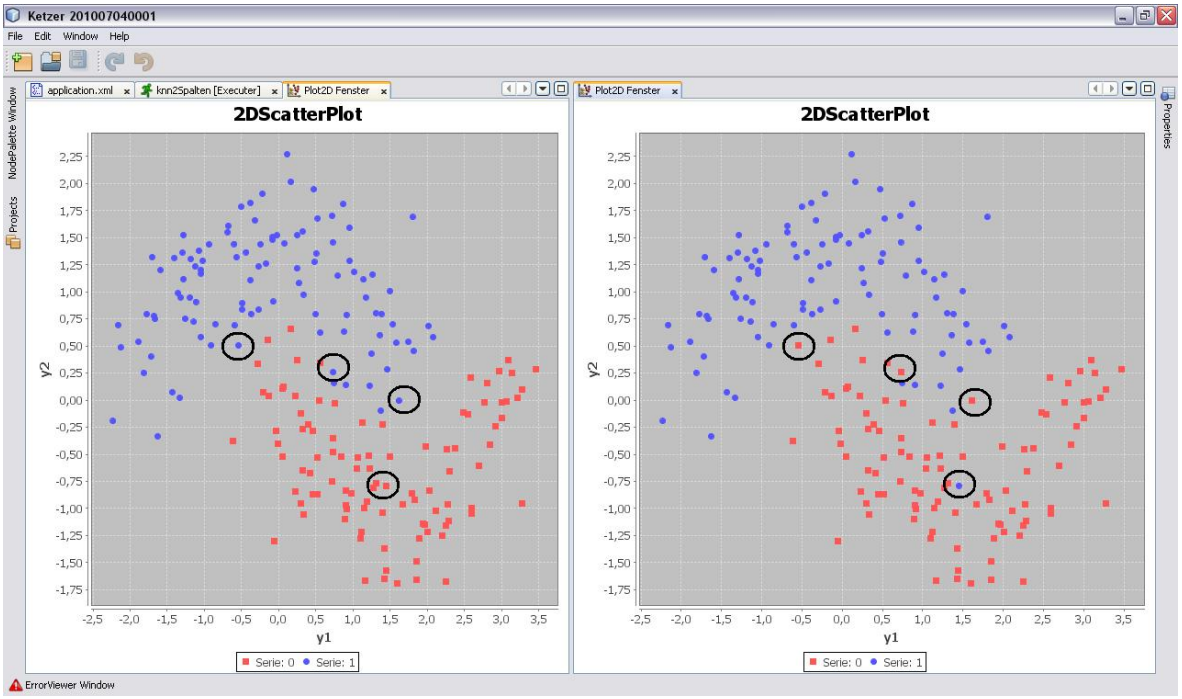


Abbildung 7.8: Ergebnisse der Klassifikation

Die Ergebnisse des in Brick realisierten kNN-Algorithmus sind in Abbildung 7.8 zu sehen. In dem linken 2DScatterPlot sieht man die Cluster-Zuordnung mit den Orginaldaten. Im rechten Plot die Klassifizierung des kNN-Algorithmus. Die vier falsch klassifizierten der insgesamt 200 Objekte wurden eingekreist.

7.3.5 Dichte-basiertes Clusteringverfahren mit DBSCAN

Eines der Hauptziele der Projektgruppe war es, durch Einsatz von bekannten Algorithmen des Data Mining innerhalb von Brick, den Astronomen Werkzeuge zur Verfügung zu stellen, um die Analyse großer Datenmengen zu ermöglichen. Für das Ende der Entwicklungszeit war unter anderem der Einsatz des *DBSCAN-Algorithmus* auf den Daten des öffentlich verfügbaren SDSS-Katalogs vorgesehen. Der Zugriff auf die Daten des SDSS-Katalogs sollte über SQL-Abfragen erfolgen.

Für Astronomen ist es von Interesse, bei bereits in Datenbanken eingetragenen Objekten zu prüfen, wie dicht diese beieinander liegen, um so z.B. Sternhaufen finden zu können.

Im Folgenden werden die Grundlagen des in Brick integrierten *DBSCAN-Algorithmus* vorgestellt.

Dichte-basiertes Clustering

Der Algorithmus DBSCAN (Abk. für „Density-Based Spatial Clustering of Applications with Noise“ [29]) ist ein Verfahren, das eine Dichte-basierte Analyse der Daten durchführt und Cluster als Ergebnis liefert.

Die Grundidee dieses Algorithmus liegt in der sogenannten *Dichteverbundenheit* der Objekte. DBSCAN versucht, Objekte, die weniger dicht beieinander liegen, von den dicht beieinander liegenden Objekten getrennt zu betrachten. Dabei werden einerseits aufgrund der vorher festgesetzten Dichteparameter verschieden große Cluster gebildet, und andererseits wird eine Gesamtmenge der rar bestückten Gebiete als Rauschen definiert. Ein Cluster besteht dabei aus einer Menge von Objekten, deren jeweilige lokale Punktdichte innerhalb des Clusters einen vorher festgesetzten Grenzwert nicht unterschreitet. Die lokale Punktdichte eines Objektes wird durch die Anzahl der sogenannten Nachbarschaftsobjekte in dessen vorher festgelegter Nachbarschaftsumgebung bestimmt.

Um eine formale Beschreibung des dichtebasierten Clusters und des Rauschens zu formulieren, sind folgende Definitionen der Begriffe *Kernpunkt*, *Dichte-Erreichbarkeit* und *Dichte-Verbundenheit* erforderlich. Die hier verwendeten Definitionen stammen hauptsächlich aus [30].

Sei O eine Menge von Objekten und o ein dazugehöriges beliebiges Objekt. Der minimale Dichtegrenzwert wird durch folgende Parameter bestimmt:

- ε beschreibt den Nachbarschaftsradius
- *MinPts* gibt die minimale Anzahl der Objekte an, die innerhalb des Radius liegen müssen

Kernobjekt

Ein Objekt $o \in O$ ist ein *Kernobjekt*, falls gilt:

$$|N_\epsilon(o)| \geq MinPts, \text{ wobei } N_\epsilon(o) = \{o' \in O \mid dist(o, o') \leq \epsilon\}$$

Falls in der ε-Umgebung eines Objektes mindestens *MinPts* Objekte liegen, ist dies ein Kernobjekt (siehe Abbildung 7.9a). Jedes Kernobjekt befindet sich innerhalb eines Clusters. Falls ein Objekt kein Kernobjekt ist, ist es entweder ein Randobjekt eines Clusters oder es gehört zum Rauschen.

Direkte Dichte-Erreichbarkeit

Ein Objekt $q \in O$ ist *direkt Dichte-erreichbar* von $p \in O$ (bzgl. ε und *MinPts* in O, aber im Folgenden sind diese Parameter fix und werden nicht jedesmal erwähnt), falls gilt:

- p ist ein Kernobjekt in O
- $q \in N_\epsilon(p)$

Falls p ein Kernobjekt ist, sind also alle Objekte, die in der ε-Umgebung von p liegen, direkt Dichte-erreichbar von p (siehe Abbildung 7.9a).

Über die direkte Dichte-Erreichbarkeit lässt sich eine weitere Definition formulieren.

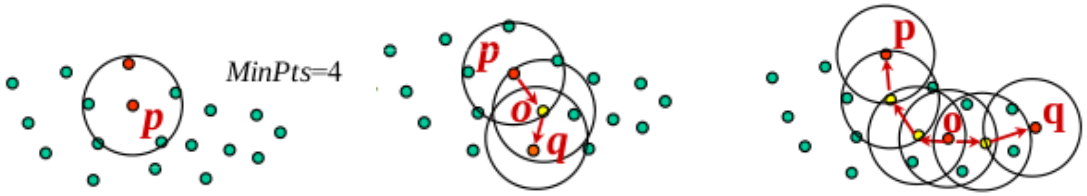
Dichte-Erreichbarkeit

Falls eine Folge von Objekten $p_1, \ldots, p_n$ existiert, so dass $p_1 = p$, $p_n = q$ ist und falls gilt: p_{i+1} ist direkt Dichte-erreichbar von p_i für $1 \leq i < n$, dann ist ein Objekt q Dichte-erreichbar von einem Objekt p.

Die *Dichte-Erreichbarkeit* ist die transitive Hülle der *direkten* Dichte-Erreichbarkeit. Einfach ausgedrückt ist ein Objekt q Dichte-erreichbar von p, wenn zwischen diesen beiden Punkten eine Kette von direkt Dichte-erreichbaren Punkten existiert (siehe Abbildung 7.9b). Diese Ketten sollen verständlicherweise Teil eines Clusters sein. Um die Verbundenheit dieser Ketten innerhalb eines Clusters zu beschreiben, ist eine weitere Definition nötig.

Dichte-Verbundenheit

Zwei Objekte p und q sind *Dichte-verbunden*, falls ein Objekt $o \in O$ existiert, so dass p und q Dichte-erreichbar von o sind (siehe Abbildung 7.9c).



(a) Kernpunkt und direkte Dichte-Erreichbarkeit (b) Dichte-Erreichbarkeit (c) Dichte-Verbundenheit

Abbildung 7.9: Beispiele zu Definitionen des Dichte-basierten Clusteringverfahrens [23]

Die obigen Definitionen der Dichte-Erreichbarkeit und der Dichte-Verbundenheit dienen der Formulierung von einzelnen Clustern, des gesamten Clustering und des Rauschens. Nun kann der Begriff eines dichtebasierten Clusters ebenfalls formal definiert werden.

Dichte-basierter Cluster

Ein Cluster C ist eine nicht leere Teilmenge von O für die gilt:

1. Maximalität:
   für alle $p \in C$, $q \in O$ gilt, dass wenn q Dichte-erreichbar von p ist, dann ist $q \in C$.
2. Verbundenheit:
   für alle $p, q \in C$ gilt , dass p Dichte-verbunden ist mit q.

Einfach ausgedrückt stellen die Mengen der Objekte, die alle miteinander Dichte-verbunden sind und alle Objekte, die von irgendeinem Kernpunkt des Clusters Dichte-erreichbar sind, einen Cluster dar. Aus den oberen Definitionen kann eine wesentliche Eigenschaft der Dichte-basierten Cluster subsumiert werden: Ausgehend von einem beliebigen Kernpunkt kann über die Dichte-erreichbaren Objekte ein gesamter Cluster gefunden werden. Dieser wird durch folgendes Lemma definiert [30].

Lemma 1. *Seien C ein Dichte-basierter Cluster und $p \in C$ ein Kernobjekt. Dann gilt:* $C = \{o \in O \mid o \text{ Dichte-erreichbar von } p\}$.

Dichte-basiertes Clustering und Rauschen

Es folgen zwei allgemeine Definitionen des Dichte-basierten Clustering und des Rauschens – diese beziehen sich immer auf die Parameter ε und*MinPts* und die Objektmenge O.

1. *Ein Dichte-basiertes Clustering CL* ist eine Menge von Dichte-basierten Clustern $CL = \{C_1, \ldots, C_k\}$ so dass für alle C gilt: wenn C ein Dichte-basierter Cluster ist, dann ist C in CL.

2. Falls $CL = \{C_1, \ldots, C_k\}$ ein Dichte-basiertes Clustering ist, dann ist die Menge des Rauschens, $Noise_{CL}$, definiert als Menge aller nicht zu einem Cluster C_i gehörenden Objekte.

Weiter sei angemerkt, dass sich bei größeren *MinPts*-Werten ($MinPts \geq 4$) Randpunkte der Dichte-basierten Cluster überlappen können. Folgendes Lemma verdeutlicht diesen Sachverhalt [30].

Lemma 2. *Falls CL ein Clustering von O ist, gilt für alle C_i, $C_j \in CL$:*

1. *Falls $C_i \neq C_j$ ist, gilt für alle $p \in C_i \cap C_j$, dass $|N_\epsilon(p)| < MinPts$ – d.h. kein p, das in zwei Clustern vorkommt, ist kein Kernobjekt.*

2. *Falls $MinPts \leq 3$ und $C_i \neq C_j$, dann ist $C_i \cap C_j = \emptyset$, was bedeutet, dass CL überlappungsfrei ist.*

Algorithmus DBSCAN

In Algorithmus 3 wird das Vorgehen von DBSCAN durch Pseudocode beschrieben. Die Menge der noch nicht besuchten Objekte (das sind zu Beginn alle) wird linear durchlaufen und für jedes davon wird versucht, einen Cluster zu finden. Hierzu wird zunächst geprüft, ob ein Objekt ein Kernobjekt ist. Falls es keines ist, wird es vorläufig als Rauschen gekennzeichnet, ansonsten wird die Funktion `expandiereCluster` aufgerufen.

Diese führt das in Lemma 1 entwickelte Verfahren aus. Dabei werden alle vom neuen Kernobjekt aus direkt Dichte-erreichbaren Objekte (also die innerhalb der ε-Umgebung) dem Cluster zugeordnet. Diese Punkte besitzen evtl. wiederum direkt Dichte-erreichbare Nachbarschaftsobjekte, wodurch diese vom Kernpunkt aus gesehen Dichte-erreichbar sind und somit ebenso zum Cluster zugeordnet werden. Eine wiederholte Berechnung der direkten Dichte-Erreichbarkeit errechnet den kompletten Cluster zum jeweiligen Kernpunkt.

Für ein Clustering benötigt DBSCAN nur zwei Parameter – ε und *MinPts*, wobei das Ergebnis bis auf die Zuordnung von Randpunkten unabhängig von der Reihenfolge der Daten

Algorithmus 3 : DBSCAN

Eingabe : Objektmenge O, Parameter ε und *MinPts*
Ausgabe : Clustering $CL = \{C_1, \dots C_c\}$

```
1 Clusterindex i = 0
2 forall o nicht besuchter Punkt aus O do
3     markiere o als besucht
4     N = Nachbarschaft(o, ε)
5     if |N| < MinPts then
6         markiere o als Rauschen
7     else
8         zähle i eins hoch
9         expandiereCluster(o, N, i, ε, MinPts)
```

Funktion `expandiereCluster`$(o, N, i, \varepsilon, MinPts)$

Eingabe : Objekt o, Nachbarschaft N, Clusterindex i, Parameter ε und *MinPts*
Ausgabe : Cluster C_i, das o als Kernobjekt enthält

```
1 füge o zum Cluster C_i hinzu
2 forall o' ∈ N do
3     if o' ist nicht besucht then
4         markiere o' als besucht
5         N' = Nachbarschaft(o', ε)
6         if |N'| ≥ MinPts then
7             N := N ∪ N'
8     if o' gehört noch nicht zu einem Cluster then
9         füge o' zum Cluster C_i hinzu
```

ist. Des Weiteren ist das Verfahren in der Lage, Rauschen zu erkennen. Verglichen mit anderen Clusteringverfahren, wie z.B. *k-Means*, benötigt DBSCAN im Voraus keine Informationen über die Anzahl der Cluster. DBSCAN ist einerseits in der Lage, willkürlich geformte Cluster zu entdecken, andererseits ermöglicht er durch den *MinPts*-Parameter, den Single-Link-Effekt (siehe Abschnitt 7.3.3, Seite 67) zu reduzieren – den Fall, in dem verschiedene Cluster durch eine dünne Linie von Objekten verbunden sind. Es existieren jedoch auch Fälle, in denen DBSCAN keine sinnvollen Ergebnisse liefern kann. Diese werden im Folgenden genannt [30]:

- Rauschen und Cluster sind nicht gut getrennt
- stark unterschiedliche Dichte in verschiedenen Bereichen des Raumes

Laufzeit

DBSCAN berechnet die ε-Nachbarschaft für jedes der n Objekte in der Objektmenge einmal. Falls für jedes Objekt die Nachbarschaftsbestimmung einzeln über die Bedingung $dist(p, o) \leq \epsilon$ geprüft wird, besitzt der gesamte Algorithmus eine quadratische Laufzeit $O(n^2)$. Bei der in Brick implementierten Version des Algorithmus wurde diese Variante eingesetzt.

Es gibt jedoch Indexstrukturen, wie z.B. R^*-*Bäume*, die in niedrig-dimensionalen Räumen eine Bereichsabfrage in einer logarithmischen Laufzeit garantieren, also $O(n \cdot \log n)$. Durch die Verwendung von *Gridfiles* ist der Zugriff auf die Nachbarschaftsumgebung sogar in konstanter Zeit möglich, was zu einer Laufzeit von $O(n)$ führt [30]. Dieser Aspekt könnte bei der zukünftigen Entwicklung von BRICK im Hinblick auf die Auswertung großer Datenmengen eine Rolle spielen.

Parameterbestimmung

Für jedes der Verfahren aus dem Bereich des Data Mining stellt die optimale Bestimmung der Parameter ein Problem dar. Im Falle von DBSCAN werden die Cluster gefunden, deren Dichte höher ist als die durch Parameter ε und *MinPts* definierte Grenzdichte. Eine automatische Bestimmung von guten Kandidaten für eine möglichst optimale Clusterbestimmung auf verschiedenen Objektmengen ist nur in bestimmten Fällen möglich. Hier bieten sich Heuristiken als mögliche Lösung an. Das bisher bekannteste Verfahren basiert auf dem Diagramm der sortierten kNN-Distanzen. Die Implementierung und Einsatz dieser Heuristik wäre evtl. für die zukünftige Verwendung von BRICK zu empfehlen. Hierzu sei auf die ausführliche Beschreibung in [30] verwiesen.

Als alternativer Ansatz für die Parameterbestimmung wurde jedoch eine *halbautomatische Parameterbestimmung* durch den Einsatz eines weiteren Parameters implementiert. Bei der Berechnung von Clustern aus einer gegebenen Datenmenge wird ebenfalls eine Teilmenge der Objekte, die zum Rauschen gehören, bestimmt. Das Verhältnis dieser Teilmenge zur Gesamtmenge wurde als *Rauschanteil* bezeichnet. Der neue Parameter *noiseRatio* gibt den sogenannten *Soll-Rauschanteil* innerhalb der Gesamtmenge an, der bei der Clusterbestimmung berücksichtigt wurde.

Beim aktivierten *noiseRatio* Parameter werden entweder ein oder beide Parameter ε und *MinPts* halbautomatisch bestimmt, indem jeweils der Minimum- und Maximum-Wert gesetzt wird und der Algorithmus anschließend so lange wiederholt ausgeführt wird, bis der angegebene Anteil des Rauschens bestmöglich angenähert ist. Die Parameter nehmen bei jeder Wiederholung neue Werte zwischen dem angegebenen Minimum und Maximum an.

DBSCAN im Rahmen des Meilensteins 4.0

In dieser Projektgruppenphase ging es hauptsächlich darum, letzte Anpassungen und die Optimierung des Algorithmus DBSCAN vorzunehmen. Ein weiteres Arbeitsgebiet innerhalb dieses Zeitraums war die Erstellung eines Anwendungsfalls.

Abbildung 7.10 zeigt das Ergebnis der DBSCAN-Abfrage mit dem Algorithmus DBSCAN dargestellt durch das *2DScatterPlot*-Modul des BRICK-Frameworks – ein Cluster, der einen bereits bekannten Sternhaufen identifiziert.

Für das Anwendungsbeispiel wurde ein Himmelsausschnitt, der einen bekannten Sternhaufen enthält, ausgewählt. Ziel war es, diesen Sternhaufen in den vom SDSS-Katalog gelieferten Daten mit Hilfe des DBSCAN-Algorithmus zu lokalisieren und die Richtigkeit des Ergebnisses durch die Visualisierung der Ausgabe zu belegen.

Durch die Veränderung der DBSCAN-Parameter konnten die Kriterien zur Clustersuche je nach Wunsch verschärft oder gelockert werden. Ein höherer Wert des *MinPts*-Parameters führte zu kleinerer Anzahl gefundener Cluster bzw. zu Rauschen, während ein niedriger Wert für eine hohe Anzahl an erzeugten Clustern sorgte. Die Änderung des ε-Parameter wirkte sich analog auf die Clusterbestimmung aus.

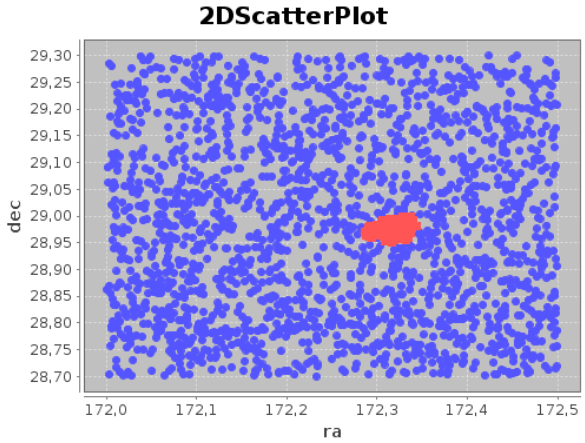


Abbildung 7.10: Ergebnis eines DBSCAN-Clusterings als 2D-Plot

Auch bei der automatischen Bestimmung der beiden Parameter durch den Einsatz eines dritten, nämlich des *noiseRatio*-Parameters, der das Verhältnis zwischen dem *Soll-Rauschanteil* und dem gesamten Rauschanteil angibt, wurde erwartetes Verhalten des DBSCAN-Algorithmus beobachtet.

Durch die erfolgreiche Implementierung dieses Anwendungsfalls konnte ein erster Schritt in Richtung der Entdeckung noch unbekannter Sternencluster gemacht werden. Dabei liegt der besondere Augenmerk auf der Kenntnis der jeweils untersuchten Astronomiedaten und der dazu geeigneten Suchparameter.

Eine detaillierte Anleitung des oben erwähnten Anwendungsbeispiels kann dem Anhang des zu Brick gehörigen Handbuchs [43] entnommen werden.

7.3.6 Quasardetektion

Bei der Quasardetektion werden astronomische Objekte anhand ihrer Spektraldaten in Quasare und andere Objekte aufgeteilt. Diese Fragestellung ist ein offenes Forschungsgebiet und verfügt über ein hohes Verbesserungspotenzial. So haben die parallel an dieser Fragestellung arbeitenden Astronomen der Ruhr-Universität Bochum konnten in Zusammenarbeit mit einem der Betreuer dieser Projektgruppe durch die alleinige Überführung der Daten in einen Data-Mining-Algorithmus eine Genauigkeit von über 70% erreichen können. Durch die Verwendung von berechneten Merkmalen (z.B. positive Fläche der Peaks) wurden sogar 90% erreicht [33].

Aufgabenanalyse

Als diese Aufgabe an die Projektgruppe übertragen wurde, war die Herangehensweise weitgehend unklar. Durch Gespräche mit den die Projektgruppe unterstützenden Mitgliedern des Lehrstuhls für Astronomie der Ruhr-Universität Bochum konnten allerdings mehr oder minder feste Charakteristika für die Unterscheidung von Quasaren und anderen Objekten herausgefunden werden:

Quasare unterscheiden sich von anderen Objekten durch drei eher breite Peaks an vorgegebenen Positionen im Spektrum. Je nach Entfernung des Quasars ergibt sich eine schwache oder starke Rotverschiebung, sodass einzelne Peaks auch außerhalb des Messbereiches liegen

können und das zunehmende Rauschen die Daten stark verfälscht. Zudem besitzen Quasare einen abfallenden Fluss, der sich jedoch mit zunehmender Rotverschiebung zu einer gerade oder gar ansteigend verlaufenden Kurve ändern kann.

Ergebnisse der Verfahrensrecherche

Der erste Schritt zur Bewältigung der Aufgabe bestand darin, mögliche Verfahren und Herangehensweisen zur Peakdetektion und Flussbestimmung zu recherchieren. Diese sind im Folgenden aufgeführt.

Douglas Peucker

Der *Douglas-Peucker-Algorithmus* wurde von den Projektgruppenbetreuern vorgeschlagen und wird zur Kurvenglättung in Graphen, insbesondere in digitalen Karten und Routenplanern, eingesetzt. Der Algorithmus ist rekursiv aufgebaut und beginnt mit der gesamten Punktemenge. Er verbindet Anfangs- und Endpunkt und bestimmt danach den Punkt P, der den maximalen Abstand zur berechneten Geraden aufweist. Dann zerlegt er die Punkte anhand von P in zwei Mengen und fährt auf diesen rekursiv fort. Die Rekursion stoppt, wenn der Abstand von P zur Gerade unter einen angegebenen Parameter fällt.

Der Douglas Peucker Algorithmus wurde in Bezug auf die Reduktion der Spektralpunktmenge und zur Rauschunterdrückung untersucht. Er hat allerdings im Bezug auf die Peakanalyse den Nachteil, dass er Peaks eher verbreitert.

Kürzeste Wege

Dieser Algorithmus wurde von Herrn Prof. Dr. Heinrich Müller vom Lehrstuhl VII der Fakultät Information an der TU Dortmund zur Kurvenglättung vorgeschlagen. Er sortiert (falls die Daten nicht sortiert vorliegen) die Datenpunkte bezüglich einer Koordinate. Nun fügt er von jedem Datenpunkt aus ungerichtete Kanten zu den Datenpunkten in seiner Nachbarschaft hinzu. Die Nachbarschaft beschreibt dabei die Anzahl Punkte, die bezüglich der Sortierung am nächsten liegen. Auf dem so entstandenen Graph berechnet der Algorithmus nun einen kürzesten Weg vom ersten bis zum letzten Datenpunkt bezüglich der Sortierung.

Die Vorteile des Kürzeste-Wege-Algorithmus sind die einfache und effiziente Implementierung und die daraus resultierende gute Laufzeit. Allerdings besitzt er den Nachteil, dass bei Fluktuationen allgemein und insbesondere beim Rauschen, die resultierende Kurve an den oberen oder unteren Peaks so verlaufen kann, dass der kürzeste Weg ausschließlich von Peak zu Peak verläuft. Somit würde die resultierende Kurve nicht den Durchschnitt, sondern das Minimum oder Maximum widerspiegeln. Der Grund, warum es beim Rauschen verstärkt auftritt oder hier der Fehler besonders hoch ist, liegt daran, dass beim Rauschen die Fluktuationsspitzen eher nahe beieinander liegen, also innerhalb einer empfehlenswerten Größe des Nachbarschaftsparameters. Dadurch werden die Spitzen von normalen Charakteristika (Peaks und Absorptionen) eher nicht direkt verbunden. Ein größer gewählter Nachbarschaftsparameter, der auch Peaks und Absorptionen miteinander verbindet, würde in jedem Fall sämtliche Charakteristika des Spektrums abschneiden.

Beispiel: Die Werte eines zu untersuchenden Spektrums seien abwechselnd hoch und niedrig, sodass, wenn man die Punkte verbindet, eine Zick-Zack-Kurve entsteht. Der erste Punkt dieses Zick-Zack-Verlaufs sei ein hoher Punkt. Sei weiter der Nachbarschaftsparameter

auf zwei gesetzt. So würde der Algorithmus im Preprocessing die Punkte zu dem Zick-Zack und zusätzlich die oberen bzw. unteren Ausschläge miteinander verbinden. Der kürzeste Weg verläuft nun entlang der direkt verbundenen oberen Ausschläge, da die Kurve oben beginnt. Somit ist der errechnete Durchschnitt nicht medial im Zick-Zack, sondern verläuft ausschließlich entlang der oberen Ausschläge.

Im Bezug auf die Peakanalyse besitzt der Algorithmus zudem den Nachteil, dass er die Peaks in der Regel abschneidet. In dem vorherigen Beispiel schneidet er die negativen Peaks (Absorptionen) ab. Dieser Nachteil kann sich allerdings zum Vorteil umkehren, wenn sehr schmale Peaks herausgefiltert werden sollen.

Least Squares

Least Squares (auch *Least Squares Fitting* genannt), ebenfalls von Prof. Dr. Heinrich Müller vorgeschlagen, ist das mathematische Standardverfahren zur Ausgleichsrechnung von Kurven. Dabei wird zunächst eine Funktionenklasse (z.B. Gaußkurve) ausgewählt, wobei eine einzelne Funktion daraus durch Festlegung aller durch die Klasse bestimmten Parameter, den Koeffizienten der Funktion, identifiziert wird. Der Algorithmus berechnet diese Parameter so, dass die Summe der quadratischen Abweichungen der Kurve zu den Ursprungspunkten minimiert wird.

Dieses Verfahren ist relativ aufwendig und hat zusätzlich den Nachteil, dass die Parameter in bestimmten Situationen nicht in linearer Abhängigkeit zur Eingabe stehen, da ein lineares Gleichungssystem zur Bestimmung der Koeffizienten erstellt wird, der Algorithmus zur Lösung dieses Gleichungssystems jedoch im *worst case* eine exponentielle Laufzeit besitzt. Zwar ließe sich durch eine geeignete Funktionsklassenauswahl (z.B. Gaußglocken) und einer geschickten Positionierung dieser Funktionen unterhalb der gesuchten Peaks die Laufzeit und der Erfolg verbessern, der Algorithmus bleibt aber weiterhin vergleichsweise aufwändig. Zudem ist bekannt, dass das SDSS ebenfalls dieses Verfahren einsetzt. Da die Projektgruppe auf bessere Ergebnisse hofft, sollte eine anderen Herangehensweise gewählt werden.

Sliding Window

Bei *Sliding Window* handelt es sich um ein allgemeines Verfahren, das nicht auf Kurven oder einzelne Punkte beschränkt ist. Hierbei wird ein Fenster (Teilbereich der zu untersuchenden Menge von Objekten) über die Daten geschoben, und jeweils lokal ein oder mehrere Funktionswerte berechnet. Da dies ein allgemeines Verfahren ist, existieren viele verschiedene problemspezifische Anpassungsmöglichkeiten.

Dieser Algorithmus wird von den Bochumer Astronomen zur Bestimmung von Peaks verwendet, eignet sich allerdings ebenfalls für die Bestimmung des Flusses und zur Kurvenreduktion.

Epsilon Clipping

Das *Epsilon Clipping* wurde von den Astronomen der Ruhr-Universität Bochum im Zusammenhang mit dem Sliding-Window-Algorithmus angesprochen, und lässt sich als Zusatz zu einem beliebigen anderen Verfahren implementieren. Dabei wird innerhalb eines Fensters zunächst der Durchschnitt (bzgl. einer Dimension) der Datenpunkte bestimmt. Anschließend werden die Datenpunkte bezüglich ihres Abstandes zum Durchschnitt absteigend sortiert. Nun werden die ersten ε Datenpunkte[2] verworfen und der Durchschnitt anhand der verblie-

[2] Bei ε handelt es sich um einen Parameter. Er beschreibt die Anzahl der zu verwerfenden Datenpunkte.

benen Datenpunkte neu berechnet.

Mit diesem Verfahren wird verhindert, dass bei einzelnen Peaks die idealisierte Flusskurve an diesen Stellen zu stark in Richtung der Peaks verschoben wird. Somit werden die Ausprägungen von Charakteristika (z.B. Peaks) bei der Subtraktion von der Originalflusskurve und der berechneten idealisierten Flusskurve besser erhalten.

Splines und Wavelets

Splines n-ten Grades sind Funktionen, die stückweise aus Polynomen mit maximalem Grad n zusammengesetzt sind. Diese Funktionen beschreiben eine interpolierte Kurve, die durch die Datenpunkte (Stützstellen) vom Start- bis zum Endpunkt verläuft. Dieses Verfahren ähnelt sehr den im weiteren Verlauf genannten Bézier-Kurven. Eine ausführliche Abhandlung der einzelnen Funktionen und der speziellen *B-Splines* findet sich z.B. in den Vorlesungsfolien zur Veranstaltung *Mensch-aschine-Interaktion* von Herrn Prof. Dr. Müller [38].

Mit dem Begriff *Wavelet* werden die einer kontinuierlichen oder diskreten Wavelet-Transformation zugrunde liegenden Funktionen bezeichnet. Eine genaue Abhandlung geht über den Rahmen dieser Ausführungen hinaus und benötigt zudem spezielle Vorkenntnisse aus dem Bereich der Fourier-Transformation. Daher wird hier auf folgende Literatur verwiesen: [37], [46]

Die beiden Verfahren wurden einerseits von Prof. Dr. Heinrich Müller und andererseits von den Bochumer Astronomen vorgeschlagen, die diese zur Flussbestimmung und Peakerkennung verwenden. Zudem verwendet das SDSS die Wavelets zur Datenanalyse von Spektraldaten. Allerdings wurden diese Verfahren nicht weiter untersucht, da man hoffte, mit anderen Verfahren bessere Ergebnisse und eine effiziente Pipeline zu erreichen.

Sweep Lines

Sweep Lines ist ein in der Informatik allgemeines Schema bei der Konstruktion von Algorithmen. Im zweidimensionalen Fall wird der Raum mittels einer Parallelen zu einer der Achsen (der sogenannten *Sweep Line*) durchlaufen und das Ergebnis sukzessive anhand der bisher betrachteten Objekte nach konstruiert. Dieses Verfahren wird beispielsweise häufig bei der Berechnung von Flächenüberdeckungen verwendet. Dabei wird die Sweep Line von links nach rechts über eine der Achsen verschoben und die aktuelle Flächenüberdeckung anhand der die Sweep Line kreuzenden Flächen ermittelt. Die explizite Implementierung von Sweep-Line-Algorithmen variiert je nach Anwendungsfall und nach Anzahl der Dimensionen des zu betrachtenden Raumes. So wird im n-dimensionalen Fall eine $(n-1)$-dimensionale Hyperebene als Sweep Line verwendet.

Dieses Verfahren wurde im Hinblick auf die Peakerkennung untersucht, jedoch würde sich eine Implementierung nur minimal zu der eines Sliding Window unterscheiden. Das liegt daran, dass sowohl der Peak-Beginn als auch das Peak-Ende gespeichert werden müssen, und somit die zu speichernde Historie dem aktuellen Fensterbereich eines Sliding Window sehr nahe kommt. Eine alternative Implementierung sah vor, die Sweap Line nicht entlang der x-Achse (Wellenlängen), sondern der y-Achse (Intensitäten) zu verschieben. Diese wurde allerdings auf Grund von Implementierungsfortschritten bei anderen Verfahren nicht weiter verfolgt.

Bézier-Kurven

Die *Bézier-Kurven* werden vor allem in der Vektorgrafikverarbeitung angewendet. Dabei wird eine Kurve anhand von Parametern und Kontrollpunkten interpoliert.

Mit einer geeigneten Wahl der Kontrollpunkte lässt sich der Fluss eines Spektrums relativ gut und platzeffizient bestimmen. Auch der Aufwand der Interpolation der Kurve zur Spektrensubtraktion ist vergleichbar mit anderen Verfahren. Das Problem liegt in der Wahl der Kontrollpunkte. Diese müssten nicht nur vorab berechnet werden, oftmals reichen diese schon aus, um den Fluss ausreichend exakt darzustellen.

Herangehensweisen und verwendete Verfahren

Die Astronomen der Universität Bochum haben sich vorwiegend auf die Flussbestimmung und die Peakdetektion konzentriert und die zusätzlich gewonnenen Informationen einem Data-Mining-Algorithmus als zusätzliche Eingaben zur Verfügung gestellt. Dabei verwendeten sie zur Flussbestimmung Splines und zur Peakdetektion ein Sliding Windows Verfahren.

Neben Prof. Dr. Heinrich Müller und den Bochumer Astronomen als externe fachübergreifende Informationsquellen wurde ebenfalls ein Gespräch mit Prof. Dr. Dr. Wolfgang Rhode vom Lehrstuhl E5b der Fakultät Physik der TU Dortmund gesucht. Dieser betrachtet die Spektraldaten im Vergleich zu den Astronomen weniger visuell sondern eher mathematisch. So wurde eine Klassifizierungspipeline entwickelt, die eine völlig andere Herangehensweise verfolgt. Dabei werden in einem Preprocessing die Spektraldaten der einzelnen Objekte analysiert und anhand von Ausschlusskriterien die Menge der möglichen Kandidaten reduziert.

Das Preprocessing besteht im ersten Schritt darin, den Fluss des Spektrums zu bestimmen und Objekte mit stark aufsteigenden Flüssen zu verwerfen. Anschließend werden die Peaks lokalisiert und mit Hilfe des *Full-Width-Half-Height*-Kriteriums (Bestimmung der Breite bei halber Höhe) analysiert. Objekte mit schmalen Peaks werden hier ebenfalls verworfen. Im dritten Schritt erfolgt eine Bestimmung der Peakabstände und -positionen unter Berücksichtigung der Rotverschiebung. Auch hier werden Objekte, deren Peaks nicht an den für Quasare üblichen Positionen sind, verworfen. Anschließend werden die übrigen Kandidaten, ggf. mit einer Datenaufbereitung, als Eingabe für einen Data-Mining-Algorithmus verwendet.

Die Projektgruppe hat entschieden, sich an der Herangehensweise der Bochumer Astronomen zu orientieren, da diese in Bezug auf die Eindeutigkeit der Ausschlusskriterien für das von Herrn Rhode vorgeschlagene Preprocessing Bedenken äußerten. Es sei nicht so einfach möglich, die Rotverschiebung dahingehend zu berechnen, dass die Positionen der für Quasare üblichen Peaks bestimmt und in den Spektraldaten erkannt werden können. Auch gibt es zu viele Ausnahmen und Sonderfälle, die nicht berücksichtigt worden seien.

Die Herangehensweise der Projektgruppe unterscheidet sich aber dennoch insofern von jener der Astronomen aus Bochum, als das Originalspektrum nicht als Eingabe in den Data-Mining-Algorithmus überführt wird. Es werden lediglich die extrahierten Merkmale verwendet, da die zur Verfügung stehenden Algorithmen über keine sinnvolle Abstandsmessung auf Spektren verfügen.

Die Pipeline zur Quasardetektion

Die von der Projektgruppe entwickelte Pipeline zur Quasardetektion setzt sich aus drei – bzw. wenn man den Data-Mining-Algorithmus mitzählt vier – Schritten zusammen.

1. **Flussbestimmung:** Zur Flussbestimmung wird ein Sliding Window Verfahren verwendet, das zusätzlich das Epsilon Clipping unterstützt. Der Algorithmus iteriert von Eingabeanfang bis zum -ende und schiebt das Fenster ggf. über den Wertebereich hinaus. Dadurch

wird erreicht, dass die Eingabelänge in jedem Fall erhalten bleibt. Zudem ermöglicht ein Parameter eine Überlappung der einzelnen Fensteriterationen.

Beim Sichten der Spektraldaten des SDSS-Kataloges fiel auf, dass die Datenpunkte an den Randbereichen starken Fluktuationen unterliegen. Es ist nicht ausgeschlossen, dass es sich dabei um Messfehler oder andere Störungen handelt, da dieses Phänomen bei fast allen untersuchten Daten auftrat. Diese Datenpunkte lassen sich daher optional über die *margin*-Parameter vor dem Durchlauf des Algorithmus abschneiden.

2. **Subtraktion:** Im zweiten Schritt der Pipeline wird nun der Fluss von den Originaldaten subtrahiert. In einem Preprocessing werden zunächst sämtliche Datenpunkte, die in einem Spektrum vorhanden sind und in dem anderen fehlen, gegenseitig interpoliert. Dadurch verläuft die anschließende Subtraktion komplett ohne Interpolation. Neben der reinen Subtraktion lässt sich das Ergebnis zusätzlich durch den Subtrahenden dividieren, wodurch die Intensitätsverhältnisse möglichst gut herausgearbeitet werden. Ein numerischer Parameter ermöglicht es, Rauschen zu unterdrücken, indem Intensitätsdifferenzen, die kleiner als dieser Wert sind, auf 0 gesetzt werden.

3. **Merkmalsextraktion:** Der letzte Schritt vor der Überführung der Daten in einen Data-Mining-Algorithmus besteht darin, verschiedene Merkmale aus dem Subtraktionsergebnis zu extrahieren. Diese werden als zusätzliche Eingabe für den Algorithmus verwendet. Die Merkmalsextraktion erfolgt, wie die Flussbestimmung auch, mit einem Sliding Window Verfahren. Dabei wird vom Ausgangspunkt des Fensters aus ein „starker“ Ausschlag nach oben mit anschließendem entsprechenden Abfall gesucht. Sind diese beiden Faktoren gefunden, wurde ein Peak in Emission erkannt, der analysiert werden kann. Für Peaks in Absorption verfährt der Algorithmus entsprechend analog.

   Die Merkmalsextraktion berechnet die folgenden Merkmale:

   - Anzahl positiver Peaks
   - Anzahl negativer Peaks
   - Summe der Flächen der positiven Peaks
   - Summe der Flächen der negativen Peaks (Absorptionen)
   - Maximale Breite der Peaks bzw. Absorptionen
   - Maximale Höhe der Peaks bzw. Absorptionen

Ergebnisse der Quasardetektion

Zum Zeitpunkt der Fertigstellung dieses Abschlussberichtes war die Pipeline zur Quasardetektion fertiggestellt und auf randomisierten Daten erfolgreich getestet worden. Verzug bei der Aufbereitung und Integration vorklassifizierter Realdaten erlaubten jedoch keine rechtzeitige Anwendung auf diese und deswegen konnte keine Analyse der Ergebnisse erfolgen.

Da die implementierten Methoden jedoch den in [33] verwendeten entsprechen, ist nicht davon auszugehen, dass die Ergebnisse stark von den dort erzielten abweichen.

8 Fazit und Ausblick

Zum Zeitpunkt der Abgabe dieses Abschlussberichts hat sich jedes Mitglied der PG über ein Jahr lang mit dem Projekt beschäftigt. Das begann im letzten Sommer mit der Vorbereitung der Vorträge für die Seminarphase, entwickelte sich über Organisations- und Konzeptionsarbeit zur zentralen Aufgabe der Implementierung und endete kürzlich in der Arbeit am Abschlussbericht und dem Handbuch zu Brick. Die folgenden Abschnitte ziehen ein Fazit der erreichten Ziele und geben einen Ausblick auf die mögliche weitere Entwicklung. Abschließend dankt die Projektgruppe ihren Unterstützern.

8.1 Fazit

Die Ziele der Projektgruppe ergaben sich direkt aus dem Antrag [54]. Darin werden einige Eigenschaften sowie Verwendungen des gewünschten Systems aufgezählt, die im Folgenden auf ihre Erfüllung hin untersucht werden. Jegliche Abweichung von den Zielen fand dabei natürlich in Absprache mit den Betreuern statt.

Modularität / Schnittstelle für Data-Mining-Algorithmen

Dieses Ziel bestimmte die Entwicklung von Brick von Anfang an und fand in der freien Erweiterbarkeit durch die Kapselung von Datenschnittstellen und Algorithmen in Plugins (siehe Abschnitt 4.3.2, Seite 37) seinen Ausdruck. Die Verwendung der Netbeans RCP auf GUI-Seite (siehe Abschnitt 5.2.3, Seite 43) ermöglicht außerdem ein modulares Erweitern der Benutzeroberfläche um Visualisierungen oder sogar Funktionalitäten.

Ferner ist es möglich, ohne Änderungen am Framework einen neuen Client zu entwickeln, wie die kurzfristige Umsetzung eines CLI zeigt.

Implementierung und Anbindung von Data-Mining-Algorithmen

Mit k-Means, kNN, DBSCAN und dem Hierarchischen Clustering wurden vier Data-Mining-Algorithmen integriert; keiner davon allerdings in einer Variante mit optimaler Laufzeit.

Darüber hinaus wurden Plugins realisiert, die zur Verwendung dieser und anderer Algorithmen nötig sind. Dazu gehören Datenschnittstellen (u.a. für SQL-Datenbanken, CSV-Dateien und VOTables), Module zur Datenaufbereitung (u.a. Normalisierer und SDSS-Spektrenkonverter) und -verwaltung (u.a. Spalten- und Zeilenfilter) sowie Visualisierungen (u.a. 2D-, 3D- und 5D-Plots).

Verarbeitung großer Datenmengen

Die Datenschnittstelle wurde – wie der Rest von Brick ebenfalls – in Java geschrieben. Dabei wurden keine Schritte unternommen, eine eigene Speicherverwaltung umzusetzen, d.h. die Effizienz der Datenschnittstelle hängt direkt von jener der JVM ab.

Mit der Unterscheidung nach sequentiellem und wahlfreien Zugriff (siehe Abschnitt 4.3.5, Seite 39) wurde gemeinsam mit dem Threading der Knoten allerdings dafür gesorgt, dass die Möglichkeit besteht, große Datenmengen sukzessive durch eine Anwendung bearbeiten zu lassen. Damit soll der Verkehr zwischen Hauptspeicher und Festplatte reduziert werden.

Ferner ermöglicht die Aufteilung von Brick in einen Client und einen Server, dass im Regelfall leistungsstarke Systeme auf großen Datenbeständen arbeiten.

Bearbeitung forschungsrelevanter Problemstellungen

Mit der Spektren-gestützten Quasardetektion (siehe Abschnitt 7.3.6, Seite 78) wurde in Brick ein Verfahren umgesetzt, welches Mitarbeiter des die Projektgruppe betreuenden Lehrstuhls 11 und des Lehrstuhls für Astronomie der Ruhr-Universität Bochum gemeinsam entwickelt haben [33].

Außerdem wurde DBSCAN erfolgreich verwendet, um die im SDSS-Katalog abgelegten Objekte nach räumlichen Clustern zu durchsuchen. Ob diese real existierenden oder bekannten Sternenhaufen entsprechen war nicht automatisch zu ermitteln, da diese Information im SDSS-Katalog nicht abgelegt sind. Stichprobenartige Kontrollen ergaben allerdings, dass einige der Cluster in der Tat bekannten Sternenhaufen entsprachen.

Visualisierung der SDSS-Bilddaten

Nachdem beschlossen wurde Brick als allgemeine Data-Mining-Suite zu entwerfen, verallgemeinerte sich auch der Visualisierungsansatz, und der SDSS-Katalog war nur noch eine Datenquelle unter vielen möglichen, die man verarbeiten und anzeigen wollte. Deswegen wurde der Fokus auf die Darstellung von Tabellendaten gerichtet, wie sie auch im SDSS-Katalog einen bedeutenden Teil ausmachen. Für diese wurden verschiedene Visualisierungen umgesetzt: eine Tabellenansicht, 2D-, 3D- und 5D-Plots und ein Matrix-Scatter-Plot, bei dem es sich um eine Matrix von Plots handelt, die alle möglichen Kombination von je zwei Dimensionen des Datensatzes in je einem 2D-Plot darstellt.

Die im SDSS-Katalog ebenfalls enthaltenen Bilddaten wurden allerdings schon früh als Grundlage für Data Mining verworfen und damit erschien es nicht sinnvoll, dafür gesonderte Darstellungsmechanismen umzusetzen.

Bedienbarkeit

Obwohl dies im Antrag nicht explizit ausformuliert wurde, war eine einfache Bedienbarkeit stets ein zentraler Anspruch. Mit Astronomen handelt es sich bei der primären Zielgruppe nicht um informatisch geschulte Personen und auch bei der von der PG vorgenommenen Verallgemeinerung der Anwendbarkeit von Brick rückten selbstverständlich keine Informatiker in den Blickpunkt. Stattdessen sollte das System gerade Personen, die Data-Mining-Algorithmen nicht selbst umsetzen können, eine Möglichkeit geben, diese dennoch zu verwenden.

Vor dem Hintergrund lag – insbesondere in der GUI, aber auch beim CLI – das Entwicklungsziel auf einer einfachen Bedienbarkeit. Durch die intuitive grafische Darstellung einer Anwendung sowie die umfangreichen und differenzierten Fehlermeldungen sollte dies gelungen sein.

8.2 Ausblick

Wie geht es mit BRICK weiter? In diesem Abschnitt werden zunächst mögliche neue Features und Plugins beschrieben, bevor ein kurzer Blick auf die Möglichkeiten einer zukünftigen Arbeit am System geworfen wird.

8.2.1 Weiterentwicklung

Obwohl BRICK alle angedachten Konzepte umsetzt und damit schon die zentralen Funktionalitäten einer Data-Mining-Suite anbietet, ist selbstverständlich viel Raum für Erweiterungen. Diese könnten existente Konzepte oder die Bedienbarkeit verbessern, aber natürlich auch neue Aspekte abdecken. Wie auch während des letzten Jahres ist wohl nur die zur Verfügung stehende Zeit die Grenze der Innovation.

Erweiterung um externe Bibliotheken

Eine mögliche Erweiterung von BRICK könnte eine Integration von externen Komponenten wie *Apache Mahout*[2] oder *SVM-light*[18] umfassen. Einhergehend mit einer Adaption von Mahout liegt auch eine Integration von *Apache Hadoop*[1] nahe, so dass Brick über die Fähigkeit verfügen könnte, innerhalb eines Clusters Datenmengen im *Petabyte*-Bereich parallel zu verarbeiten. Eine weitere erwägenswerte Bibliothek, um die Fähigkeiten von BRICK stark zu erweitern, ist *WEKA*[36], welches von der Universität in Waikato, Neuseeland, entwickelt wird.

Autoparameterzuweisung für Module

Die Autoparameterzuweisung ist momentan nur für Knoten implementiert, weswegen bei der Implementierung eines Moduls alle Parameterwerte manuell ausgelesen werden müssen. Eine Ausweitung des Mechanismus auf Module würde eine Vereinfachung der Modulprogrammierung bedeuten.

Update-Funktion der Netbeans-Plattform nutzen

Die Netbeans-Plattform bietet eine Möglichkeit an, Updates über eine zentrale Quelle zu beziehen [13]. Diese könnte bei BRICK nicht nur dazu verwendet werden, Aktualisierungen zu verbreiten, sondern auch eine zentrale Plattform für den Austausch von selbst erstellten Knoten oder Plugins (z.B. für die Visualisierung) bieten.

Verbesserungen der GUI-Benutzbarkeit

Während der Entwicklung und dem Testen der GUI kamen eine Vielzahl von Ideen auf, welche die Benutzbarkeit der Oberfläche verbessern könnten. Leider konnten wegen der knappen Zeit nur sehr wenige davon umgesetzt werden. Die nachfolgende Liste soll aufzeigen, an welchen Stellen die graphische Benutzeroberfläche beispielsweise noch verbessert werden könnte:

- Einführen von Tastaturkürzeln, bspw. zum Speichern oder Entfernen von Knoten/Kanten.
- Unterschiedliche Farbschemen für fehlerhafte Knoten/Kanten im Application Builder.

- Bessere Interaktion zwischen der Fehlerausgabe und dem graphischen Editor (z.B. Springen zu fehlerhaften Knoten mittels eines Mausklicks in der Fehlerliste).
- Vermeidung der Anzeige von Passwörtern im Klartext (wie z.B. in der Parameterliste).
- Fehlerhafte Parameter farblich in der Parameterliste hervorheben.
- Verbesserte Möglichkeiten in den Plots Daten zur Weiterverarbeitung auszuwählen.

Automatische Erzeugung und Ausführung von Batch-Jobs

Bei der Untersuchung von Daten mit Hilfe von Data-Mining-Algorithmen tritt häufig der Fall auf, dass die optimalen Parameterwerte der verwendeten Algorithmen nicht oder nur ungefähr bekannt sind. Daher ist es wünschenswert, eine Anwendung auf einer vorgegebenen Datenmenge mehrfach mit verschiedenen Einstellungen laufen zu lassen.

Sofern ein Algorithmus jedoch nicht von sich aus die Möglichkeiten bereit stellt für Parameter einen gewissen Bereich vorzugeben, müssen Batch-Jobs dieser Art in Brick bisher „von Hand“ erzeugt werden. Zwar können diese durch Verwendung von Variablen in (handgeschriebenen) Application-Dateien und dem CLI leicht durch Shellskripte umgesetzt werden, jedoch wäre eine – GUI-gestützte – Automatisierung dieses Vorgangs wünschenswert.

Dazu könnte z.B. beim Einstellen von numerischen Parametern in der GUI ein Batch-Job-Haken gesetzt werden, dessen Aktivierung die Angabe einer Spanne von Werten (Minimum, Maximum und Schrittweite) statt nur eines einzelnen Wertes ermöglicht. Die Anwendung würde dann mehrmals mit den verschiedenen Parametern ausgeführt.

Knotenrahmen per Wizard erstellen

Obwohl die Integration eines Algorithmus in Brick für einen versierten Programmierer kein Hindernis darstellen dürfte, kann es doch eine zeitintensive Aufgabe sein. Dementsprechend wäre es – in Anlehnung an GUI-Wizards, welche Programmcode entsprechend der zusammengesetzten GUI-Elemente erzeugen – eine erhebliche Erleichterung, wenn dem Knotenprogrammierer die Möglichkeit gegeben würde, über ein GUI-Frontend bspw. die vom Knoten benötigten Parameter und Kanäle samt ihrer jeweiligen Typen anzugeben. Des Weiteren wäre es über einen solchen Wizard möglich, den Rahmen des Gerüsts in drei Klassen (was nicht zwingend notwendig ist) aufzuteilen und dem Benutzer hiermit schon im Vorfeld der Entwicklung eine gewisse Arbeitserleichterung zu bieten. Weitere Funktionalitäten eines solchen Wizards könnten unter anderem das Anlegen der Kanäle samt Berichten oder das Auslesen der Eingabe sein.

8.2.2 Weitere Plugins

Allein schon im Bereich der Data-Mining-Algorithmen ist ein unerschöpflich scheinender Vorrat von möglichen Plugins für Brick gegeben. Ebenso sind verschiedenste Visualisierungsmodule denkbar. Einige darüber hinausgehende Ideen zur Integration oder Verwendung weiterer Plugins sind im Folgenden beschrieben.

Erweiterung und Verbesserung von DBSCAN

Wie bereits in Abschnitt 7.3.5 (siehe Seite 73) erwähnt, würde eine zukünftige Verbesserung der Parameterbestimmung durch die Heuristik der sortierten kNN-Distanzen möglicherweise zu einem genaueren Ergebnis der Clusterbestimmung durch DBSCAN und einer einfacheren Handhabung dieses Anwendungsfalls führen. Des Weiteren könnte evtl. die durch entsprechend große Datenmengen verursachte längere Laufzeit durch den Einsatz von Indexstrukturen wie *R*-Bäumen* oder *Gridfiles* wesentlich reduziert werden. Die Implementierung dieser Datenstrukturen war aus zeitlichen Gründen leider nicht innerhalb des Projektrahmens möglich.

Export in andere Formate

Die Ergebnisse einer Application in Formaten externer Programme (z.B. Aladin) abzulegen, würde vielen Anwendern, die bereits seit längerem mit solchen Programmen arbeiten, die Möglichkeit bieten, die mit Brick erarbeiteten Resultate in einem bekannten Arbeitsumfeld zu evaluieren bzw. weiter zu bearbeiten.

Visualisierung nicht-tabellenartiger Daten

Neben den implementierten Darstellungsformen für tabellenartige Daten (z.B. 2D/3D-Scatter-Plot, Matrix-Scatter-Plot) ist es bei manchen Anwendungsfällen angebracht, die Ergebnisse auf andere Art darzustellen. So könnten z.B. die Ergebnisse des hierarchischen Clusterings als Dendrogramm oder die Ausgabe des Algorithmus C4.5 als Entscheidungsbaum visualisiert werden, um dem Anwender die Möglichkeit zu geben, anhand dieser das Ergebnis genauer zu beurteilen.

Ausbau der Arbeit mit Spektren

Die Arbeit mit Spektraldaten wurde ausschließlich auf den Bereich der Quasardetektion begrenzt. Außerdem konnten aufgrund von weiteren Aufgaben viele Ideen zur Optimierung der Quasardetektion nicht entwickelt werden. Daher birgt dieser Bereich ein hohes Potenzial für weitere Aufgabenstellungen und Optimierungen, die im Folgenden aufgeführt werden.

Metriken und Spektraldaten

Die Projektgruppe hat im Laufe der Arbeiten mit den Spektraldaten entschieden, diese in zwei Arrays – je eines für die Wellenlängen und die Intensitäten – zu speichern. Allerdings hatte die Entscheidung zur Folge, dass diese Daten von den bisher implementierten Metriken nicht mehr unterstützt wurden. Die bisherigen Metriken benötigen ein Datum je Zelle, um den Abstand zweier als Datenpunkte interpretierter Tabellenzeilen zu bestimmen. Die Spektren bilden dahingehend eine Ausnahme, dass sie in zwei Zellen je ein Array zur Verfügung stellen.

Zum einen müssten die Metriken nun dahingehend erweitert werden, dass auch solche Datenformate (Arrays) gelesen werden können, und zum anderen, dass die Verbindungen bzw. Zuordnungen von einzelnen Wellenlängen und Intensitäten erhalten bleiben.

Algorithmen zur Vereinfachung von Spektraldaten

Die letztendlich implementierte Pipeline zur Quasardetektion umfasst nur einen Bruchteil der recherchierten Verfahren. Zwar wurden einige Verfahren im Hinblick auf die Tauglichkeit direkt verworfen, andere hingegen öffneten den Weg zu einigen Gedankenexperimenten.

Grundsätzlich ist es lohnenswert, diese Verfahren zu implementieren und deren Verhalten auf Spektraldaten zu analysieren. Im Speziellen wäre es interessant zu sehen, ob die Annahmen zur Peakdetektion – beispielsweise dass der *Douglas-Peucker-Algorithmus* die Peaks eher verbreitert – tatsächlich zutreffen oder aber widerlegt werden können.

Schablonensuche in Spektraldaten

Während der ersten Arbeitsschritte zur Quasardetektion wurden auch Vereinheitlichungen von Merkmalssuchen in Spektraldaten diskutiert. Prof. Dr. Vahrenhold motivierte ein Schablonen-Such-Verfahren, das auf Grund fehlender Kapazitäten nicht weiter verfolgt werden konnte. Bei diesem Verfahren stehen dem Benutzer vorgegebene Schablonen (siehe Abbildung 8.1) zur Verfügung. Diese können frei kombiniert werden und sollen einen gesuchten Spektralfluss symbolisieren. Anschließend soll ein entsprechender Algorithmus die Objekte finden, die einen ähnlichen Spektralfluss besitzen.

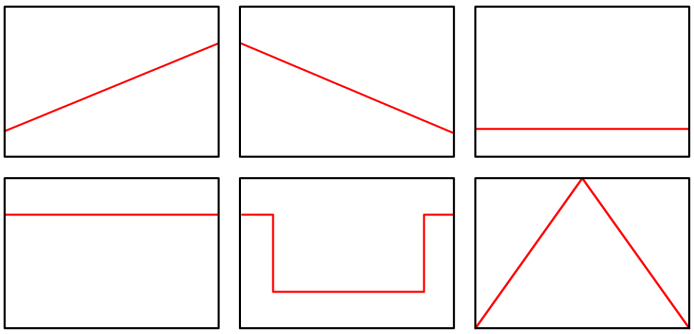


Abbildung 8.1: Mögliche Schablonen, aus denen ein Spektralfluss kombiniert werden kann.

8.2.3 Weiterführung des Projekts

Mitarbeiter des Lehrstuhls für Astronomie an der Ruhr-Universität Bochum werden in den nächsten Wochen beginnen, Brick testweise zu verwenden. Allerdings ist der Funktionsumfang mit nur vier Data-Mining-Algorithmen wohl noch zu gering, um eine umfangreiche Unterstützung der Arbeitsprozesse zu ermöglichen. Gleichzeitig muss beachtet werden, dass keiner der Algorithmen in einer Laufzeit-optimalen Version implementiert ist. Um Brick einen erfolgreichen Start zu ermöglichen, wäre es also wichtig, Ansprechpartner anzubieten, die Erweiterungen auf Funktionalitäts- und besonders Plugin-Ebene durchführen können.

Momentan müssen allerdings Zweifel bestehen, ob es dazu kommt. Bisher hat keiner der PG-Teilnehmer Interesse geäußert, sich z.B. im Rahmen einer Diplomarbeit weiter mit Brick zu befassen. Außerdem befinden sich die meisten Teilnehmer noch mindestens ein Jahr im Studium und somit ist auch die Gründung eines Unternehmens, welches die Arbeit am System in einem kommerziellen Rahmen fortsetzt, in der nahen Zukunft eher unwahrscheinlich.

8.3 Danksagung

Zwei Semester sind vergangen und als Resultat kann die Projektgruppe stolz Brick präsentieren. Und auch wenn sie dessen Entwicklung und Implementierung eigenständig bewältigt hat, wäre dies ohne das fruchtbare Umfeld, das ihr zur Verfügung stand, wenn überhaupt nur

sehr viel schwerer möglich gewesen. Dass dieses existierte ist einigen „Helfern“ zu verdanken, die es mit Erfahrung, Wissen, Inspiration und Motivation geschaffen haben.

Die Projektgruppe möchte insbesondere den Projektgruppenbetreuern des Lehrstuhls XI der Fakultät für Informatik der TU Dortmund, Prof. Dr. Jan Vahrenhold, Fabian Gieseke, Dr. Oliver Kramer und Andreas Thom, für die sehr gute persönliche Betreuung, die umfangreichen Freiheiten und die andauernde Motivation für Ideen und Exkurse, die fachliche Unterstützung und die technischen Rahmenbedingungen danken. Weiter bedankt sich die Projektgruppe bei den Kooperationspartnern des Lehrstuhls für Astronomie der Ruhr-Universität Bochum, Dr. Dominik Bomans und Kai Polsterer, für die gute Zusammenarbeit und ausführliche Hilfe bei sämtlichen astronomischen Fragestellungen.

Bei der Ausarbeitung und Umsetzung der verschiedenen Anwendungsfälle – insbesondere der Quasardetektion – wurde versucht, fachübergreifende Informationen für neue Wege zur Lösung der Aufgaben zu finden. Die Projektgruppe dankt daher Prof. Dr. Heinrich Müller vom Lehrstuhl VII der Fakultät für Informatik und Prof. Dr. Dr. Wolfgang Rhode vom Lehrstuhl Experimentelle Physik V der Fakultät Physik der TU Dortmund für ihre Zeit, ihr Wissen, ihre Ideen und ihre Hilfsbereitschaft.

Abschließend möchte sich die Projektgruppe bei allen Kommilitonen und Freunden bedanken, die an verschiedenen Stellen ihren Beitrag zum Erfolg dieser Lehrveranstaltung geleistet haben.

Literaturverzeichnis

[1] *Apache hadoop - reliable, scalable, distributed computing.* URL: http://hadoop.apache.org/ [Stand 5.08.2010].

[2] *Apache mahout - A Scalable machine learning library.* URL: http://mahout.apache.org/ [Stand 5.08.2010].

[3] *Diplomprüfungsordnung 2001 der Fakultät für Informatik (TU Dortmund).*

[4] *Eclipse.* URL: http://www.eclipse.org/ [Stand 31.03.2010].

[5] *Google Earth.* URL: http://earth.google.de/ [Stand 28.03.2010].

[6] *International Virtual Observatory Alliance.* URL: http://www.ivoa.net/ [Stand 31.03.2010].

[7] *IRIS-Software.* URL: http://www.astrosurf.com/buil/us/iris/iris.htm [Stand 31.03.2010].

[8] *Java Easy 3d – jzy3d.* URL: http://code.google.com/p/jzy3d/ [Stand 25.03.2010].

[9] *Java™ Binding for the OpenGL® API Wiki.* URL: http://kenai.com/projects/jogl/pages/Home [Stand 25.03.2010].

[10] *MagicDraw UML.* URL: http://www.magicdraw.com/ [Stand 30.03.2010].

[11] *Mirage.* URL: http://cm.bell-labs.com/who/tkh/mirage/ [Stand 28.03.2010].

[12] *Netbeans IDE – Unified Modelling Language Plugin.* URL: http://netbeans.org/features/uml/ [Stand 30.03.2010].

[13] *Netbeans Platform Project.* URL: http://platform.netbeans.org/ [Stand 25.03.2010].

[14] *NumPy.* URL: http://www.numpy.org/ [Stand 31.03.2010].

[15] *SciPy – Scientific Tools for Python.* URL: http://www.scipy.org/ [Stand 31.03.2010].

[16] *Sky Surveys Tutorial des SDSS.* URL: http://cas.sdss.org/astrodr7/en/proj/advanced/skysurveys/ [Stand 28.03.2010].

[17] *Sloan Digital Sky Survey.* URL: http://www.sdss.org [Stand 28.03.2010].

[18] *svm light: Support Vector Machine implementation in C.* URL: http://svmlight.joachims.org/ [Stand 13.08.2010].

[19] *The R Project for Statistical Computing.* URL: http://www.r-project.org/ [Stand 31.03.2010].

[20] *The Trac Project*. URL: http://trac.edgewall.org/ [Stand 28.03.2010].

[21] *TOPCAT*. URL: http://www.star.bris.ac.uk/~mbt/topcat/ [Stand 28.03.2010].

[22] Achtert, E.: *Inkrementelles hierarchisches Clustering*. Diplomarbeit, Ludwig Maximilians Universität München, 2004.

[23] Assfalg, J., C. Böhm, K. Borgwardt, M. Ester, E. Januzaj, K. Kailing, P. Kröger, J. Sander und M. Schubert: *Skript zur Vorlesung Knowledge Discovery in Databases - Wintersemester 2009/2010*. URL: http://www.dbs.ifi.lmu.de/Lehre/KDD/WS0910/skript/kdd-5-clustering.pdf [Stand 13.07.2010].

[24] Ball, N. L. und R. J. Brunner: *DATA MINING AND MACHINE LEARNING IN ASTRONOMY*. URL: http://arxiv.org/PS_cache/arxiv/pdf/0906/0906.2173v1.pdf [Stand 31.03.2010].

[25] Begau, Christoph, Christoph Heuel, Raffael Joliet et al.: *Endbericht PG 500 - Intuitive Visualisierung und interaktive Auswertung von Paretomengen - PAVEL*, 2007.

[26] Block, Marco: *Java Intensivkurs*. Springer-Verlag, 2010.

[27] Bonnarel, F., P. Fernique, O. Bienaymé, D. Egret, F. Genova, M. Louys, F. Ochsenbein, M. Wenger und J. G. Bartlett: *The ALADIN interactive sky atlas. A reference tool for identification of astronomical sources*. Astronomy and Astrophysics Supp. Ser., 143:33–40, April 2000.

[28] Chaudhary, Amitabh, Alexander S. Szalay, Er S. Szalay und Andrew W. Moore: *Very Fast Outlier Detection in Large Multidimensional Data Sets*. In: *DMKD'02*. ACM Press, 2002.

[29] Ester, M., H. Kriegel, J. Sander und X. Xu: *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*. In: *2nd International Conference on Knowledge Discovery and Data Mining*, München, 1996.

[30] Ester, M. und J. Sander: *Knowledge Discovery in Databases*. Springer, 2000.

[31] Frakes, W. B. und R. Baeza-Yates: *Information Retrieval*. Prentice Hall, 1992.

[32] Franz, M. O.: *Lineare Klassifikatoren*. Fachhochschule Konstanz, 2007.

[33] Gieseke, Fabian, Kai Lars Polsterer, Andreas Thom, Peter Zinn, Dominik Bomans, Ralf-Jürgen Dettmar, Oliver Kramer und Jan Vahrenhold: *Identifying Quasars in Large-Scale Spectroscopic Surveys*, 2010.

[34] International Virtual Observatory Alliance: *VOTable Format Definition*. URL: http://www.ivoa.net/Documents/VOTable/ [Stand 31.03.2010].

[35] Jain, A. K. und R. C. Dubes: *Algorithms for Clustering Data*. Prentice Hall, 1988.

[36] Mark Hall, Eibe Frank Geoffrey Holmes Bernhard Pfahringer Peter Reutemann Ian H. Witten: *The WEKA Data Mining Software: An Update*. In: *SIGKDD Explorations*.

[37] Müller, H.: *Vorlesungsfolien Digitale Bildverarbeitung.* URL: http://ls7-www.cs.uni-dortmund.de/cms/de/node/621.

[38] Müller, H.: *Vorlesungsfolien Mensch-Maschine-Interaktion.* URL: http://ls7-www.cs.uni-dortmund.de/cms/de/node/594.

[39] Mutzel, P.: *Vorlesungsfolien Algorithmen und Datenstrukturen.* URL: http://ls11-www.cs.uni-dortmund.de/people/gutweng/AD08/ad.jsp.

[40] Ochsenbein, F., P. Bauer und J. Marcout: *The VizieR database of astronomical catalogues.* Astronomy and Astrophysics Supp. Ser., 143:23–32, April 2000.

[41] Oracle: *Java Native Interface – Specification.* URL: http://java.sun.com/javase/6/docs/technotes/guides/jni/spec/jniTOC.html [Stand 31.03.2010].

[42] Padovani, P.: *The Virtual Observatory: an Overview.* URL: http://www.euro-vo.org/pub/general/presentations/VO_Overview_Padovani.ppt[Stand 31.03.2010].

[43] PG543: *Brick, Ein Data-Mining-Framework für Astronomie, Handbuch.* 2010.

[44] Quinlan, J. Ross: *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[45] Sibson, R.: *An optimally efficient algorithm for the singlelink cluster method.* The Computer Journal, 16(1):30–34, 1973.

[46] Sloan Digital Sky Survey / SkyServer: *Algorithms: Spectroscopic Redshift and Type Determination.* URL: http://www.sdss.org/dr5/algorithms/redshift_type.html [Stand 27.06.2005].

[47] Sloan Digital Sky Survey / SkyServer: *Explore.* URL: http://cas.sdss.org/dr7/de/tools/explore/obj.asp [Stand 31.03.2010].

[48] Sloan Digital Sky Survey / SkyServer: *Finding Chart.* URL: http://cas.sdss.org/dr7/de/tools/chart/chart.asp [Stand 31.03.2010].

[49] Sloan Digital Sky Survey / SkyServer: *Image List.* URL: http://cas.sdss.org/dr7/de/tools/chart/list.asp [Stand 31.03.2010].

[50] Sloan Digital Sky Survey / SkyServer: *Navigate.* URL: http://cas.sdss.org/dr7/de/tools/chart/navi.asp [Stand 31.03.2010].

[51] Sloan Digital Sky Survey / SkyServer: *Radialsuche.* URL: http://cas.sdss.org/dr7/de/tools/search/radial.asp [Stand 31.03.2010].

[52] Sloan Digital Sky Survey / SkyServer: *Science Projects.* URL: http://cas.sdss.org/dr7/en/proj/ [Stand 31.03.2010].

[53] Sloan Digital Sky Survey / SkyServer: *SDSS SQL-Tool.* URL: http://cas.sdss.org/dr7/de/tools/search/sql.asp [Stand 31.03.2010].

[54] Vahrenhold, J. und F. Gieseke: *Projektgruppenantrag: Galaxy Quest: Planvoll durchs Weltall.* URL: http://ls11-www.cs.uni-dortmund.de/people/gieseke/galaxy_quest/antrag.pdf [Stand 28.03.2010].

[55] W3C: *SOAP Specifications*. URL: http://www.w3.org/TR/soap/ [Stand 31.03.2010].

[56] Weiss, S. M. und C. A. Kulikowski: *Computer Systems that Learn*. San Mateo, CA, 1991.

[57] Wells, D. C., E. W. Greisen und R. H. Harten: *FITS – a Flexible Image Transport System*. Astronomy and Astrophysics Supp. Ser., 44:363–+, Juni 1981.

[58] Williams, R. und G. Djorgovski: *BRAVO 2007: Lectures on Virtual Observatory*. URL: http://www.astro.caltech.edu/~george/bravo/ [Stand 30.03.2010].

[59] Wu, Xindong und Vipin Kumar: *Top ten algorithms in data mining*. Chapman & Hall/CRC, 2009.

[60] Zwicker, Matthias: *Datenstrukturen & Algorithmen*. Universität Bern, 2010.