MASTERARBEIT

Integration von Data-Mining-Methoden zur Analyse der Daten aus dem astrophysikalischen Experiment LOPES

vorgelegt von

Marcin Franc

im Sommersemester 2010
an der Fachhochschule Brandenburg
am Fachbereich Informatik und Medien

Gutachter: Prof. Dr. Jochen Heinsohn (FHB)
und Prof. Dr. Johannes Blümer (KIT)

18. August 2010

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Arbeit beschäftigt sich mit der Integration von Data-Mining-Methoden, die zur Analyse der Daten aus dem astrophysikalischen Experiment LOPES benutzt werden können. Den Hauptteil der Arbeit bilden die Implementierung und Beschreibung der Software, welche die vorgenannte Integration durchführt. Der erste Teil dieser Beschreibung besteht aus der Analyse der Anforderungen an die Software zusammen mit der notwendigen Theorie der verwendeten Data-Mining-Methoden. Der zweite Teil ist eine Software-Engineering-orientierte Darstellung der implementierten Lösungen. Da die interessierte Leserschaft sowohl aus Informatikern als auch Physikern besteht, beginnt die Arbeit mit der allgemeinen Vorstellung des LOPES-Experiments und den Grundideen des Data Mining.

1.1 Vorstellung des LOPES-Experiments

Das Experiment findet an dem Karlsruher Institut für Technologie (KIT) statt und versucht im Allgemeinen die Beschaffenheit der kosmischen Strahlung zu beschreiben. Diese Strahlung besteht aus Teilchen mit einer Energie von mehr als 10^{10} eV, die mit der Erdatmosphäre kollidieren und dabei einen sog. Luftschauer aus Sekundärteilchen auslösen. Diese Sekundärteilchen wiederum werden im Erdmagnetfeld durch die Lorentzkraft abgelenkt und emittieren dabei einen Radiopuls, der mit Experimenten wie LOPES gemessen wird. Die Radio-Emission von diesen Teilchen wurde zuerst von Jelley im Jahr 1965 entdeckt und in der Publikation [Jel+65] genauer beschrieben. Das LOPES-Experiment steht im Zusammenhang mit dem KASCADE-Grande-Experiment, das für Referenz-, Test- und Kalibrierungszwecke benutzt wird. Das Experiment analysiert den Frequenzbereich von 40 bis 80 MHz. Die

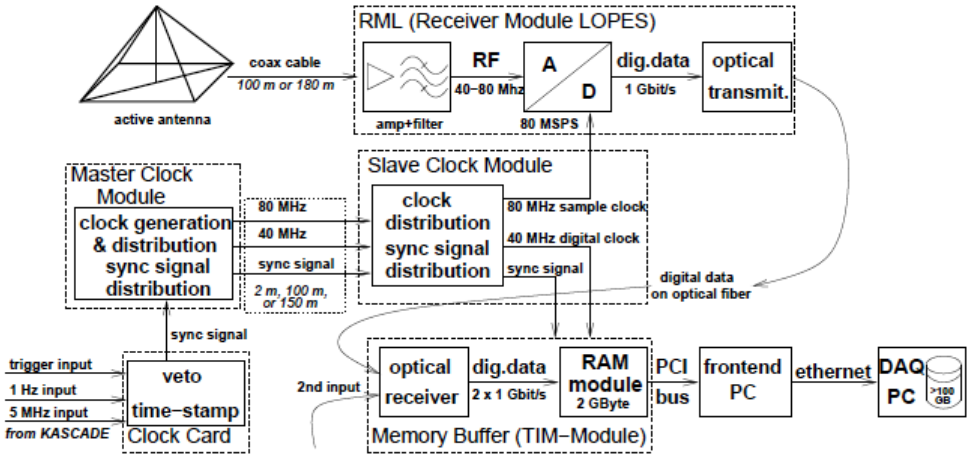

Abbildung 1.1: Überblick der Hardware von LOPES, Quelle: [Hor06].

benutzte Hardware wird mit der Abbildung 1.1 präsentiert. In starker Vereinfachung wird ein Signal mit einer Antenne empfangen, über ein Koaxialkabel an den Empfänger geschickt, digitalisiert und an das Speicher-Modul übertragen. Aus diesem Modul kann das Signal von dem „Frontend“-Rechner gelesen und über eine Ethernet-Verbindung an einen zentralen Datenerfassungsrechner geschickt werden. Das Taktsignal wird von dem „Master Clock“-Modul generiert und zusammen mit dem Synchronisierungssignal an „Slave“-Module geschickt – vgl. [Hor06, S. 32]. Für weiterführende Informationen sei auf [Fal+05] verwiesen.

1.2 Grundideen des Data Mining

In fast jedem Buch, das sich mit dem Thema des Data Mining beschäftigt, wird eine eigene Definition von diesem Begriff geliefert. Nach Meinung des Autors ist die, die in dem Buch von Berry et al. [BL04, S. 7] vorgeschlagen wird, die bestpassende: „Unter Data Mining versteht man das systematische (automatisierte oder halbautomatische) Entdecken und Extrahieren vorher unbekannter Informationszusammenhänge und Regeln aus großen Datenmengen.“ Aber auch diese Beschreibung ist nicht perfekt, da sie nicht klarstellt, was unter einer „großen Datenmenge“ zu verstehen sein soll. Diese Information ist für die Zwecke dieser Arbeit nicht von großer Bedeutung und deswegen wird die formelle Definition des Begriffs damit beendet.

Im Data Mining benutzt man Werkzeuge, die aus drei verschiedenen Be-

reichen stammen:

- Datenbanktechnologie;
- Statistik;
- maschinelles Lernen und künstliche Intelligenz.

Eine wichtige Eigenschaft der Data-Mining-Methoden ist auch, dass sie auf Grund der Testprobe, nicht der statistischen Indikatoren, beurteilt werden. Deswegen entsprechen die Data-Mining-Modelle wirklich den Abhängigkeiten zwischen Eingabewerten und werden nicht auf Grund der abstrakten Parameter entwickelt. Anders formuliert: Das Modell wird an einen Fall angepasst und nicht ein Fall an ein Modell.[1]

Vier Schritte der Data-Mining-Analyse[2]

Im Prinzip kann man vier Grundphasen des Data Mining erkennen:

- Datenvorbereitung;
- Datenexploration;
- Hauptanalyse der Daten;
- Einführung und Benutzung des entwickelten Modells.

Während der Datenvorbereitungsphase entscheidet man, welche Informationen in der weiteren Analyse benutzt werden. Die passenden Daten werden entnommen und deren Korrektheit überprüft. Falls die Daten aus verschiedenen Quellen kommen, werden diese auch in einem gemeinsamen Format vereinheitlicht. Während der nächsten Phase, der Datenexploration, werden die allgemeinen Eigenschaften der Daten, die analysiert werden sollen, geliefert. Die dritte Phase beginnt mit einer Vorauswahl der Methoden, die für die Problemlösung benutzt werden können. Danach werden die erlangten Ergebnisse beurteilt und es wird festgestellt, ob sie akzeptabel sind. Die Einführung und Benutzung des entwickelten Modells wird als letzte Phase betrachtet. Der gesamte Prozess wird in der Abbildung 1.2 dargestellt. Es ist wichtig zu beachten, dass diese Phasen nicht nacheinander stattfinden müssen. Manchmal könnte es notwendig sein, die Voraussetzungen im Laufe des Prozesses zu verändern und zurück statt weiter zu gehen.

[1] Der Satz ist eine Paraphrase des berühmten Zitats von Albert Einstein – „Wenn die Fakten nicht zur Theorie passen, ändere die Fakten“.

[2] Dieser Abschnitt wurde übernommen aus [Fra10, S. 5].

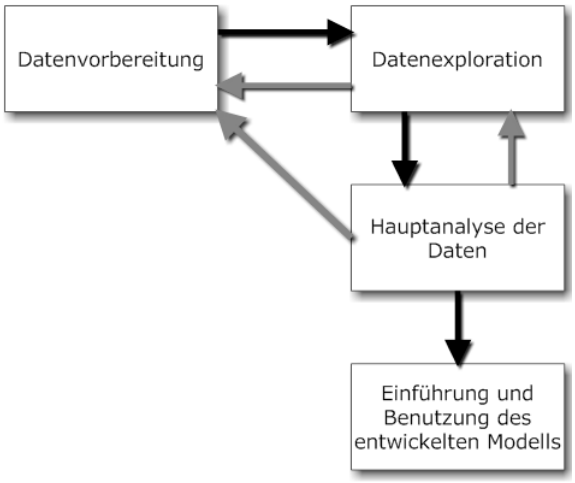

Abbildung 1.2: Grundphasen des Data Mining.

1.3 Data Mining bei LOPES

Man könnte naiv sagen, dass wenn es Daten gibt, Data-Mining-Methoden benutzt werden können. Aus logischer Perspektive ist dieser Satz nicht unbedingt falsch,[3] da die Methoden des Data Mining nach einer vorherigen, geeigneten Konvertierung der Daten fast immer verwendet werden können. Die Hauptfrage, die man stellen könnte, ist, ob solch eine Analyse nützliche Ergebnisse liefert. Man könnte theoretisch ein Data-Mining-Modell bauen, das eine mögliche Abhängigkeit zwischen zwei beliebigen Attributen beschreibt. Nur weil solch ein Modell relativ gut funktioniert, lässt sich dadurch nicht darauf schließen, dass die gefundene Abhängigkeit von Bedeutung für ein konkretes Problem ist. Analog könnte man in der Medizin während der Diagnostizierung einer Krankheit den gesammten Körper des Patienten untersuchen und alle Missbildungen als potenzielle Symptome der Krankheit betrachten. Der Nachteil dieser Methode ist, dass nicht alle Missbildungen auch Ursachen der Krankheit sein müssen und dass manche von ihnen einfach ohne Bedeutung für ein konkretes Problem sind. Ein wichtiges Kriterium während der Data-Mining-Analyse ist also zu wissen, was gefunden werden soll.

Auch kann es passieren, dass die Daten zu starkes Rauschen enthalten, was die Analyse verkompliziert und manchmal auch völlig unmöglich macht. Das

[3] Natürlich ist in der klassischen (booleschen) Logik nicht möglich, dass ein Satz teilweise oder „nicht unbedingt“ richtig ist. Das Adjektiv „logisch“ wurde an dieser Stelle von dem Autor benutzt, um die theoretische Möglichkeit darzustellen.

Problem betrifft sehr oft die aus Experimenten stammenden Daten, meistens wegen der Ungenauigkeit der Messung.

Wenn es um LOPES geht, ist dieses erste Kriterium erfüllt, da die Analyse der Daten immer mit einer Behauptung, die physikalisch begründet sein soll, beginnt. Vom Autor wurde bewiesen (siehe Kapitel 6), dass trotz der Existenz von Rauschen die Data-Mining-Algorithmen angewendet und Modelle, die für die LOPES-Daten funktionieren, gebaut werden können. Es gibt also keine theoretischen Ursachen, die gegen die Integration von Data-Mining-Methoden in LOPES sprechen würden.

1.4 Status quo

Die bisherigen Aufgaben der Physiker kann man in zwei Gruppen unterteilen. Die erste Aufgabengruppe besteht aus der Erkennung potenzieller Beziehungen zwischen den Messwerten[4] sowie der Entwicklung und Bewertung einer mathematischen Formel, mit der diese Beziehungen beschrieben werden können. Wie bereits geschrieben, ist diese Situation einfacher, da man weiß, was gesucht werden soll. Die Aufgaben aus der zweiten Gruppe umfassen die genaue Erklärung der erhaltenen Ergebnisse mit der existierenden physikalischen Theorie. Das Schema dieser Aktivitäten wird mit der Abbildung 1.3 präsentiert.[5] Die Aktivitäten der ersten Gruppe werden hier als Voranalyse benannt.[6] Zurzeit muss diese Voranalyse von Physikern durchgeführt werden, wobei sie rein statistische Methoden benutzen, die sie mit der Programmiersprache C++ und der ROOT-Bibliothek implementieren. Die Forscher beschweren sich oft, dass sie sich mehr auf die Programmierung als auf die eigentliche Physik konzentrieren müssen, da die Entwicklung der Applikation, die sich mit dem Finden der konkreten Beziehungen beschäftigt, länger als ihre Erklärung dauern kann. Grund dafür ist u. a., dass sowohl ROOT keine gute Bibliothek (siehe Abschnitt 4.2.4) als auch C++ keine einfache Programmiersprache ist (siehe Abschnitt 5.2.1).

[4] Bevor statistische Analysen betrieben werden können, müssen die LOPES-Rohdaten aufbereitet werden. Dazu gehören z. B. Kalibration und Unterdrückung des Rauschens.

[5] Dieses Schema stellt eine mögliche Arbeitsweise dar. Eine andere, die auch angewendet wird, ist z. B. die Eigenschaften der Radiopulse anhand theoretischer Modelle vorherzusagen und anschließend diese Vorhersage mit den LOPES-Daten zu testen.

[6] Dieser Name wird nur lokal benutzt und hat keine Verbindung mit der Phase „Voranalyse“ der Data-Mining-Analyse.

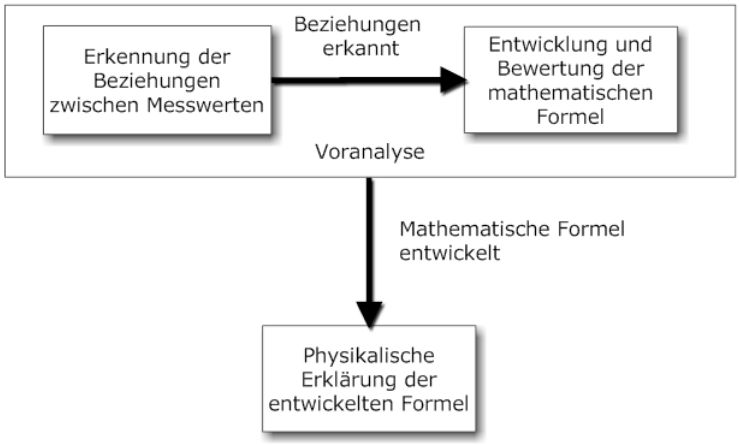

Abbildung 1.3: Schema der bisherigen Analyse-Aktivitäten.

1.5 Verbesserungsmöglichkeiten

Wie vorher geschrieben wurde, besteht zurzeit der erste Teil der Analyse aus der Entwicklung kleinerer Stücke Software, die bestimmte mathematische und statistische Operationen durchführen sollen und als Ausgabe die Formel liefern, die später aus physikalischer Sicht analysiert werden kann. Es gibt aber ein paar Probleme mit der bisherigen Lösung:

- Ein Großteil der Quellcode-Teile wurde meistens „Aufgaben-orientiert“ geschrieben, sodass sie für spätere ähnliche Aufgaben nutzlos sind. Außerdem ist dieser Quellcode auch oft nicht objektorientiert entwickelt (was damit erklärt werden kann, dass vorher Fortran und C für Programmieraufgaben benutzt wurden).

- Die benutzten Tools machen es immer schwieriger, hochqualitative Software zu entwickeln.

Diese Schwierigkeiten induzieren, dass man relativ viel Zeit und Ressourcen braucht, um eine Lösung zu finden. Die generell-formulierte Verbesserungsidee ist, sowohl die Methoden des Data Mining zu benutzen, um die Suche nach Abhängigkeiten zwischen den Messwerten teilweise zu automatisieren, als auch eine hochqualitative Software zu entwickeln, die später auch eventuell ausgebaut werden könnte.

1.6 Aufgabenstellung

Das Ziel der Arbeit ist die Evaluierung der Data-Mining-Methoden, die zur Analyse der LOPES-Daten benutzt werden könnten, die erfolgversprechendsten zu wählen und diese in die entwickelte Software zu integrieren. Darüber hinaus muss die Software die im Kapitel 2 genannten Anforderungen erfüllen.

1.7 Gliederung der Arbeit

Die Themen aus den Bereichen sowohl des Data Mining als auch des Software Engineering und der Programmierung können in dieser Arbeit nicht vollständig behandelt werden, deswegen werden Referenzen zu entsprechender Fachliteratur, meist in Form von Fußnoten, angegeben.[7] Die Arbeit beginnt mit der Darlegung der Anforderungen an die Software und nach kurzer Darstellung der „praktisch orientierten“ Theorie,[8] die mit den benutzten Methoden verbunden ist, konzentriert sie sich auf die Beschreibung des Entwurfs und der Entwicklung der Software. Im letzten Kapitel werden Anwendungsbeispiele präsentiert und Tests der Software mit LOPES-Messdaten durchgeführt. Da der gesamte Software-Engineering-Teil der Arbeit auf den Meinungen der berühmtesten Software-Architekten und -Entwickler der Welt wie Stroustrup, Alexandrescu oder van Rossum basiert,[9] könnte er auch partiell als eine Programmierreferenz betrachtet werden.

[7] Anders geschrieben liegt das Niveau der Arbeit zwischen „Ha“ und „Ri“ nach dem Prinzip des „Shu-Ha-Ri“ (jap. „守破離“) – meint im übertragenen Sinn die Stufen von Lehrling über Geselle zum Meister –, welcher einer der angemessensten Wege zum Erlernen der traditionellen japanischen Künste ist.

[8] Dieses Oxymoron wird hier absichtlich benutzt.

[9] Stroustrup ist der Erfinder der Programmiersprache C++, Alexandrescu wird als einer der größten Experten bzgl. Programmierung und Entwurf in dieser Sprache betrachtet und van Rossum ist der Autor und Hauptentwickler der Skriptsprache Python.

Kapitel 2

Anforderungen an die Software

Wie schon geschrieben wurde, besteht der praktische Teil der Arbeit aus der vom Autor implementierten Software, welche die Benutzung der Data-Mining-Methoden während der Analyse der Daten aus dem LOPES-Experiment erlaubt. Der Auftraggeber der Software ist das Institut für Kernphysik am KIT,[1] das sich mit dem LOPES-Experiment beschäftigt. Außerdem wäre es gewünscht, dass die Software möglicherweise in Zukunft als ein universelles Tool von der Radioastronomie-Gemeinde benutzt werden könnte. Es sollte möglich sein, die Software zusammen mit den bisherigen Komponenten, die teilweise während der Analyse der Daten benutzt werden und die unter dem Namen CR-Tools[2] bekannt sind, zu benutzen. An diesem Punkt soll man deutlich klarstellen, dass die implementierte Data-Mining-Lösung nicht als Konkurrenz betrachtet werden soll, sondern als eine Alternative, die auch parallel mit den bisherigen Methoden benutzt werden kann, um mehrere Sichten auf mögliche Lösungen der Probleme zu bieten. In diesem Kapitel werden sowohl globale als auch rein technische Anforderungen an die Software beschrieben. Nach Meinung des Autors wäre es besser, statt einer langen Liste von Voraussetzungen nur die wichtigsten zu nennen. Diejenigen, die weniger von Bedeutung sind, wurden während der Entwicklung der Software mit den Forschern des KIT nur grob mündlich besprochen. Die hier spezifizierten Anforderungen sind also die bedeutsamsten Punkte, die erfüllt werden sollen.

[1] `http://www-ik.fzk.de`

[2] `http://usg.lofar.org/wiki/doku.php?id=software:packages:cr-tools`

2.1 Funktionale Anforderungen

Es gibt ein paar globale Anforderungen an die Software, die von Forschern des KIT definiert wurden. Die wichtigsten sind:

1. In der einfachsten Form sollte es möglich sein, die Software ohne Data-Mining-Kenntnisse zu benutzen.

2. Es sollte möglich sein, die besten Attribute zu wählen, die eine potenzielle Beziehung aufbauen können. Wäre beispielsweise y ein Zielattribut, so sollte die Software ein Ergebnis liefern, ob dieses Attribut höhere Übereinstimmung in Abhängigkeit von entweder x_1, x_2, x_3, oder x_1, x_3, x_4 hat (wobei $y, x_1, \ldots, x_4$ die Menge der gewählten Attribute ist).

3. Man kann die Performance[3] der neuen Lösung mit der Performance der Formel, die zuvor entwickelt wurde, vergleichen.

4. Die Software soll eine Lösung liefern, die für Physiker verständlich ist, oder im Fall einer Black-Box-Lösung soll deren Performance in allgemeinverständlicher Form (z. B. mittels eines Histogramms des Fehlers) dargestellt werden.

5. Die Software soll so entwickelt werden, dass sie sowohl für spezifische Aufgaben konfiguriert als auch von den Forschern ausgebaut werden kann.

Die Diskussion der Anforderungen soll mit dem Punkt beginnen, dass die Software ohne Data-Mining-Kenntnisse benutzt werden kann. Im Prinzip würde es bedeuten, dass der Benutzer im Normalfall eine Implementierung des Data-Mining-Algorithmus verwendet, aber nicht wissen muss, wie ein Algorithmus funktioniert oder welche Konfigurationsparameter nötig sind, um die besten Ergebnisse zu erhalten. In anderen Worten soll die gesamte Data-Mining-Schicht für den Benutzer transparent sein. Diese Anforderung macht es schwierig, die existierende Data-Mining-Software in der ursprünglichen Form zu benutzen.

Da es möglich sein soll, die Performance der neuen Lösung mit der Performance der zuvor entwickelten Formel zu vergleichen, muss die Software in der Lage sein, sowohl mathematische Gleichungen zu evaluieren als auch diese Ergebnisse mit der Performance der neuen Lösung zu vergleichen. Die

[3] In der Informatik wird Performance oft mit der Geschwindigkeit einer Methode verbunden. In der Data-Mining-Literatur (also auch in dieser Arbeit) wird sie aber als die Zuverlässigkeit der Methode verwendet.

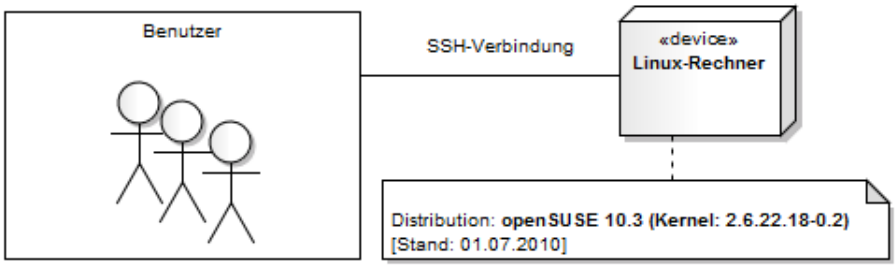

Abbildung 2.1: Schema der Benutzung der bisherigen Analyse-Software.

erste und die dritte Anforderung im Zusammenhang mit der Voraussetzung, dass die Lösung idealerweise für die Physiker verständlich sein sollte, implizieren bedingungslos, dass die beste Möglichkeit wäre, eine Lösung in Form einer mathematischen Formel zu liefern.

Die fünfte Anforderung, dass die Software sowohl für spezifische Aufgaben konfiguriert als auch einfach von den Physikern ausgebaut werden kann, steht theoretisch im Widerspruch dazu, dass das Herz der Software (also die Data-Mining-Analyse) transparent für einen Benutzer durchgeführt werden soll. Diese beiden Punkte können aber durch die Erstellung einer gut dokumentierten Schnittstelle vereinbart werden. Mit dieser wird die Hauptapplikation geschrieben und sie kann in Zukunft benutzt werden, um die Funktionalität der Lösung auszubauen.[4]

Die zweite Anforderung, dass man die Attribute wählen kann, die für eine Beziehung von Bedeutung sind, wird in diesem Moment absichtlich ausgelassen und genauer im Abschnitt 3.1.3 untersucht.

2.2 Technische Anforderungen

Die Diskussion über die technischen Anforderungen beginnt mit der Beschreibung, wie die bisherige Software für die Verarbeitung und Analyse der Messwerte – CR-Tools – benutzt wird. Die Software ist auf einem Linux-Rechner installiert, sodass sich die Benutzer per SSH auf der Maschine anmelden, die gewünschte Analyse durchführen und die Ergebnisse herunterladen können (siehe Abbildung 2.1). Die neu entwickelte Software soll nach dem gleichen Prinzip funktionieren. Deswegen wäre es auch gewünscht, die Anzahl neu-

[4] Diese Schnittstelle wird mit Hilfe der Skriptprache Python geschrieben. Die damit verbundenen Vorteile werden im Abschnitt 4.1 beschrieben.

er Bibliotheken, die installiert werden müssen, auf ein absolutes Minimum zu reduzieren. Idealerweise sollte die neue Software diejenigen benutzen, die schon für CR-Tools installiert wurden.

Eine der technischen Voraussetzungen, die aus der generellen Anforderung, dass die Software zukünftig ausgebaut werden können soll, abgeleitet wurde, ist die Wahl der Programmiersprache, die später eine eventuelle Weiterentwicklung der Software erlaubt. Selbstverständlich muss auch die neue Software im Hinblick auf optimale Entwurfsentscheidungen und Qualität geschrieben werden. Die Forscher haben explizit klargestellt, dass die benutzte Programmiersprache eine ihnen bekannte Sprache sein sollte.[5] Da viele Data-Mining-Pakete in Java geschrieben wurden, macht es diese Anforderung komplizierter (aber auch nicht unmöglich)[6] diese Lösungen zu benutzen, da Java nicht auf der genannten Liste steht. Als Beispiel kann eine der bekanntesten Open-Source-Data-Mining-Umgebungen RapidMiner[7] dienen oder Weka[8] – die Lösung, die sehr genau in dem klassischen Data-Mining-Buch [WF05] beschrieben (und auch teilweise empfohlen) wird. Wie man aber aus den folgenden Kapiteln erfahren kann, ist es auch möglich, eine Software ohne Java zu entwickeln, die alle vorgegebenen Kriterien erfüllt.

Die zweite wichtige technische Anforderung ist, dass die Software nur aus Open-Source-Komponenten bestehen darf, was wegen der großen Auswahl an Open-Source-Lösungen, die als Teile der Software benutzt werden, keine große Beschränkung ist. Die beiden zuvor aufgeführten Beispiele bekannter Data-Mining-Lösungen stehen unter Open-Source-Lizenzen.

Sowohl die Software selbst als auch ihre mittels Doxygen[9] generierte Quellcode-Dokumentation sollten auf Englisch geschrieben werden, da LOPES eine internationale Kollaboration ist.[10]

[5] Die von der KIT-LOPES-Gruppe bevorzugten Sprachen sind u. a. C, C++ und Python.

[6] Man könnte natürlich einen Wrapper für alle notwendigen Java-Funktionen entwickeln und die Benutzung dieser Funktionen aus anderen Programmiersprachen möglich machen.

[7] `http://rapid-i.com`

[8] `http://www.cs.waikato.ac.nz/ml/weka`

[9] `http://www.doxygen.org`

[10] Die Kommentare in den Quellcode-Beispielen dieser Arbeit sind allerdings zum besseren Verständnis auf Deutsch geschrieben.

Kommunikation mit dem Benutzer

Ein wichtiger technischer Aspekt, der von Anfang an klargestellt werden soll, ist, wie der Benutzer mit der Applikation kommunizieren kann. Die drei möglichen Formen der Kommunikation, die nach den ersten Konsultationen mit den Forschern gewählt werden, waren:

- Kommandozeilen-Schnittstelle;
- GUI-Schnittstelle;
- Kommunikation mittels Konfigurationsdateien.

Die Methode der Konfigurationsdateien wurde als erstes abgelehnt, da sie von dem Benutzer Kenntnisse über die eventuelle Struktur der Dateien bedingen würde. Die GUI-Lösung wurde aus einfacheren Gründen abgelehnt – die Forscher haben explizit klargestellt, dass sie vorerst keine grafische Schnittstelle brauchen, sondern eine gut funktionierende Software Priorität haben soll. So kann mehr Zeit darin investiert werden, sich auf die anderen Aspekte der Entwicklung zu konzentrieren. Deswegen wird die ergonomische Kommandozeilen-Schnittstelle implementiert.

Kapitel 3

Auswahl der Data-Mining-Methoden

In diesem Kapitel wird die Auswahl der benutzten Methoden diskutiert. Sowohl die Beschreibung der Methoden des Data Mining als auch die Grundlagen derer Algorithmen können den Quellen, die im Literaturverzeichnis angegeben sind, entnommen werden.

Die Literatur (hauptsächlich [WI98]) schlägt immer vor, dass die einfachsten Lösungen sehr oft ausreichende Resultate liefern. Deswegen wird die Auswahl der Methoden, die in der entwickelten Software benutzt werden, unter Berücksichtigung des KISS-Prinzips[1] durchgeführt. Die Diskussion beginnt mit der Beschreibung der Methoden, die für die Datenvorbereitung benutzt werden.

3.1 Datenvorbereitung

Die Datenvorbereitung erscheint wie ein Thema, das keiner weiteren Erklärung bedarf, sie ist aber genauso wichtig wie alle anderen Phasen des Data Mining. Wenn es um die Daten geht, die aus LOPES stammen, sind die Beschaffenheit und Qualität der Daten die zwei wichtigsten Gründe, warum dieses Thema ernsthaft betrachtet werden soll. Die Daten sind nämlich nicht in tabellarischer Form gespeichert und enthalten, da sie Messwerte sind, oft starkes Rauschen. Deswegen müssen sie zur Standard-Form, die von Data-Mining-Algorithmen akzeptiert wird, transformiert und wenn notwendig auch vom Rauschen befreit werden. Der dritte wichtige Zweck der Datenvorbereitung ist die Selektion der Attribute, welche die besten Ergebnisse liefern kön-

[1] KISS (engl. „Keep it simple, Stupid.“) ist ein Prinzip, dass die einfachsten Lösungen als bevorzugt ansieht.

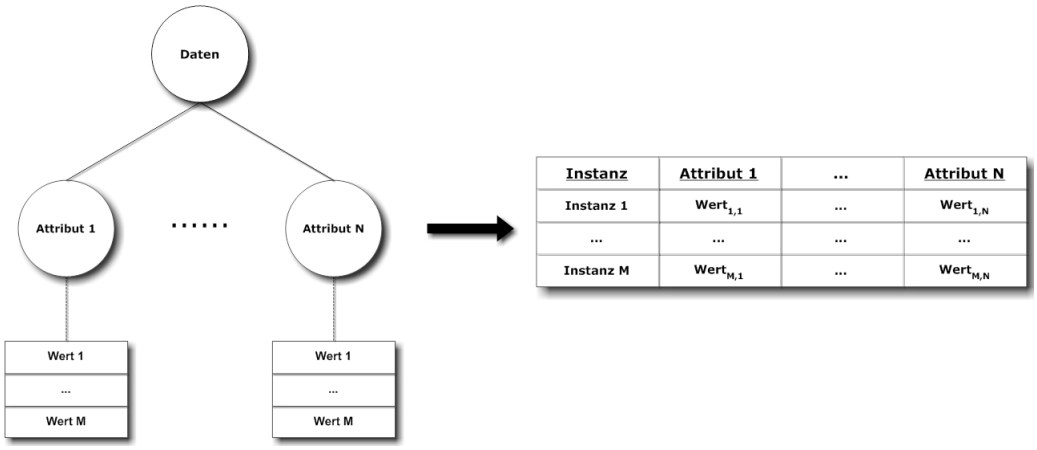

Abbildung 3.1: Schema der Konvertierung der bisherigen Datenstruktur zur Standard-Form.

nen, was auch eine der Anforderungen an die Software ist (siehe Abschnitt 2.1).

3.1.1 Standard-Form

Die beste Definition der Standard-Form wird in dem Buch [WI98, S. 52] geliefert: „Das Konzept der Standard-Form ist mehr als die simple Formatierung der Daten. Eine Standard-Form ist hilfreich, um die Vorteile und Beschränkungen der verschiedenen Analyse-Techniken zu verstehen.“ Die unterschiedlichen Methoden, die für eine Data-Mining-Aufgabe benutzt werden, haben eine allgemeine Eigenschaft – ihr „Weltbild“ –, welche man in Form einer Tabelle darstellen kann. Die Messwerte von LOPES befindet sich in Form einer serialisierten *TTree*-Struktur aus der ROOT-Bibliothek. Die generelle Form dieser Struktur zusammen mit dem Schema der Konvertierungsmethode wird mit der Abbildung 3.1 dargestellt. Sowohl die Struktur als auch der Konvertierungsprozess wird genauer im Abschnitt 4.2.4 beschrieben. In der Standard-Form-Tabelle aus der Abbildung 3.1 repräsentiert jede Zeile eine Instanz der Eingabedaten, die von Attributen (Spalten) charakterisiert wird. $Wert_{i,j}$ ist also der Wert des j-ten Attributs der i-ten Instanz.

3.1.2 Datenglättung

Die Hauptidee des Konzepts der Datenglättung besagt, dass jeder Wert aus der gegebenen Datenmenge, die sich in der Standard-Form befindet, mit der

Gleichung 3.1 dargestellt werden kann, wobei i der i-ten Instanz entspricht.

$$Wert(i) = Mittelwert(i) + Rauschen \tag{3.1}$$

Das Ziel der Methode ist, die ursprüngliche Wertmenge mit einer reduzierten Menge zu substituieren, weil es wahrscheinlich ist, dass solche Werte zu besseren Lösungen führen. Darüber hinaus wird das Rauschen in den Daten reduziert. Das Buch [JM01] nennt vier Techniken, mit denen die Daten geglättet werden können:

- Eingruppierung (engl. „binning“);[2]
- Clustering;
- kombinierte menschliche und maschinelle Kontrolle;
- Regression.

Sowohl die erste als auch die dritte Methode liefern während der Analyse der Daten aus LOPES die besten Ergebnisse.[3] Da das Hauptziel der Software die größtmögliche Automatisierung der Analyse-Prozesse ist, wurde die Methode der Eingruppierung gewählt. Dabei werden die Daten in gleichbreite Intervalle geteilt und später mit einer der zwei folgenden Methoden geglättet:

- Glättung mit dem durchschnittlichen Wert des Intervalls – wenn alle Elemente, die zu einer Gruppe gehören, mit dem durchschnittlichen Wert dieser Gruppe ersetzt werden.
- Glättung mit den Grenzen des Intervalls – wenn alle Elemente, die zu einer Gruppe gehören, mit dem Wert der nähergelegenen Grenze ersetzt werden.

Laut dem Buch [WF05] aber kann, entgegen der ursprünglichen Intention, die Rauschentfernung für eine Methode schädlich sein. Das Paradoxon lässt sich dadurch erklären, dass wenn Rauschen in der Trainingsmenge gar nicht existiert, die Methode einfach nicht lernen kann, wie sie es zu behandeln hat. Dieser Umstand sollte während der Entwicklung des Data-Mining-Modells in Betracht gezogen werden.

[2] In vielen deutschsprachigen Büchern wird für „Binning“ keine Übersetzung vewendet. Die in dieser Arbeit benutzte stammt aus der deutschen Ausgabe der Quelle [WF05] aus dem Jahr 2000.

[3] In diesem Fall bestätigt das die Richtigkeit des KISS-Prinzips, da die Eingruppierungsmethode auch die einfachste ist.

3.1.3 Selektion der Attribute

Die praktischen Experimente[4] haben gezeigt, dass die Data-Mining-Algorithmen sehr oft damit Probleme haben, wenn sie irrelevante Attribute während der Suche nach einer Lösung in Betracht ziehen müssen. Deswegen ist die Benutzung von Methoden zur Eliminierung der unbenutzten Attribute ein wichtiger Teil der Data-Mining-Analyse. Die Literaturquellen beschreiben mehrere Möglichkeiten, mit denen die Anzahl der Attribute reduziert werden könnte. Besonders effektiv sind aber die, die selbst Data-Mining-Methoden benutzen, um die relevanten Attribute zu wählen.[5]

In der entwickelten Software wird die Methode der Entscheidungsbäume verwendet. Die Idee, die hinter der Methode steht, ist die Benutzung der Entscheidungsbäume zur Reduzierung der Attributmenge. Die Attribute, die aus der ursprünglichen Menge gelöscht werden sind die, die sich nicht in der Lösung, die von dem Entscheidungsbaum vorbereitet wird, befinden. Der große Vorteil dieser Lösung ist, dass der Entscheidungsbaum relativ schnell generiert werden kann und ein solches Verfahren die Performance der anderen Methoden, die diese Daten benutzen, beeinflussen kann. An dieser Stelle sollte auch eine andere populäre Methode erwähnt werden – SVM –, die ein Werkzeug anbietet, das die Attribute in eine bestimmte Hierarchie eingliedern kann, die normalerweise von der Methode benutzt wird, um die richtige Klassifikation der Objekte durchzuführen. Die Methode wird in der Software selbst während der Analyse-Phase benutzt, was es sinnlos macht, den gleichen Mechanismus der Attributreduktion doppelt zu verwenden.[6]

Methoden der Datenreduktion

An dieser Stelle ist es sinnvoll das Konzept der Datenreduktion zu beschreiben. „Wenn sich die Daten in Standard-Form befinden, gibt es drei einfache Operationen, die ihre Anzahl reduzieren können:

- Löschen der Zeile, die eine Instanz repräsentiert;
- Löschen der Spalte, die ein Attribut repräsentiert;

[4] U. a. die in den Büchern [WI98] und [WF05] beschriebenen.

[5] Die Verwendung der Data-Mining-Analyse-Methoden während der Datenvorbereitungsphase steht im Widerspruch zu dem Data-Mining-Schema aus Abbildung 1.2. Wie geschrieben, wird diese Strategie aber häufig wegen der Effektivität der beschriebenen Methode benutzt.

[6] Laut [WF05] sollte die Kombination von zwei Methoden (in diesem Fall zwei Mechanismen zur Identifizierung der unbrauchbaren Attribute) bessere Ergebnisse liefern als eine Methode allein.

- Reduktion der Anzahl der möglichen Werte (Diskretisierung).

Die Operationen versuchen sowohl die Dimensionen der Standard-Form zu reduzieren als auch die ursprüngliche Beschaffenheit der Daten zu bewahren – bis auf die dritte Operation, die nur eine Diskretisierung der Daten vornimmt“ [Fra10, S. 16]. Die zwei letzten wurden bereits zuvor in diesem Kapitel beschrieben, sodass nun die erste noch kurz vorgestellt werden soll. Im Prinzip benutzt man verschiedene Methoden von Sampling, um die Anzahl der Attribute zu reduzieren. Die einfachste Lösung ist das einzige Sample, das aus n Instanzen besteht, zu benutzen. Wenn es um die Daten aus LOPES geht, ist es aber sinnvoll, eine Möglichkeit des Samplings auf Grund des Wertes des gewählten Attributs zu verwenden.[7]

3.1.4 Normalisierung

In diesem Abschnitt soll noch das Konzept der Normalisierung, die direkt mit der Datenvorbereitung verbunden ist, kurz beschrieben werden. Manche Methoden des Data Mining funktionieren besser, wenn normalisierte Daten als ihre Eingabe verwendet werden. In der einfachsten Form ist die Normalisierung eine Konvertierung der Eingabewerte, die aus dem Bereich [A, B] stammen, in einen anderen Bereich (meistens [0, 1] oder [-1, 1]). Die einfachste Methode der Normalisierung ist die sog. dezimale Skalierung, die mit der Gleichung 3.2 beschrieben wird.

$$Wert'(i) = \frac{Wert(i)}{10^k} \tag{3.2}$$

Der Parameter k ist der kleinste Wert, für das $max(|Wert'(i)|)$ kleiner als der Grenzwert des Intervalls ist. Die andere Methode (die häufig im Bereich der Bildverarbeitung benutzt wird) ist die Anwendung einer linearen Funktion, die Werte adäquat transformiert (siehe Abbildung 3.2).

3.2 SVM-Methode

SVM (engl. „Support Vector Machine“) ist eine Data-Mining-Methode, die sowohl für Klassifikations- als auch Regressionsprobleme benutzt werden kann. Die Technik wurde von Vapnik entwickelt und zuerst in dem Buch [Vap95] beschrieben. Die Methode wird sehr oft mit der Methode der neuronalen Netze

[7] Solches Attribut müsste dann von dem Forscher, der die Software benutzt, gewählt werden.

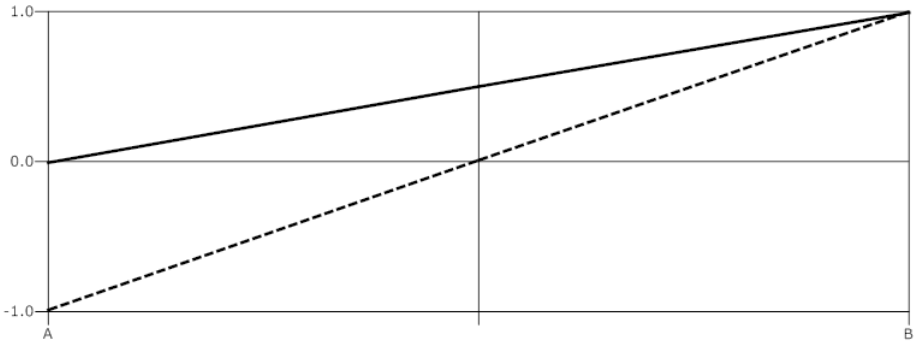

Abbildung 3.2: Lineare Transformationsfunktionen $f_1 : [A, B] \to [-1, 1]$ (gestrichelte Linie) und $f_2 : [A, B] \to [0, 1]$.

verglichen, deswegen wird auch eine kurze Gegenüberstellung der beiden Lösungen im Abschnitt 3.2.2 vorgenommen. Obwohl die Methode selbst relativ einfach zu benutzen ist, müssen jedoch einige Bemerkungen zur Maximierung der Wahrscheinlichkeit akzeptabler Ergebnisse gemacht werden. Diese Bemerkungen sind mit der Auswahl optimaler Parameter und des richtigen Kernels sowie mit Transformationen der Eingabedaten verbunden.

3.2.1 Vorgehen

Ein häufig benutztes (aber leider nicht optimales) Verfahren, wenn es um die Anwendung der SVM-Methode geht, besteht aus der Transformation der Daten in die Standard-Form sowie der beliebigen Auswahl des Kernels und anderer Parameter. Oft kann man aber nicht mit den Ergebnissen zufrieden sein, da auch SVM, wie jede Data-Mining-Methode, ohne korrekte Parameter-Einstellungen nicht die optimale Performance bieten kann. Ein praktischer Algorithmus besteht aus den folgenden Schritten:

1. Transformation der Daten in die Standard-Form;
2. Skalierung der Daten;
3. Auswahl des Kernels (für die Regressionsprobleme kann ein sog. RBF-Kernel[8] akzeptable Ergebnisse liefern);[9]

[8] Radiale Basisfunktion (engl. „Radial Based Function“).

[9] Natürlich ist die RBF-Kernel-Funktion kein Patentrezept für alle Regressionsprobleme. Praktische Experimente haben gezeigt, dass die Funktion eine Tendenz hat schlechte Ergebnisse zu liefern, wenn sich die Anzahl der Attribute deutlich von der Anzahl der Instanzen unterscheidet.

4. Benutzung der Methode der Kreuzvalidierung,[10] um die optimalen Parameter C und γ zu wählen;

5. Anwendung der SVM-Methode mit eingestelltem Kernel und ausgewählten Parametern C und γ auf die gesamte Trainingsmenge.

Normalisierung der Eingabedaten ist besonders wichtig, wenn es um die SVM-Methode geht. Da die Kernel-Funktionen oft von dem Produkt der Eingabewerte abhängig sind, können große Werte Überlauf-Fehler induzieren oder die Kalkulationen zumindest langsamer machen. Das und die weiteren Probleme werden genauer in dem zweiten Teil der Quelle [Sar97] dargestellt.[11] Der bevorzugte RBF-Kernel ist eine Funktion, deren Form mit der Gleichung 3.3 dargestellt wird.[12] Das Ergebnis der Funktion ist direkt abhängig von dem vorher erwähnten Parameter γ.

$$K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}, \gamma > 0 \tag{3.3}$$

Der ebenfalls zuvor aufgeführte Parameter C ist der sog. Bestrafungsparameter (engl. „penalty parameter“). Je höher der Wert des Parameters ist, desto mehr wird der Fehler jeder Instanz der Trainingsmenge „bestraft“. Der Parameter sollte mit Vorsicht gewählt werden, um das Phänomen der Überanpassung (engl. „over fitting“) zu vermeiden.[13]

Die Suche nach dem optimalen Parameterpaar C und γ wird mit der Methode der Kreuzvalidierung durchgeführt. Unterschiedliche Paare (C, γ) werden untersucht und das Paar mit dem besten Kreuzvalidierungsergebnis wird gewählt. Ein praktisches Verfahren, das meist gute Ergebnisse liefert, ist, die Elemente aus den exponentiell steigenden Sequenzen von C und γ zu kombinieren. In der entwickelten Software wurden die folgenden Sequenzen benutzt:[14] $C = 2^{-5}, 2^{-3}, \ldots, 2^{15}$ und $\gamma = 2^{-15}, 2^{-13}, \ldots, 2^{3}$. Obwohl die Me-

[10] Die Kreuzvalidierung ist ein Verfahren, um den Fehler der Data-Mining-Methode abzuschätzen. Die Daten werden in n Teile partitioniert und die Trainingsprozedur wird n mal wiederholt. Jedes Mal wird die Trainingsmenge aus $n-1$ Teilen und die Testmenge aus einem Teil bestehen. Die n Fehlerkalkulationen werden später benutzt, um den Fehler der Methode festzustellen. Viele Quellen, u. a. das Buch [WF05], empfehlen zehn Teile als die optimale praktische Lösung für die Kreuzvalidierung.

[11] Obwohl sich die Quelle mit neuronalen Netzen beschäftigt, ist die Mehrheit der Argumente auch für die SVM-Methode anwendbar.

[12] Für eine weitere Referenz siehe [KL03].

[13] Dieses Phänomen ist damit verbunden, dass je besser eine Lösung zu der speziellen Trainingsmenge passt, desto mehr verliert sie an generellen Eigenschaften und kann deswegen Testdaten nicht gut interpretieren.

[14] Diese Strategie wird von den Autoren der LIBSVM-Bibliothek, die in die Orange-Software integriert ist, empfohlen. Die Orange-Software wird in der entwickelten Software benutzt – siehe Kapitel 4.2.1.

thode, dass alle Möglichkeiten getestet werden müssen, ein wenig naiv zu sein scheint und es attraktiv wäre, sie mit heuristischen Verfahren zu ersetzen, ist die Zeit zur Findung der beiden Parameter relativ kurz. Außerdem kann diese Lösung einfach parallelisiert werden, da die Kreuzvalidierungen unabhängig durchgeführt werden können, was die Suchzeit auf einem Mehrkernprozessor-Rechner reduziert.

3.2.2 Vorteile von SVM gegenüber neuronalen Netzen

Laut [WI98] sind neuronale Netze theoretisch in der Lage, jede komplizierte mathematische Abhängigkeit zwischen den Daten zu modellieren. Allerdings geht das im Jahr 1998 geschriebene Buch nicht auf SVM ein und kann daher keinen Vergleich dazu ziehen. Ebenfalls keine Referenz kann man in dem Klassiker von Berry et al. – [BL99] – und in der ersten, im Jahr 2000 publizierten Version von [WF05] finden. Nach Meinung des Autors war es zu diesem Zeitpunkt einfach „zu früh“, diese Methode, die im Jahr 1995 zuerst publiziert wurde, als Data-Mining-Tool zu betrachten. Zurzeit wird SVM oft als eine Alternative für neuronale Netze genannt.[15] Interessant genug ist der Fakt, dass die Prinzipien der Entwicklung beider Methoden völlig unterschiedlich waren. Den Entwicklungsprozess der ersten Methode kann man als „heuristisch“ bezeichnen, da der Theorie viele Experimente vorausgingen. Im Gegensatz dazu wurde zuerst eine Theorie aufgestellt, als es um die Entwicklung von SVM geht. Die zweite Methode ist theoretisch in manchen Aspekten besser als die Methode der neuronalen Netze. Neuronale Netze haben z. B. ein Problem mit lokalen Minima, SVM hingegen nicht. Viele Experten sehen den Hauptgrund, dass SVM oft bessere Ergebnisse als neuronale Netze liefert, in der größeren Resistenz der Methode gegen das Phänomen der Überanpassung. Es wäre allerdings übertrieben zu behaupten, dass die Methode in der aktuellen Form besser als neuronale Netze sei. Die Methode hat sich allerdings während der Verarbeitung der Daten aus LOPES praktisch besser bewährt und wird deswegen in der Software benutzt.

3.3 Genetische Programmierung

Die genetische Programmierung (GP) wird in der Software benutzt, um mathematische Beziehungen zwischen Eingabedaten zu finden. Der folgende Ab-

[15] Es wird angenommen, dass man unter den neuronalen Netzen die häufig benutzten Feedforward-Netze, die Backpropagation (bzw. Fehlerrückführung) als Lernverfahren verwenden, versteht. Diese Lösung ist beispielsweise innerhalb von RapidMiner implementiert.

schnitt fängt mit den Definitionen von symbolischer Regression und genetischer Programmierung selbst an. Symbolische Regression ist der Prozess der Identifizierung des Modells, der die Eingabedaten bestmöglich anpasst. Für die Zwecke dieser Arbeit wird sie aber als Entwicklung der mathematischen Formel, welche die beste Beziehung zwischen den LOPES-Daten modelliert, betrachtet.[16] Im allgemeinen Sinn versteht man unter genetischer Programmierung den Prozess der symbolischen Regression, der nicht von Menschen sondern von evolutionären Algorithmen[17] durchgeführt wird – vgl. [Zel04]. Eine wichtige Eigenschaft der genetischen Programmierung ist, dass sie problemunabhängig ist, was eine Erstellung des Basis-Ablaufschemas erlaubt. Solches Schema wird mit der Abbildung 3.3 präsentiert.[18] Im Fall der Erstellung der mathematischen Formel benutzt der evolutionäre Algorithmus eine Menge von Terminalsymbolen[19] und eine Menge von Funktionen. Wenn beispielsweise eine unbekannte Funktion der Variable x zu finden ist, wird die Menge der Terminalsymbole T aus dieser Variable und der numerischen Konstante A bestehen: $T = \{x, A\}$. Die Menge der Funktionen F könnte in dem einfachsten Fall aus elementaren arithmetischen Operatoren (Addition, Subtraktion, Multiplikation und Division) bestehen: $F = \{+, -, *, /\}$.[20] Es ist noch wichtig, eine sog. Fitnessfunktion, also eine Funktion, die eine Information über die Qualität der Lösung liefern kann, einzustellen. Je geringer entfernt die Ausgabe der Funktion von 0 ist, desto besser ist eine Lösung. Durch den Algorithmus werden Objekte nach dem Prinzip des Überlebens

[16] Diese Prozedur ist kongruent mit der Definition der Sequenz der Aktivitäten, die in dem Abschnitt 1.2 als „Voranalyse“ bezeichnet wird. Die Schritte aus der Datenvorbereitungsphase dürfen aber nicht vergessen werden.

[17] Ein evolutionärer Algorithmus ist ein Algorithmus, der auf der Evolutionstheorie von Darwin basiert. Obwohl die Ausdrücke „evolutionäre Algorithmen“ und „genetische Algorithmen“ sehr häufig als Synonyme benutzt werden (siehe beispielsweise [BL04]), ist die Bedeutung dieser Begriffe nicht gleich. Stark vereinfacht sind genetische Algorithmen eine Untermenge der evolutionären Algorithmen. Für eine weitere Referenz siehe [Wei02].

[18] In dem Schema sind die genetischen Operationen Rekombination (engl. „crossover“), Reproduktion und Mutation zu sehen. Stark vereinfacht ist Rekombination die Erzeugung eines neuen Programms aus zwei „Eltern“-Programmen, Reproduktion das Kopieren der gewählten individuellen Programme in die neue Population und Mutation die Erzeugung eines Programms aus einem beliebig modifizierten alten Programm. Für eine weitere Referenz siehe [Wei02] und [Koz07].

[19] In der Informatik wird Terminalsymbol mit der Theorie der formalen Sprachen assoziiert als ein Grammatiksymbol, das nicht mit einer Produktion ersetzt werden kann (siehe [Aho+06]). In diesem Kontext besteht die Menge der Terminalsymbole nur aus Variablen und numerischen Konstanten.

[20] Aus praktischen Gründen ist der Algorithmus oft so konfiguriert, dass er 1 als Ergebnis der Division durch 0 liefert. Das kann während der Validierung des Ergebnisses problematisch sein (siehe Abschnitt 5.2.3).

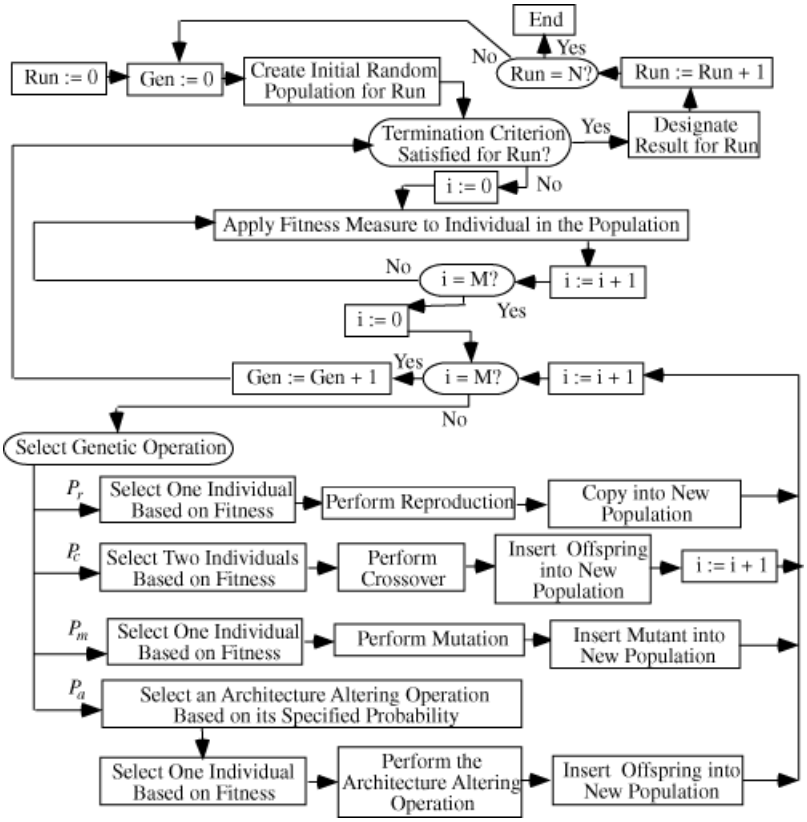

Abbildung 3.3: Basis-Ablaufschema der genetischen Programmierung, Quelle: [Koz07].

der Stärkeren (engl. „survival of the fittest“) gewählt und durch Anwendung genetischer Operationen wird die nächste Generation von Objekten geschaffen. Das Hauptziel ist natürlich die bestpassende Lösung zu entwickeln.

GP als Methode des Data Mining

Man kann diskutieren, ob genetische Programmierung selbst eine Art von Data Mining ist. Das Buch [BL04] klassifiziert evolutionäre Algorithmen als eine der Methoden des Data Mining (mit der Anmerkung, dass diese Methoden relativ selten benutzt werden), trifft aber über genetische Programmierung keine Aussage. Das Ziel der Methode ist aber kongruent mit der Definition des Data Mining (siehe Abschnitt 1.2). Deswegen wird für den Zweck dieser Arbeit als Voraussetzung angesehen, dass die Methode wirklich als eine der Data-Mining-Methoden betrachtet werden kann.

Kapitel 4

Beschreibung der Umsetzungsmethoden

In diesem Kapitel werden die wichtigsten Entscheidungen, die vor dem Softwareentwurf und der Implementierung getroffen wurden, dargestellt und begründet. Dies betrifft die Beschreibung der benutzten Programmiersprachen, Bibliotheken von Drittherstellern, Regeln und Konventionen. Die Diskussion beginnt mit der Auswahl der Programmiersprachen.

4.1 Auswahl der Programmiersprachen

Erste Informationen über die möglichen Programmiersprachen wurden im Abschnitt 2.2 geliefert. Die Entscheidung wurde getroffen, Python als die Hauptsprache der Software zu wählen. Wegen der benutzten Bibliotheken musste aber auch eine Menge interner Komponenten in C++ geschrieben werden. Die Verbindung von den in C++ und in Python geschriebenen Komponenten wurde mit Boost-Bibliotheken hergestellt. Alle verwendeten Bibliotheken werden im Abschnitt 4.2 beschrieben.

Eine der wichtigsten Anforderungen an die Software, die im Abschnitt 2.1 dargestellt wurde, ist, dass sie einfach konfiguriert und ausgebaut werden können soll. Um das zu erreichen, wurde eine Bibliothek von Python-Klassen[1] aufgebaut, die sowohl für die Entwicklung der Hauptapplikation benutzt wurde als auch das einfache Schreiben eigener Skripte erlaubt. Da die Hauptapplikation in Python geschrieben wurde, könnte ihr Quellcode selbst von

[1] Die von dem Autor entwickelte Sammlung von Klassen entspricht der Definition einer Klassenbibliothek aus dem Buch [Wei08].

den Forschern modifiziert werden, um die gewünschte Funktionalität zu erreichen. Man darf nicht vergessen, dass die Zielgruppe der Software sowohl aus Entwicklern als auch technisch fortgeschrittenen Benutzern besteht, die eine genaue Vorstellung davon haben, was sie von der Software erwarten. Die Möglichkeit der beliebigen Modifikationen mit einer Skriptsprache kann diese Erwartungen erfüllen. Die mit einer Skriptsprache entwickelte Applikation ist keine kompilierte „Black Box"-Software, sondern liegt im Quellcode vor, der später beliebig modifiziert werden kann.[2]

Vorteile der Skriptsprachen

Die häufigste Benutzung der Skriptsprachen ist eine Integration und ein eventueller Ausbau der Komponenten, die von ihnen benutzt werden. Deswegen werden diese Sprachen oft als „Klebstoff-Sprachen" (engl. „glue languages") bezeichnet, was auch als der größte Unterschied zwischen Skriptsprachen und System-Programmiersprachen, wie C, C++ oder Java, betrachtet wird. System-Programmiersprachen wurden entwickelt, um die Datenstrukturen und Algorithmen von Grund auf (engl. „from scratch") zu bauen, oft mit den primitivsten Elementen wie ein Datenwort (engl. „word of memory"). Die wichtigsten Vorteile der Skript-Sprachen, die eine schnelle und einfache Entwicklung der Programme (Skripte) erlauben, sind dynamische Typisierung[3] und die große Anzahl der Maschinenbefehle pro Zeile.[4] Ein Vergleich zwischen System-Programmiersprachen und Skriptsprachen im Bereich von diesen zwei Kategorien wird mit der Abbildung 4.1 und ein anderer kleiner Benchmark mit der Abbildung 4.2 illustriert. Jede Zeile in der Tabelle beschreibt eine Applikation, die einmal mit einer der System-Programmiersprachen und einmal mit der Tcl-Skriptsprache[5] implementiert wurde. Das Quellcode-Verhältnis (engl. „code ratio") liefert eine Information über den Quotienten der Anzahl

[2] Eine ähnliche Funktionalität könnte mit der Lösung, die Konfigurationsdateien zur Steuerung der Applikation benutzt, erreicht werden. Diese Lösung wurde abgelehnt, da zur Entwicklung eines fortgeschrittenen Kontroll-Mechanismus solche Dateien eine komplizierte Struktur hätten, die auch später von den Forschern erlernt werden müsste.

[3] Die Festlegung des Typs einer Variablen während der Laufzeit eines Programms.

[4] Eine typische Anweisung einer Skriptsprache wird in tausende und eine typische Anweisung einer System-Programmiersprache in ca. fünf Maschinenbefehle übersetzt. Diese Differenz liegt teilweise daran, dass die Skriptsprachen interpretiert werden müssen, was weniger effizient als der kompilierte Quellcode der System-Programmiersprachen ist. Die Hauptursache dieser Differenz ist aber, dass die primitiven Anweisungen der Skriptsprachen größere Funktionalität haben.

[5] Tcl (engl. „Tool Command Language", häufig „tickle" genannt) ist eine der Skriptsprachen mit Python-ähnlicher Syntax.

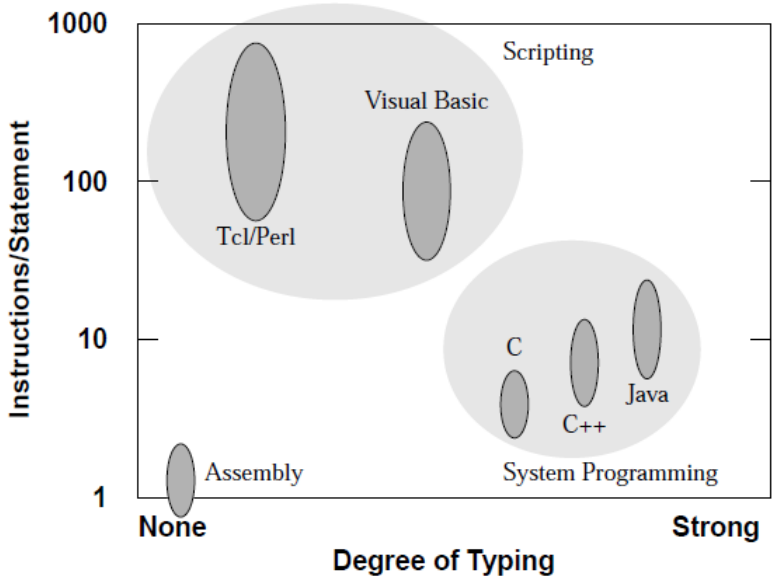

Abbildung 4.1: Vergleich verschiedener Programmiersprachen auf Grund der Anzahl der Maschinenbefehle pro Anweisung und Grad der Typisierung, Quelle: [Ous98].

der Quellcode-Zeilen (engl. „LOC – lines of code“) beider Implementationen. Wenn diese Zahl größer als Eins ist, bedeutet das, dass die Lösung mit einer System-Programmiersprache mehr „LOC“ brauchte. Das Aufwandsverhältnis (engl. „effort ratio“) liefert eine Information über das Verhältnis zwischen den Zeiten, die notwendig waren, um die beiden Lösungen zu entwickeln. Die Informationen wurden von verschiedenen Tcl-Entwicklern als Antwort auf einen Artikel in der „comp.lang.tcl“-Newsgroup[6] publiziert.

Die gezeigten Beispiele präsentieren eindeutig das große Potenzial der Skriptsprachen, das in der entwickelten Software genutzt wird.

4.2 Auswahl der Komponenten von Drittherstellern

In diesem Kapitel werden alle Dritthersteller-Komponenten, die in der entwickelten Software benutzt werden, beschrieben. Die Beschreibung beginnt mit den Data-Mining-Komponenten.

[6] `http://wiki.tcl.tk/844`

Application (Contributor)	Comparison	Code Ratio	Effort Ratio	Comments
Database application (Ken Corey)	C++ version: 2 months Tcl version:1 day		60	C++ version implemented first; Tcl version had more functionality.
Computer system test and installation (Andy Belsey)	C test application: 272 Klines, 120 months. C FIS application: 90 Klines, 60 months. Tcl/Perl version: 7.7K lines, 8 months	47	22	C version implemented first. Tcl/Perl version replaced both C applications.
Database library (Ken Corey)	C++ version: 2-3 months Tcl version: 1 week		8-12	C++ version implemented first.
Security scanner (Jim Graham)	C version: 3000 lines Tcl version: 300 lines	10		C version implemented first. Tcl version had more functionality.
Display oil well production curves (Dan Schenck)	C version: 3 months Tcl version: 2 weeks		6	Tcl version implemented first.
Query dispatcher (Paul Healy)	C version: 1200 lines, 4-8 weeks Tcl version: 500 lines, 1 week	2.5	4-8	C version implemented first, uncommented. Tcl version had comments, more functionality.
Spreadsheet tool	C version: 1460 lines Tcl version: 380 lines	4		Tcl version implemented first.
Simulator and GUI (Randy Wang)	Java version: 3400 lines, 3-4 weeks. Tcl version: 1600 lines, < 1 week.	2	3-4	Tcl version had 10-20% more functionality, was implemented first.

Abbildung 4.2: Vergleich zwischen Skript- und System-Programmiersprachen auf Grund der Anzahl der Quellcode-Zeilen und der zur Anwendungsentwicklung benötigten Zeit, Quelle: [Ous98].

4.2.1 Data-Mining-Softwarepakete

Eine der Möglichkeiten, die im Kapitel 3 beschriebenen Data-Mining-Methoden zu verwenden, wäre ihre Implementierung von Grund auf. Dieser Ansatz würde allerdings bedeuten, das „Rad neu zu erfinden“, da es eine Menge fertiger Softwarepakete gibt, die für die bestimmten Aspekte der Data-Mining-Analyse benutzt werden können. Im Abschnitt 2.2 wurden schon zwei Data-Mining-Lösungen genannt: RapidMiner und Weka. Beide Lösungen wurden in Java implementiert und erlauben die Entwicklung eigener Programme nicht nur direkt mit der GUI sondern auch mit Hilfe ihrer Bibliotheken direkt im Java-Quellcode. Wie aber schon im Abschnitt 2.2 erwähnt, würde diese Lösung die Implementierung eines Wrappers, der einen

Aufruf der Java-Funktionen aus Python erlaubt, voraussetzen.[7] Es existiert aber Data-Mining-Software, die direkt Python-Scripting erlaubt. Solch eine Software wäre z. B. Orange, die vom Artificial Intelligence Laboratory der Universität Ljubljana entwickelt wurde.[8] Da die Software unter GNU GPL[9] lizenziert ist, wurde der Quellcode der Software veröffentlicht, was auch eine tiefere Analyse der benutzten Lösungen durch den Autor erlaubte. Die Module von Orange, die am häufigsten benutzt werden, sind die, welche für die Durchführung sowohl der SVM-Methode als auch der Methode der Entscheidungsbäume verantwortlich sind. Wie vorher erwähnt wurde, wird die Funktionalität des SVM-Modells von der integrierten LIBSVM-Bibliothek[10] angeboten.

Ein großer Vorteil von Orange ist, dass Python als Zugriffssprache benutzt wird und alle „aufwendigen“ Operationen in C++ durchgeführt werden.[11] Obwohl die Software hauptsächlich für die Familie der Windows-Betriebssysteme unterstützt wird, kann man den Quellcode selbst kompilieren und dann die notwendingen Elemente auch unter anderen Betriebssystemen verwenden. Da nur der Scripting-Teil in der implementierten Software verwendet wird,[12] reduziert sich die Installation der Software auf die Installation eines Python-Paketes.[13] Die Software hat aber auch ein paar Nachteile. U. a. sind das die fehlende Dokumentation der verschiedenen Module und die Nichtbeachtung der Python-Code-Konventionen – siehe [RW09].

Da im Abschnitt 3.3 die Voraussetzung formuliert wurde, dass die genetische Programmierung als Data-Mining-Methode betrachtet werden kann, wird in diesem Abschnitt eine Software vorgestellt, die eine eigene Implementierung von genetischer Programmierung anbietet. Sie heißt Eureqa[14] und wurde vom Cornell Computational Synthesis Laboratory der Cornell-Universität entwickelt.[15] Sie besteht sowohl aus einer GUI-Applikation als auch einem in C++

[7] Eine der Lösungen, die dafür benutzt werden könnte, ist das Jython-Projekt. Siehe `http://www.jython.org` für eine weitere Referenz.

[8] `http://www.ailab.si/orange`

[9] `http://www.gnu.org/licenses/gpl.html`

[10] `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

[11] Ein ähnliches Prinzip wurde vom Autor während der Entwicklung der Software benutzt. Siehe Abschnitt 5.2.2.

[12] Die Software erlaubt auch die Entwicklung eigener Programme direkt mit der GUI.

[13] Die Operation könnte durch Kopieren der Dateien in ein Verzeichnis und Einstellung der PYTHONPATH-Variable ersetzt werden, wenn die Pakete nicht im Standard-Verzeichnis installiert werden sollen (z. B. wegen fehlender Administrator-Privilegien).

[14] Für eine weitere Referenz siehe [SL09].

[15] `http://ccsl.mae.cornell.edu/eureqa`

geschriebenen API,[16] das in der entwickelten Software benutzt wird. Eureqa kann unter der neuen BSD-Lizenz[17] benutzt werden. Der nach Meinung des Autors größte Vorteil der Software ist, dass der Quellcode hochqualitativ geschrieben und die Boost-Bibliotheken häufig verwendet wurden. Eine Alternative wäre beispielsweise das in Java entwickelte Distributed Genetic Programming Framework (DGPF).[18] Obwohl die Funktionalität des Eureqa API nicht mit der, die von DGPF angeboten wird, verglichen werden könnte, entspricht sie genau den Erwartungen – sie findet eine möglichst optimale mathematische Formel, die Abhängigkeiten zwischen den Daten modelliert – und stellt somit eine dem Zweck angemessene Lösung dar. Das Eureqa API ermöglicht die Suche nach Beziehungen unter Benutzung verschiedener Server. Im einfachsten Fall soll solch ein Server auf dem lokalen Rechner laufen, bevor das API verwendet werden kann. Obwohl die Bibliothek ausschließlich aus Headerdateien besteht (engl. „header only“), wurde die Binärdatei für den Server, der für die Durchführung der genetischen Kalkulationen verwendet wird, als Teil dieser Bibliothek ausgeliefert. Diese Datei muss zuerst gestartet werden, um die Benutzung des APIs zu ermöglichen, was die gesamte Prozedur verkompliziert.

4.2.2 Boost-Bibliotheken

Die Boost-Bibliotheken sind eine Menge moderner, auf dem derzeitigen C++-Standard basierender Bibliotheken. Verantwortlich für deren Entwicklung und Veröffentlichung ist die Boost-Community. Es handelt sich hierbei um eine große Gruppe von C++-Entwicklern aus aller Welt, die sich online über die offizielle Website[19] und Mailinglisten koordinieren. Ziel dieser Community ist es, qualitativ hochwertige Bibliotheken zu entwickeln, die den C++-Standard ergänzen – vgl. [Sch10]. Die Bibliotheken sind auch Teil des neuen Standards,[20] der im Jahr 2011 als Ersatz des aktuellen Standards (siehe [SC03]) veröffentlicht werden soll. Die Bibliotheken sind nach Meinung des Autors neben der STL (Standard Template Library)[21] fester Bestandteil der modernen Programmierung in C++. Sie können unter der Boost-Softwarelizenz[22] benutzt werden, was garantiert, dass man sie ohne weitere Beschränkungen verändern und verbreiten kann.

[16] `http://code.google.com/p/eureqa-api`
[17] `http://www.opensource.org/licenses/bsd-license.php`
[18] `http://dgpf.sourceforge.net`
[19] `http://www.boost.org`
[20] Der Standard ist zurzeit unter dem inoffiziellen Namen „C++0x“ bekannt.
[21] `http://www.sgi.com/tech/stl`
[22] `http://www.boost.org/LICENSE_1_0.txt`

Die überwiegende Anzahl der Bibliotheken ist vom Typ „header-only“, die anderen müssen zuerst kompiliert und später vor der Benutzung in externen Programmen mit einem Linker verknüpfen werden. Die Liste der benutzten Bibliotheken besteht aus:

- Any (ermöglicht die Benutzung einer generischen Komponente);
- Exception (Ausnahme-Behandlung);
- Foreach (ermöglicht die Benutzung des *foreach*-Iterators);
- Python (stellt die Verbindung zwischen in Python und C++ geschriebenen Komponenten her);
- Thread (ermöglicht die sichere Durchführung paralleler Operationen);
- uBLAS (ermöglicht die Benutzung der Matrix-Struktur);
- Utility (bietet u. a. die Möglichkeit, Singleton-Klassen zu entwerfen).

4.2.3 Python-Pakete

Die zwei Pakete, die standardmäßig nicht in der benutzten Version von Python[23] eingesetzt werden, sind NumPy[24] und matplotlib.[25] Das erste Paket wird wegen der Orange-Software benutzt, da es notwendig ist, um die numerische Regression mit ihr durchzuführen. Das zweite wird für die grafische Aufbereitung der Ergebnisse benutzt.

4.2.4 ROOT-Bibliothek

Diese Bibliothek ist eine der am häufigsten benutzten sowohl innerhalb des LOPES-Experiments als auch aller anderen Projekte, die von der Radioastronomie-Gemeinde durchgeführt werden. Alle Daten werden mit ihr bearbeitet und als ihre serialisierten Strukturen gespeichert. Die Benutzung der Bibliothek wurde aber in dem entwickelten Quellcode wegen folgender Gründe auf ein absolutes Minimum beschränkt:

- Die Schnittstellen der Klassen sind typische Beispiele von „Do-It-All Interfaces“ [Ale01, S. 4], was sie unnötig groß und unverständlich macht.

[23] Die Software benutzt die Version 2.6 des Interpreters.
[24] `http://numpy.scipy.org`
[25] `http://matplotlib.sourceforge.net`

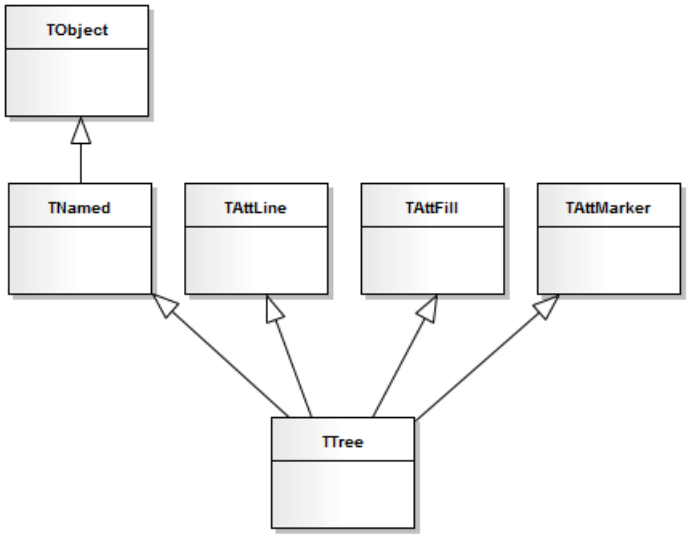

Abbildung 4.3: Vererbungshierarchie für die *TTree*-Klasse.

- Die Klassenhierarchie wurde immer nach dem Prinzip der Vererbung (engl. „is a“) entwickelt.
- Bereits einfachste Operationen induzieren Speicherlecke (engl. „memory leaks“).

Als ein Beispiel wurde die Klasse *TTree* genommen, die auch mit der Abbildung 4.5 auf Seite 38 genauer dargestellt wird. Die Klasse wurde aus den fünf Klassen abgeleitet, obwohl sie eigentlich nur eine Spezialisierung von *TObject* ist (siehe Abbildung 4.3), was auch im Gegensatz zu einem wichtigen Entwurfsprinzip steht, dass Aggregation gegenüber Vererbung bevorzugt werden soll – siehe [Ale04]. Die Klasse selbst bietet über hundert Methoden an, die benutzt werden können. Außerdem besteht die Schnittstelle der Klasse auf Grund der Vererbung aus allen öffentlichen Funktionen der Eltern-Klassen und bietet somit Funktionalität an, die nicht ihrem Zweck entspricht.

In Abbildung 4.4 werden die Ergebnisse des Speicherleck-Tests, der mit dem Valgrind-Programm[26] durchgeführt wurde, präsentiert.[27] Der Quellcode, der getestet wurde, bestand nur aus einer Anweisung, welche die Header der Bibliothek inkludiert hat.

[26] Valgrind ist eine Werkzeugsammlung zur dynamischen Fehleranalyse von Programmen. Siehe `http://valgrind.org`

[27] Als System kommt ein PC mit Intel Core 2 Duo 2,26 GHz, 2048 MB RAM unter Ubuntu 9.10 (Kernel-Version: 2.6.31-22) zum Einsatz. Wenn nicht anders erwähnt, wird dieses System auch für alle späteren Tests dieser Art benutzt.

```
==6464== Memcheck, a memory error detector
==6464== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==6464== Using Valgrind-3.5.0-Debian and LibVEX; rerun with -h for copyright info
==6464== Command: ./root-test
==6464==
==6464==
==6464== HEAP SUMMARY:
==6464==     in use at exit: 871,511 bytes in 18,261 blocks
==6464==   total heap usage: 37,860 allocs, 19,599 frees, 1,732,632 bytes allocated
==6464==
==6464== LEAK SUMMARY:
==6464==    definitely lost: 240 bytes in 2 blocks
==6464==    indirectly lost: 120 bytes in 10 blocks
==6464==      possibly lost: 34,848 bytes in 961 blocks
==6464==    still reachable: 836,303 bytes in 17,288 blocks
==6464==         suppressed: 0 bytes in 0 blocks
==6464== Rerun with --leak-check=full to see details of leaked memory
==6464==
==6464== For counts of detected and suppressed errors, rerun with: -v
==6464== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 63 from 8)
```

Abbildung 4.4: Ergebnisse des Speicherleck-Tests, der mit dem Valgrind-Programm durchgeführt wurde.

Grund der Nutzung der Bibliothek

Man könnte sich die Frage stellen, warum die Bibliothek, die so viele negative Merkmale hat, überhaupt in der neu entwickelten Software benutzt wird. Es existiert keine Dokumentation, die genügend Informationen über den Serialisierungsprozess der *TTree*-Struktur liefert. Wegen der Komplexität der Struktur ist es auch nicht trivial, die notwendigen Informationen mit Methoden des Reverse Engineering[28] zu erhalten. Da die ROOT-Bibliothek trotzdem in der Zielumgebung der neu entwickelten Applikation installiert wird, wäre es nicht besonders produktiv, einen Großteil der Zeit damit zu verbringen, eine Unabhängigkeit von ihr zu erreichen.

4.2.5 Zusammenfassung

Im Kapitel 2.2 wurde geschrieben, dass es besonders wünschenswert wäre, die Anzahl der Dritthersteller-Bibliotheken zu minimieren. Dieses Ziel wurde dadurch erreicht, dass nur die Python-Module von Orange zusätzlich installiert werden müssen.[29] Wie im Abschnitt 4.2.1 schon bemerkt wurde, kann die Installation von Orange mit der richtigen Einstellung der Python-Umgebungsvariable umgangen werden.

[28] In diesem Fall wurden Dateien mit einem Hex-Editor analysiert.

[29] Die Liste der für die CR-Tools notwendigen und daher bereits installierten Software ist unter `http://usg.lofar.org/wiki/doku.php?id=software:packages:cr-tools:installation_of_cr-tools` zu finden.

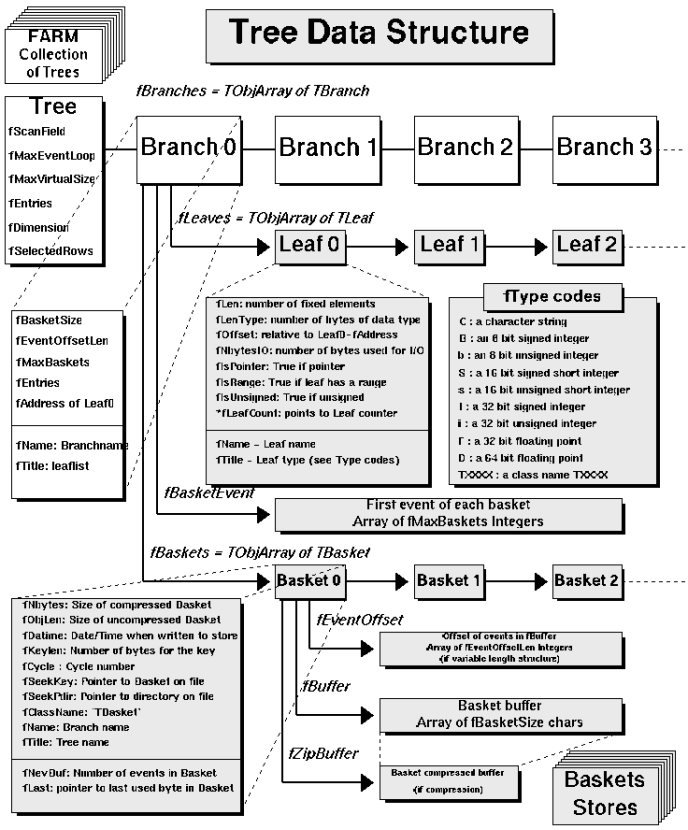

Abbildung 4.5: Struktur der *TTree*-Klasse. Quelle: `http://root.cern.ch/root/html/gif/tree_layout.gif`

4.3 Prinzipien und Konventionen

In diesem Abschnitt werden die wichtigsten Prinzipien, nach denen der Entwurf und die Entwicklung der Software durchgeführt wurde, beschrieben. Die Diskussion beginnt mit einer kurzen Zusammenfassung der wichtigsten Eigenschaften hochqualitativer und zuverlässiger Software.

4.3.1 Wichtigste Eigenschaften hochqualitativer und zuverlässiger Software

In seinem Buch [McC04] präsentiert McConnell eine Liste von Prinzipien des hochqualitativen Entwurfs, die aus den folgenden Eigenschaften besteht:

- minimale Komplexität;
- gute Wartbarkeit;

- minimale Abhängigkeiten zwischen Komponenten;
- Erweiterbarkeit;
- Wiederverwendbarkeit;
- hohes „Fan-In“;[30]
- geringes „Fan-Out“;
- Portabilität;
- Schlankheit (engl. „leanness“);[31]
- Schichtenbildung;[32]
- Benutzung von Standardtechniken.

McConnell selbst macht aber eine Anmerkung, dass die Verbindung dieser Eigenschaften schwer zu erreichen ist. Trotzdem versucht der Autor bei der Entwicklung der Software möglichst alle dieser Anforderungen zu erfüllen. Die Software selbst wurde unter Beachtung der Prinzipien des objektorientierten Entwurfs entwickelt und immer, wenn möglich, wurden Entwurfsmuster[33] als eine der Standardtechniken benutzt, die es einfacher machen, Software von Dritten zu verstehen.

Eine der wichtigsten Eigenschaften des entwickelten Quellcodes ist, dass er mit den Methoden des defensiven Programmierens geschrieben wurde. Die Benutzung dieser Methoden kann in erster Linie als ein Dämpfer des Ausmaßes von Murphys Gesetz angesehen werden.[34] Der zweite Grund dafür wird in dem Buch [HL02] von Howard et al. benannt. Obwohl sich dieses Buch überwiegend mit Themen des Entwurfs sicherer Software beschäftigt, ist der Entwurf von solcher, die gegen verschiedene Hack-Methoden[35] resistent ist,

[30] Bedeutet eine große Anzahl von Klassen, die eine bestimmte Klasse benutzen. Der Gegenbegriff ist das sog. „Fan-Out“.

[31] Bedeutet, dass die Software keine überflüssigen Teile hat.

[32] Bedeutet, dass die Software aus verschiedenen Sichten betrachtet werden kann.

[33] Siehe auch Kapitel 5, [Gam+95] und [Sav97] für weitere Referenzen.

[34] Murphys Gesetz ist die Aussage des amerikanischen Ingenieurs über die Konsequenzen der Fehler in komplexen Systemen: „Alles, was schiefgehen kann, wird auch schiefgehen“.

[35] Als Beispiel kann man die Pufferüberlauf-Methode (engl. „buffer overrun“) nennen, die auf einem Fehler des Entwicklers basiert, der mit der Unterschätzung der Größe des Puffers verbunden ist.

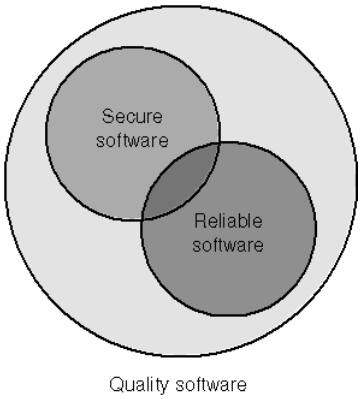

Abbildung 4.6: Sicherheit der Software ist von der Qualität und Zuverlässigkeit abhängig, Quelle: [HL02].

abhängig von dem Entwurf hochqualitativer und zuverlässiger Software (siehe Abbildung 4.6). Eine wichtige Eigenschaft solcher Software ist, dass jede Eingabe überprüft werden soll, sodass kein undefinierter Zustand eintreten kann. Howard et al. stellen das mit der Aussage „alle Eingaben sind böse“ (engl. „all input is evil“) fest.

Wenn es um die Entwicklung in der Programmiersprache C++ selbst geht, wurde der Quellcode der Software unter Beachtung sowohl des Standards [SC03] als auch u. a. der Entwurfs- und Programmierempfehlungen aus [Str01], [Ale04] und [Mey98] geschrieben. Da C++ die dynamische Speicherverwaltung erlaubt, muss sichergestellt werden, dass die benutzten Ressourcen immer korrekt freigegeben werden. In dem entwickelten Quellcode wurde das durch Benutzung der Implementierung von intelligenten Zeigern (engl. „smart pointers“)[36] der Boost-Bibliotheken erreicht. Die entwickelten Klassen sind deswegen frei von Speicherlecken. Auch wurden alle Operationen, die mit der Typumwandlung (engl. „cast“) undefinierter Objekte verbunden sind, mit den Methoden der dynamischen Typverifikation gesichert.

Wichtige Teile der Software-Entwicklung sind auch Optimierung und Refactoring, die u. a. in den Quellen [Fow+99] und [Ise98] beschrieben werden. Ein weiterer und letzter Teil besteht aus sog. „Kleinigkeiten“, die weder Performance noch Zuverlässigkeit deutlich verbessern können, aber nach Meinung des Autors wichtig für einen Entwickler sind, der immer versucht, seinen Quellcode möglichst perfekt zu schreiben. Beispiele dieser „Kleinigkeiten“ für

[36] Der intelligente Zeiger stellt sicher, dass alle Ressourcen im Fall einer Exception oder nach vollständigem Ablauf des Programms freigegeben werden.

C++ und Python werden mit den Listings 4.1 und 4.2 dargestellt.

Das erste Beispiel zeigt die sog. Schleifenpartitionierung (engl. „loop partitioning“), die von verschiedenen Autoren oft als eine Optimierungstechnik betrachtet wird. Für alle Entwickler, die Kenntnisse von Compilern haben, ist es offensichtlich, dass die erste *for*-Schleife wegen der Bedingung, die für jede Iteration überprüft werden muss, von einem Compiler nicht trivial optimiert werden kann. Nach Meinung des Autors sollte diese Methode nicht als Optimierungstechnik sondern als obligatorische Regel betrachtet werden.

```
// Version 1
for(unsigned int i=0; i<size; ++i)
    if(i == size-1)
        g(i);
    else
        f(i);

// Version 2
for(unsigned int i=0; i<size-1; ++i)
    f(i);
g(size-1);
```

Listing 4.1: Beispiel der Schleifenpartitionierung in C++.

Wenn es um das zweite Beispiel geht, öffnen viele Python-Skripte, die man als Beispiele im Internet finden kann, eine Datei genauso wie in Version 1 des Listings 4.2 gezeigt wird. Dies ist problematisch, wenn diese Datei sofort nach Ausführung des Skriptes benötigt wird. Die erste Version lässt diese Datei für unbestimmte Zeit geöffnet, wobei die zweite sie schließt und alle Ressourcen sofort nach dem Ende des *with*-Blocks freigibt.

```
# Version 1
for line in open('example.txt'):
    print line

# Version 2
with open('example.txt') as file:
    for line in file:
        print line
```

Listing 4.2: Beispiel der Anwendung von Pythons *with*-Anweisung.

4.3.2 Konventionen

Neben dem Entwurf ist es auch wichtig, dass der Quellcode verständlich für Personen ist, die ihn lesen. Um das zu garantieren, kamen die folgenden Konventionen zur Anwendung:

- Der Quellcode wurde mit Doxygen-Kommentaren dokumentiert, sodass man eine Dokumentation entsprechend der eigenen Bedürfnisse generieren kann.
- Der Python-Quellcode wird in Übereinstimmung mit den offiziellen Code-Konventionen [RW09] entwickelt.
- Da es im Standard von C++ keine Informationen über den Stil, mit dem Quellcode entwickelt werden soll, gibt und der Autor der Sprache selbst nur von der Benutzung der ungarischen Notation abrät (siehe [Str09]), wurde der Stil des Entwicklers benutzt, der zum Großteil dem in [Hof08] beschriebenen entspricht.

Kapitel 5

Entwurfsentscheidungen

In diesem Kapitel werden sowohl die Struktur der Software als auch die Entscheidungen, die während des Entwurfs und der Implementierung getroffen wurden, beschrieben. Auf Grund des Umfangs der Arbeit war es unmöglich, jeden Teil der Software zu beschreiben und dieses Kapitel konzentriert sich nur auf die Teile, die nach Meinung des Autors besonders wichtig sind.

5.1 Struktur der Software

Die Software besteht aus drei Hauptgruppen von Komponenten:

- in C++ geschriebene;
- in Python geschriebene;
- als Wrapper von C++-Objekten dienende, sodass sie implizit aus Python aufrufbar sind (Python-Bindings).

Diese Komponenten, zusammen mit den Bibliotheken, die sie benutzen, werden mit der Abbildung 5.1 präsentiert. Es wurde die Konvention benutzt, dass die Namen aller in C++ entwickelten Komponenten, die von Python aufrufbar sind, mit „__“ und deren Wrapper mit „_“ beginnen. Beispielsweise *__RootTreeConverter* (C++), *_RootTreeConverter* (Python-Bindings) und *RootTreeConverter* (Python).

5.2 Software-Komponenten

Die Diskussion über die Software-Komponenten beginnt mit den Elementen, die in C++ geschrieben wurden.

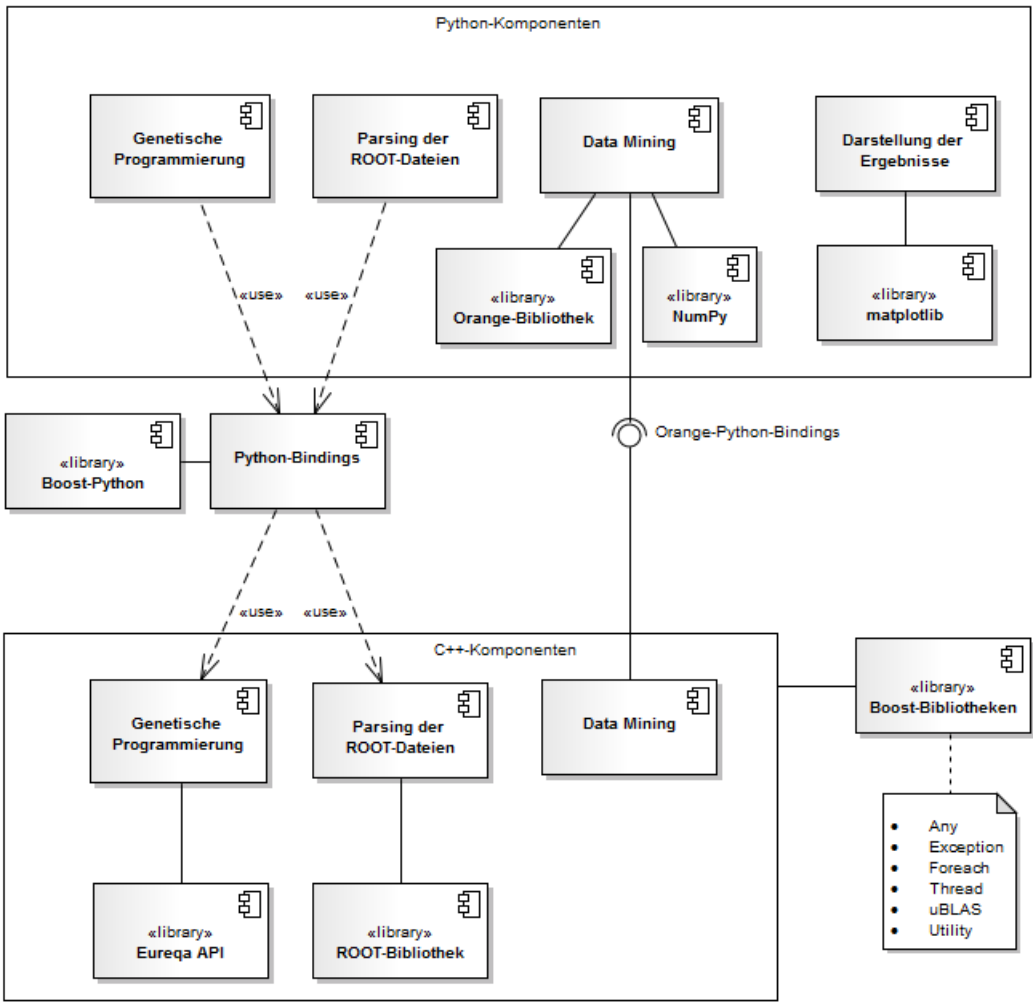

Abbildung 5.1: Überblick der wichtigsten Software-Komponenten.

5.2.1 C++-Komponenten

Der Teil des Quellcodes, der in C++ geschrieben wurde, besteht insgesamt aus drei Hauptgruppen von Objekten. Solche,

- die mit der genetischen Programmierung verbunden sind und die benötigte Funktionalität des Eureqa API (siehe Abschnitt 4.2.1) anbieten;
- die das Parsing der ROOT-Dateien erlauben;
- die als Datenstrukturen dienen.

Die Beschreibung beginnt mit den Objekten, welche die Funktionalität des Eureqa API kapseln.

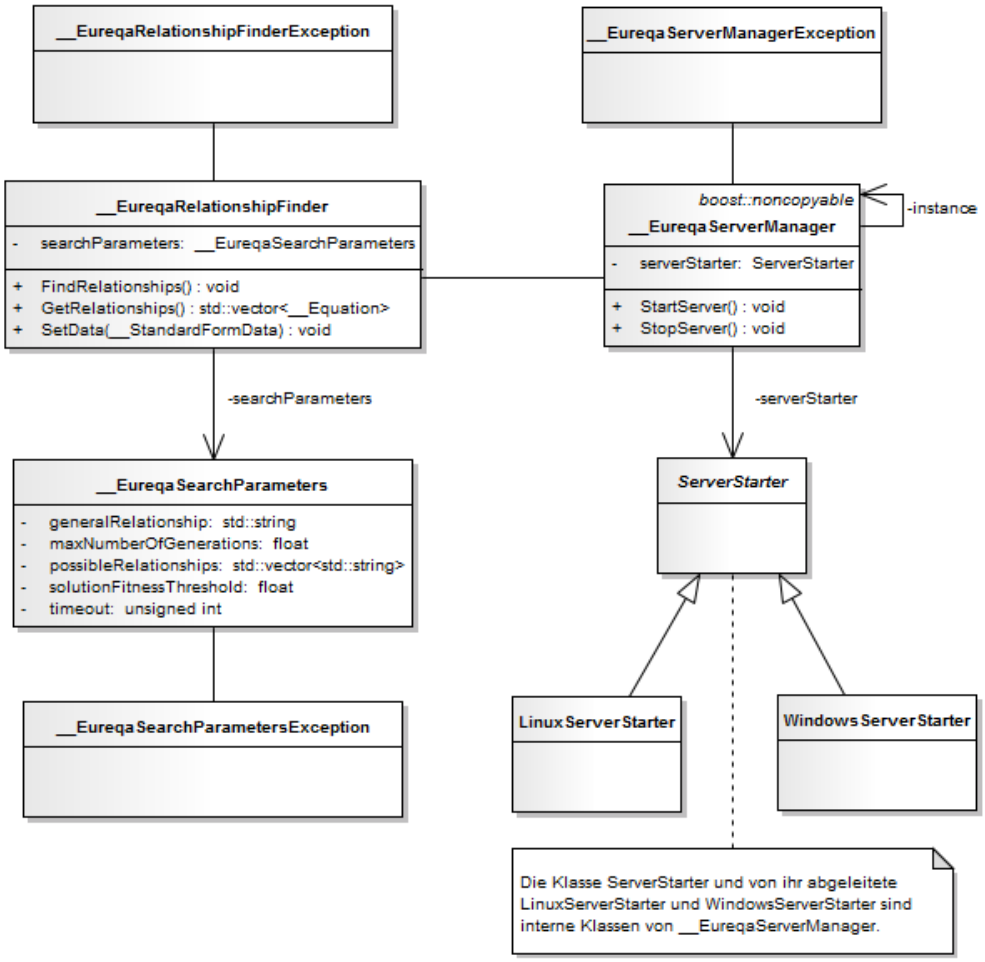

Abbildung 5.2: Vereinfachtes Klassendiagramm für die Komponenten, die das Eureqa API benutzen.

Eureqa

Die Gruppe der Klassen, welche die benötigte Funktionalität des Eureqa API enthalten, besteht aus:

- *__EureqaSearchParameters*;
- *__EureqaRelationshipFinder*;
- *__EureqaServerManager*.

Das Klassendiagramm, in dem die wichtigste Funktionalität der Klassen zu sehen ist, wird mit der Abbildung 5.2 dargestellt. Die Klasse

__*EureqaSearchParameters* erlaubt die Einstellung der Suchparameter, wie beispielsweise:

- maximale Dauer der Suche (*timeout*);
- maximale Anzahl von Generationen (*maxNumberOfGenerations*) – siehe Abschnitt 4.2.1;
- die allgemeine Form der Beziehung (*generalRelationship*), die gefunden werden soll;
- eine Liste mathematischer Funktionen, die während der Suche benutzt werden sollen (*possibleRelationships*).

Die ersten beiden sind Beispiele für Abbruchsparameter und mindestens einer von ihnen muss eingestellt werden.

Die __*EureqaSearchParameters*-Klasse wird später von der Klasse __*EureqaRelationshipFinder* benutzt, um die Parameter der Suche einzustellen. Nachdem die Eingabedaten mit der Funktion *SetData* festgelegt wurden, kann die Suche gestartet werden.

Wie schon im Abschnitt 4.2.1 geschrieben wurde, bietet das Eureqa API die Möglichkeit zur Suche nach Beziehungen mit verschiedenen Servern, mit denen die Applikation kommunizieren kann. Im einfachsten Fall, wenn die Suche nur auf dem lokalen Rechner durchgeführt wird, muss aber auch ein Server auf diesem gestartet werden. Die Situation verkompliziert sich, weil die benutzte Version des Eureqa API[1] keine Funktionalität bietet, die Server-Applikation zu starten. Stattdessen wird eine Binärdatei angeboten (jeweils für Windows und Linux), die gestartet werden muss. Dafür ist die Singleton-Klasse __*EureqaServerManager* verantwortlich. Innerhalb dieser Klasse befindet sich eine Instanz von *ServerStarter*, die abhängig von dem benutzten Betriebssystem als eine der abgeleiteten Klassen *WindowsServerStarter* oder *LinuxServerStarter* initialisiert wird.[2] Da es keine Anforderung des Auftraggebers gab, dass die Software unter der Windows-Umgebung funktionieren soll, sind die Methoden der *WindowsServerStarter*-Klasse zu diesem Zeitpunkt nicht implementiert. Die getroffenen Entwurfsentscheidungen bieten aber eine Möglichkeit, diese Funktionalität sehr einfach zu ergänzen. Für *LinuxServerStarter* wurde die benötigte Funktionalität mit den Linux-

[1] Die von der Software benutzte Version ist 1.02.

[2] Was auch dem liskovschen Substitutionsprinzip (engl. „Liskov substitution principle“) entspricht.

spezifischen Methoden der Interprozesskommunikation erreicht. Die genaue Beschreibung dieser Methoden wäre relativ umfangreich und wurde daher im Rahmen dieser Arbeit nicht durchgeführt. Stark vereinfacht wird die Methode *StartServer* aus der Klasse *__EureqaServerManager* aufgerufen. Diese Methode startet und kontrolliert einen neuen Prozess, der für den Ablauf und die Kommunikation mit der Server-Applikation verantwortlich ist. Nach der Suche wird dieser Prozess beendet. Für eine Referenz über die benutzten Methoden siehe [Roc04].

Implementation der Singleton-Klasse in C++

Die Beschreibung der *__EureqaServerManager*-Klasse eignet sich sowohl für eine kurze Diskussion über die Vor- und Nachteile der Programmiersprache C++ als auch die Wichtigkeit korrekter Entwurfsentscheidungen, wenn diese Sprache von einem Entwickler benutzt wird. Diese Diskussion betrifft das Singleton-Entwurfsmuster und seine möglichen Implementierungen.

Da das Muster ursprünglich von der „Gang of Four“ entwickelt wurde, wird diese Version zuerst analysiert.[3] Auf den ersten Blick sieht der Quellcode (Listing 5.1) recht logisch aus, obwohl man objektiv feststellen muss, dass er für solch ein einfaches Muster relativ komplex ist. Die Probleme stecken aber im Detail. Zuerst wird bemerkt, dass vor dem ersten Aufruf der Funktion *Instance* kein Singleton-Objekt existiert. Noch schlimmer ist, dass es keinen Destruktor gibt, also wird das Objekt nie gelöscht. In diesem Fall ist das kein Speicherleck,[4] aber auf jeden Fall ein Ressourcenleck (engl. „resource leak“), weil der Singleton-Konstruktor viele Ressourcen verbrauchen kann.

Die zweite Möglichkeit, wie man ein Singleton implementieren kann, basiert auf einer globalen Variable (Listing 5.2). Die Lösung kann aber nicht als sicher betrachtet werden, weil die Reihenfolge der Initialisierung Compiler-abhängig ist und es passieren kann, dass man die nicht initialisierte Klasse benutzen würde. Viele andere Probleme, die für diese Arbeit nicht relevant sind, hängen mit Parallelisierung zusammen. Insgesamt ist die optimale Implementierung des Singletons so kompliziert, dass sie ein ganzes Kapitel im Buch [Ale01] verdient hat. Die finale Version des Singletons (siehe Listing 5.3), die alle aufgeführten negativen Merkmale eliminiert, wurde für die Implementierung der *__EureqaServerManager*-Klasse benutzt.

[3] Der Software-Engineering-Klassiker [Gam+95] ist die erste Publikation, in der das Singleton-Entwurfsmuster beschrieben wird.

[4] Ein Speicherleck würde auftreten, wenn der Speicherblock alloziiert wird und alle Referenzen auf diesen Block verloren gingen.

```
// Deklaration
class Singleton
{
    public:
        static Singleton* Instance();
    protected:
        Singleton();
    private:
        static Singleton* instance;
};

// Definition
Singleton* Singleton::instance = NULL;

Singleton* Singleton::Instance()
{
    if(instance == NULL)
        instance = new Singleton();
    return instance;
}
```

Listing 5.1: Von der „Gang of Four“ vorgeschlagene Implementation von Singleton.

```
// Deklaration
class Singleton : private boost::noncopyable
{
    public:
        static Singleton instance;

    private:
        Singleton();
        ~Singleton();
};

// Definition
Singleton Singleton::instance;
```

Listing 5.2: Implementation von Singleton durch Benutzung der globalen Variable.

Eine der Schlussfolgerungen, die man aus dieser Analyse ziehen könnte, wäre, dass „Entwurfsmuster ein Zeichen von Schwäche sind, da die Sprache transparente Mechanismen enthalten soll, sodass man solche Muster gar nicht bräuchte“.[5] Es ist natürlich schwierig so eine Sprache zu bauen, aber auf Grund des Beispiels kann man eigene Konklusionen ziehen, wie schwierig es sein könnte, in der Sprache C++ richtige Entwurfsentscheidungen zu treffen.

```
// Deklaration
class Singleton : private boost::noncopyable
{
    public:
        static Singleton* Instance();

    private:
        Singleton();
        ~Singleton();
        // Statische Initialisierung
        static bool initialized;
        // Dynamische Initialisierung
        static Singleton instance;
};

// Definition
bool Singleton::initialized;
Singleton Singleton::instance;

Singleton::Singleton()
{
    initialized = true;
}

Singleton::~Singleton()
{
    initialized = false;
}

Singleton* Singleton::Instance()
{
    return initialized ? &instance : NULL;
}
```

Listing 5.3: Finale Implementation der Singleton-Klasse.

[5] Nach Meinung des berühmten Perl-Programmierers Mark Dominus. Quelle: `http://blog.plover.com/2006/09/11`

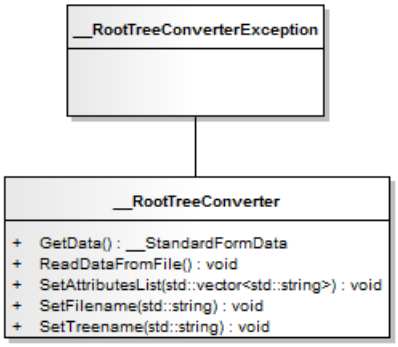

Abbildung 5.3: Darstellung der essenziellen Methoden der Klasse, die für das Parsing der ROOT-Dateien verantwortlich ist.

ROOT-Bibliothek

Wie vorher geschrieben wurde, ist ROOT eine „spezifische“ Bibliothek, in der kaum gute Standards bzgl. der C++-Programmierung zur Anwendung kommen – siehe Abschnitt 4.2.4. Deswegen wurden die Abhängigkeiten zu dieser Bibliothek auf das absolute Minimum beschränkt. Die einzige Klasse, die für das Parsing der Daten aus den ROOT-Dateien verantwortlich ist, benutzt die Methoden der Bibliothek, um die Daten, die sich in Form der serialisierten *TTree*-Struktur befinden, in die Standard-Form zu konvertieren. Das vereinfachte Diagramm für diese Klasse wird mit der Abbildung 5.3 präsentiert.

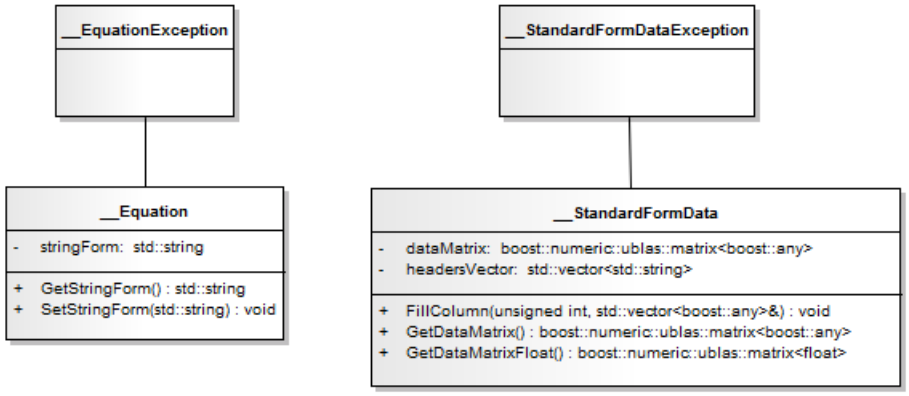

Abbildung 5.4: Vereinfachte Darstellung der benutzten C++-Datenstrukturen.

Datenstrukturen

In C++ wurden zwei Datenstrukturen definiert, die notwendig sind, um die Kommunikation zwischen C++-Komponenten zu erlauben. Die *__StandardFormData*-Klasse ist ein Objekt, das Daten in der Standard-Form speichert und die *__Equation*-Klasse enthält eine mathematische Abhängigkeit, die benutzt wird, um die Relationen zwischen Attributen zu beschreiben. Die C++-Version dieser Klasse speichert nur die Ausgabe von Eureqa, die vom Typ *std::string* ist. Im Vergleich dazu ist die Klasse *__StandardFormData* eine der wichtigsten Strukturen der gesamten Klassenbibliothek. Wie aus Abbildung 5.4 ersichtlich ist, wurden die Daten als *boost::matrix* gespeichert. Da es in C++ keinen Typ wie ein universelles „Object“[6] gibt, ist es kompliziert, eine heterogene Kollektion zu implementieren. Eine der Möglichkeiten wäre, einen *void*-Zeiger (*void**) und dynamische Speicherverwaltung zu benutzen. Diese Lösung ist aber nicht besonders gut, weil der Entwickler keine Möglichkeit hat, den Typ des Objekts, das sich hinter der Adresse eines Zeigers verbirgt, zu erkennen. Eine bessere Möglichkeit wäre, eine Klassenhierarchie aufzubauen und einen RTTI-Mechanismus[7] zu benutzen, um den Typ des Objekts zur Laufzeit zu bestimmen. Die Lösung wäre akzeptabel, aber ein generischer Mechanismus wird bereits von einer der Boost-Bibliotheken implementiert, die eine Lösung in Form von *boost::any*-Objekten, die mit jeder C++-Struktur eingesetzt werden können, anbietet. Diese Objekte werden innerhalb von *boost::matrix* gespeichert.

6 Der Name entstammt dem *Object*-Typ aus C#.

7 RTTI steht für „Run-Time Type Identification“ und ist der in C++ für Reflection verantwortliche Mechanismus.

Ausnahme-Behandlung

Alle Exceptions, die in dem C++-Quellcode benutzt werden, sind von der *__ApplicationException*-Struktur abgeleitet (das Klassendiagram ist in der Abbildung 5.5 dargestellt). Es gibt mehrere Gründe für diese Entwurfsentscheidung, die hier vorgestellt werden. Vor allem bietet diese Vererbung eine Möglichkeit, dass nur ein *catch*-Block geschrieben werden muss, um alle Exceptions, die von den entwickelten Klassen geworfen werden, abzufangen. Dieser Mechanismus wird vor allem bei der Übersetzung der Exceptions aus C++ in Python benutzt (siehe Abschnitt 5.2.2). Wie man auch in der Abbildung 5.5 sieht, sind alle Exceptions von *std::exception* und *boost::exception* abgeleitet. Die Ableitung von *std::exception* bietet eine Möglichkeit, die Funktion *what* zu definieren, die Informationen über den Typ des Exception-Objekts liefert. Die Ableitung von *boost::exception* bietet mehr Funktionalität und eine ausführliche Begründung, warum man überhaupt diesen Mechanismus benutzen soll, kann man in der Dokumentation der Boost-Bibliotheken finden – siehe [DAR10].

Das Prinzip der Exceptions ist in vielen Programmiersprachen gleich. Es gibt einen *throw*-Block, in dem ein Exception-Objekt konstruiert und mit Daten, die hilfreich für die Identifikation des Fehlers sind, gefüllt wird und einen *catch*-Block, in dem ein Objekt aus dem *throw*-Block abgefangen werden soll. Das Problem ist aber sehr oft, dass innerhalb des *throw*-Blocks, in dem Moment, in dem die Exception geworfen wird, nicht genügend Informationen bekannt sind – beispielsweise wenn sie aus einer Funktion, die eine Datei liest, geworfen wird und diese keine Informationen über den Dateinamen, sondern nur einen Zeiger auf das geöffnete Datei-Objekt – *FILE** – hat. Eine der häufigsten Lösungen, die sehr oft von C++-Entwicklern mit Java-Hintergrund verwendet wird, ist das sog. „exception wrapping“.[8] Nach Meinung des Autors ist diese Lösung ein Entwurfsfehler, da es u .a. zu sog. „object slicing“[9] und zum Verlust von Informationen, die für die Identifizierung eines Problems nützlich wären, führen kann. Stattdessen wurde die „elegante“ Lösung aus einer der Boost-Bibliotheken verwendet. Diese bietet u. a. die wichtige Möglichkeit an, dass innerhalb des *throw*-Blocks so viele Informationen angegeben werden können, wie in diesem Moment bekannt und relevant sind. Das Exception-Objekt kann später auf höheren Ebenen mit

[8] Eine Technik, eine abgefangene Exception innerhalb einer neuen Exception zu verpacken und sie danach erneut zu werfen.

[9] Eine Situation wenn eine Objekt-Zuweisung einer Instanz der Eltern-Klasse zur Kind-Klasse erfolgt und den Verlust der zusätzlichen Informationen der Kind-Klasse nach sich zieht.

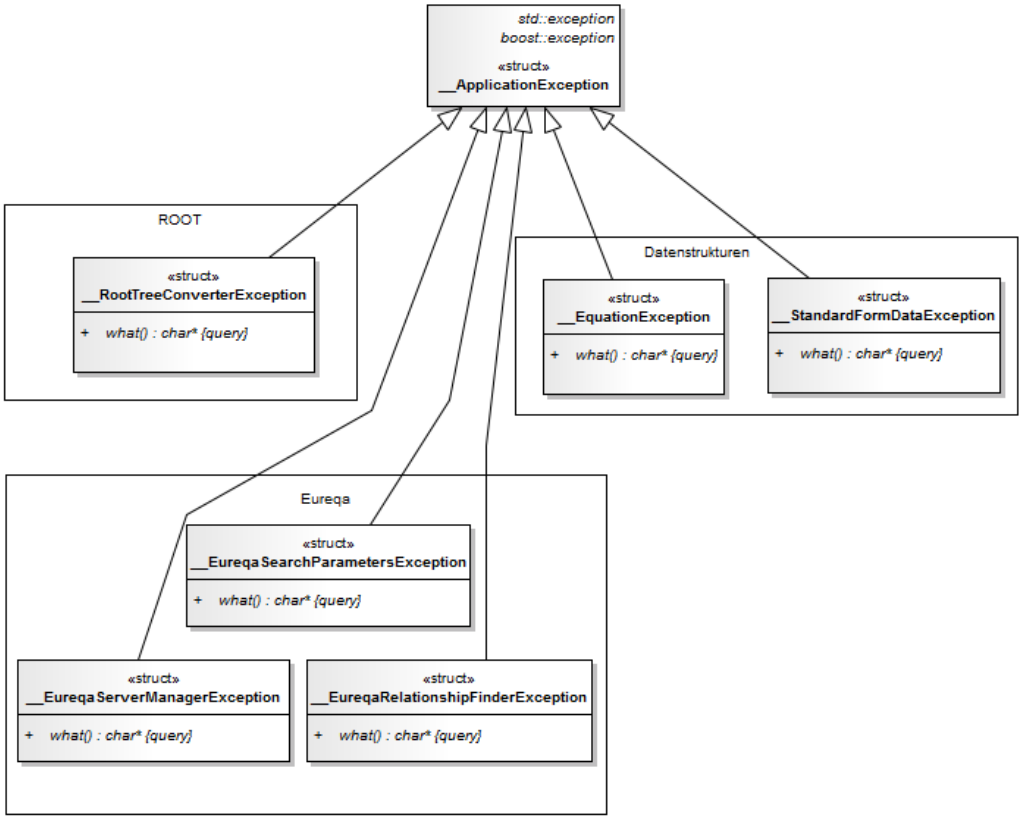

Abbildung 5.5: Hierarchie der C++-Exceptions.

genaueren Informationen ergänzt werden.

5.2.2 Python-Bindings

Bevor die C++-Komponenten aus Python benutzten werden können, müssen ihre Funktionen Argumente akzeptieren, die von Python vestanden werden. Beispielsweise ist es unmöglich, eine Funktion, die als Parameter *std::vector* erwartet, direkt aus Python aufzurufen, da es diesen Typ in Python nicht gibt.

In diesem Moment ist es noch wichtig, einen Bezug zur Orange-Bibliothek herzustellen. Wie vorher geschrieben wurde, bietet diese Bibliothek eine Python-Schnittstelle an, wobei alle Operationen in C++ ausgeführt werden. Die Daten werden aber als Python-Strukturen abgelegt und müssen für jede Operation in C++ übersetzt werden. Die von dem Autor geschriebene Software geht einen Schritt weiter. Nicht nur die Operationen werden in

C++ ausgeführt sondern auch die C++-Objekte werden aus Python implizit benutzt. Man könnte sich jetzt die Frage stellen, warum keine in C++ geschriebene Data-Mining-Bibliothek verwendet und mit den entwickelten Python-Bindings in Python übersetzt wurde. Die Antwort ist, dass es keine solche Bibliothek gibt, die benutzt werde könnte. Die in C++ geschriebene Data-Mining-Bibliothek MLC++ wurde zwar von der berühmten Firma SGI[10] entwickelt aber zuletzt 1997 aktualisiert und nicht weiter unterstützt.

Als Beispiel wird die Python-Klasse *StandardFormData* betrachtet. Die Klasse selbst ist von *_StandardFormData*, die als Python-Binding-Klasse dient, vererbt und diese wiederum besitzt ein Exemplar der reinen C++-Klasse *__StandardFormData*. Diese Beziehungen werden mit der Abbildung 5.6 dargestellt. Die Erklärung für diese Entwurfsentscheidung ist, dass *_StandardFormData* kein Objekt vom Typ *__StandardFormData* ist. Eine andere Möglichkeit der Implementation dieser Beziehung wäre im Fall der Programmiersprache C++ die private Vererbung, die auch eine Art der Aggregation ist. Laut der Quelle [Cli10] sollte Aggregation benutzt werden, wenn es möglich und die private Vererbung nur wenn es notwendig ist. Wenn es um die Beziehung zwischen den Klassen *StandardFormData* und *_StandardFormData* geht, wurde entschieden, sie nach dem Prinzip der Vererbung zu entwickeln, da *StandardFormData* die Funktionalität der Klasse *_StandardFormData* erweitert.

Bezüglich der Klassen aus der „StandardForm“-Familie, ist auch wichtig zu bemerken, dass es Unterschiede zwischen den Schnittstellen, die sie anbieten, gibt. Da Python eine dynamisch-typisierte Sprache ist und die Daten in der C++-Klasse als eine heterogene Matrix gespeichert sind, muss sich die Python-Binding-Klasse *_StandardFormData* explizit um die Konsistenz der Spalten, die entweder numerische oder nominale Elemente enthalten können, kümmern.

Ausnahme-Übersetzung

Außer den Objekten müssen auch die mit ihnen verbundenen Exceptions in Python übersetzt werden. Wie schon mit der Abbildung 5.5 präsentiert wurde, werden alle Exceptions von *__ApplicationException* abgeleitet, was die saubere und elegante Durchführung dieser Prozedur ermöglicht. Diese Exception wird mit Hilfe der Boost-Python-Bibliothek in Pythons *RuntimeError*-Exception übersetzt. Eine weitere Gruppe von Exceptions

[10] `http://www.sgi.com`

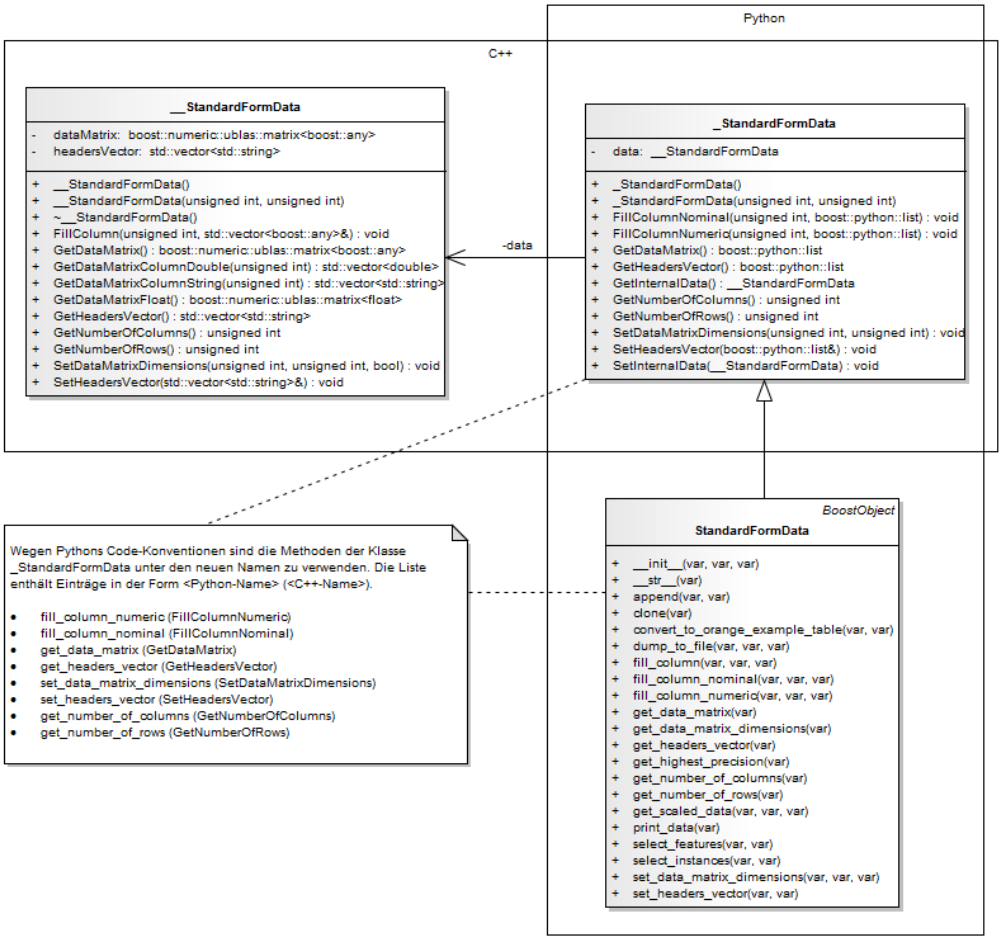

Abbildung 5.6: Beispiel einer Relation zwischen Python und C++.

muss innerhalb von Python behandelt werden – diejenigen, welche die Boost-Python-Bibliothek selbst generiert. Aus der Abbildung 5.6 ist ersichtlich, dass *StandardFormData* von *BoostObject* abgeleitet ist. Diese Klasse implementiert Mechanismen, die für die korrekte Behandlung aller beschriebenen Exceptions verantwortlich ist.

5.2.3 Python-Komponenten

In Bezug auf die entwickelte Software ist die Gruppe der Python-Komponenten die größte, deswegen werden nur die wichtigsten Elemente dieser Gruppe beschrieben. Ihre Haupt-Datenstrukturen sind:

- *StandardFormData*

- *Equation*
- *DMSolution*

Wie bereits zuvor erwähnt sind die ersten beiden Komponenten mit den C++-Strukturen *__StandardFormData* und *__Equation* verbunden. Die letzte ist eine reine Python-Datenstruktur, die als Behälter für das Data-Mining-Modell von Orange dient. All diese Strukturen kann man einfach serialisieren und deserialisieren (dafür ist u. a. das Standard-Python-Modul *pickle* verantwortlich). Die ersten beiden Strukturen lassen sich auch in menschenlesbarer Form speichern.

Interpretation mathematischer Formeln

Eine der Anforderungen an die Software war die Möglichkeit, die neue Lösung mit der bisherigen mathematischen Formel zu vergleichen (falls vorher solch eine Formel entwickelt wurde). Um das zu erreichen, implementiert die Klasse *Equation* eine Funktionalität, die eine mathematische Formel für eine bestimmte Menge an Variablen auswerten kann.

Ein naives Verfahren, solche Funktionalität zu implementieren, wäre der Versuch, einen Parser zu schreiben, der auf regulären Ausdrücken basiert. Doch dies ist ausgeschlossen, da reguläre Ausdrücke es beispielsweise unmöglich machen, Klammerpaare zu finden[11] und ohne deren Beachtung können mathematische Formeln nicht korrekt evaluiert werden.

Ein anderes Verfahren wäre, eine Grammatik, die mathematische Formeln interpretiert, zu definieren und dafür einen passenden Compiler zu entwickeln. Diese Lösung wäre die einzig mögliche, wenn man nur eine Sprache benutzt, die in erster Linie kompiliert werden muss. Da Python eine interpretierte Sprache ist und es möglich macht, ihren eigenen Interpreter innerhalb eines Skripts aufzurufen, wird die Eingabeformel mit diesem Mechanismus evaluiert. Vorher wird sie semantisch überprüft und nach den Regeln der Python-Syntax konvertiert. Listing 5.4 zeigt ein einfaches Beispiel dieser Konvertierung (*MathExtension* ist eines der Module der geschriebenen

[11] Der Beweis dafür ist, dass reguläre Ausdrücke nicht mächtig genug sind, um eine komplexe Syntax zu beschreiben. Ein regulärer Ausdruck R beschreibt eine Menge von Zeichenketten, die als Sprache $L(R)$ bezeichnet werden. Sie ist die Sprache eines deterministischen endlichen Automaten, der die Eingabe auf Grund des Zustands, in dem er sich befindet, entweder akzeptiert oder nicht. Für eine undefinierte Anzahl von Klammern ist es unmöglich, die Menge der erlaubten Zeichenketten zu definieren, also die Sprache $L(R)$ zu formulieren. Für eine weitere Referenz siehe [Aho+06].

```
# Eingabe
cos(x) + gauss(y) + 3.14

# interpretierbare Version
math.cos(x) + MathExtension.gauss(y) + 3.14
```

Listing 5.4: Beispiel der Konvertierung einer Formel in eine von Python interpretierbare Form.

Software, das Funktionen definiert, die weder im Standard-Mathematikpaket *math* noch in *NumPy* enthalten sind – beispielsweise die *gauss*-Funktion, die als Teil der von dem Eureqa API gelieferten Lösung vorkommen kann). In Verbindung mit dem Thema der mathematischen Lösungen, die von dem Eureqa API geliefert werden, muss auch das Problem der Division durch 0 betrachtet werden. Es passiert nämlich oft, dass innerhalb einer Testmenge, mit der die Performance der Lösung beurteilt werden soll, Werte existieren, für die eine Division durch 0 durchgeführt werden muss. In der genetischen Programmierung wird für die Berechnung der Fitnessfunktion vorausgesetzt, dass das Ergebnis dieser Operation als 1 betrachtet werden soll. Das gleiche Verfahren wurde in der Software implementiert.

Sicherheitsmechanismen

Wie schon vorher oft erwähnt wurde, kann jeder Benutzer der Software auf Basis der entwickelten Klassenbibliothek eigene Skripte verfassen. Obwohl die Klassendokumentation alle notwendigen Informationen dafür liefert, wurden zwei zusätzliche Mechanismen implementiert, um eventuelle Probleme möglichst schnell zu lokalisieren und zu eliminieren. Der erste von ihnen ist Logging. Jeder Funktionsaufruf wird in der Log-Datei gespeichert, sodass eine Sequenz, die problematisch war, später analysiert werden könnte. Der zweite Mechanismus, der als Unterstützung des ersten dient, ist die konstante Überwachung der Operationen, die von bestimmten Objekten ausgeführt werden mit Ausgabe einer sofortigen Warnmeldung, wenn eine Operation durchgeführt werden soll, die zu einem Zeitpunkt nicht erlaubt ist. Diese Funktionalität wurde durch Darstellung der Abfolge der Aktivitäten, die eine Klasse durchführen kann, in Form eines endlichen Automaten erreicht. Die Klasse *RootTreeConverter* beispielsweise liest ROOT-Dateien und behält wegen der Zeit-Komplexität der Leseoperation die gelesenen Daten im Arbeitsspeicher. Auf die Daten kann später mit der Funktion *get_data* erneut zugegriffen werden. Der Aufruf der Funktion kann sofort nach der Objekterstellung passieren aber als potenzieller Fehler betrachtet werden, da es schlichtweg nichts

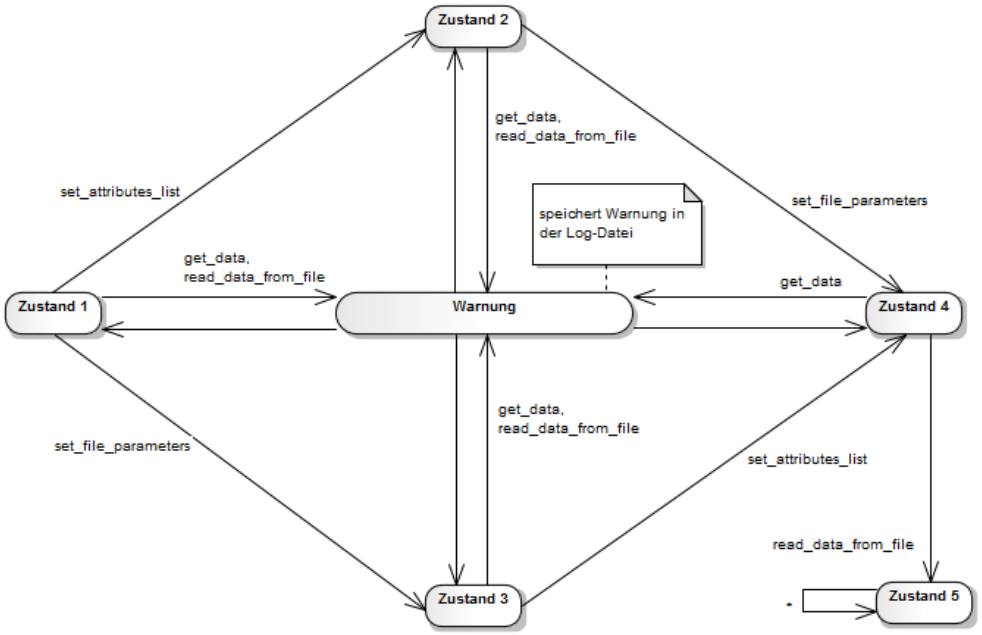

Abbildung 5.7: Endlicher Automat, der die Abfolge der Aktivitäten für die Klasse *RootTreeConverter* definiert.

zu Lesen gibt. In diesem Fall produziert der Monitor-Mechanismus eine Warnung, die in der Log-Datei gespeichert wird. Ein endlicher Automat, der dafür benutzt wurde, ist in der Abbildung 5.7 dargestellt.

Die Klasse, die diese Funktionalität anbietet, wurde mit dem Entwurfsmuster „Zustand“ (engl. „State“) implementiert. Die Idee, die Funktionsaufrufe zu speichern, basiert teilweise auf dem Entwurfsmuster „Kommando“ (engl. „Command“). Für eine weitere Referenz siehe [Gam+95] und [Sav97].

Singleton in Python

Im Abschnitt 5.2.1 wurde eine Implementierung des Singleton-Entwurfsmusters in der Programmiersprache C++ besprochen. Die Implementierung in Python ist einfacher als die in C++, aber es gibt trotzdem mindestens zwei verschiedene Möglichkeiten sie durchzuführen. Die „klassische“ basiert auf der, die von der „Gang of Four“ vorgeschlagen und in [Sav97] beschrieben wurde. Die Alternative ist unter dem Namen „Borg“ bekannt. „Borg“ simuliert das Prinzip des Singleton-Entwurfsmusters durch Erstellung einer Menge von Objekten, die den gleichen Zustand haben. Es ist nicht möglich zu entscheiden, welche der beiden Lösungen allgemein besser ist. Die Imple-

mentierung von „Borg“ (Listing 5.5) erfordert weniger Quellcode und wurde deswegen u. a. für die Entwicklung der Klasse, die für Logging verantwortlich ist, gewählt.

```
class Borg:
    __shared_state = {}
    def __init__(self):
        self.__dict__ = self.__shared_state
```

Listing 5.5: Implementierung von „Borg“ in Python.

5.3 Kurzes Resümee

Wie vorher geschrieben wurde, wäre es widersinnig die implementierte Software detailliert zu beschreiben, da dies die Kapazität der Arbeit übersteigen würde (selbst ein kleiner Teil eines vollständigen Klassendiagramms wäre schwierig auf einer A4-Seite abzubilden). Deswegen wird die Diskussion der Software-Entwicklung damit abgeschlossen. Für weitere Referenzen sei auf den Quellcode, der Anhang der elektronischen Version dieser Arbeit ist, verwiesen.

Kapitel 6

Anwendungsmöglichkeiten und Ergebnisse

In diesem Kapitel werden die zwei Möglichkeiten der Anwendung der entwickelten Software und Klassenbibliothek präsentiert. Außerdem werden die Ergebnisse einer der zurzeit benutzten Formeln mit den Data-Mining-Lösungen, die von der Software ermittelt wurden, verglichen. Die Diskussion wird mit einer kurzen Darstellung der möglichen Anwendungen der entwickelten Software eröffnet.

6.1 Anwendungsmöglichkeiten

Die einfachste Methode, die Software zu nutzen, ist der Aufruf der in Python geschriebenen Hauptapplikation. Sie bietet die Möglichkeit, Daten aus einer ROOT-Datei zu laden, Instanzen zu wählen und die Kreuzvalidierung für die SVM-Methode oder die Methode der genetischen Programmierung durchzuführen. Das vereinfachte Sequenzdiagramm wird mit Abbildung 6.1 präsentiert.

Die zweite Möglichkeit ist die Klassenbibliothek zu benutzen, um eigene Skripte zu schreiben. Diese Methode bietet selbstverständlich mehr Freiheit und da Python eine sehr intuitive Sprache ist, kann sie von jedem Forscher mit minimaler Erfahrung in der Software-Entwicklung benutzt werden. Man könnte in diesem Moment eigentlich unendlich viele Beispiele solcher Skripte anführen. Eines davon wird gewählt und hier dargestellt. Es zeigt die parametrisierte Anwendung der Methoden der genetischen Programmierung. Die Daten werden aus einer ROOT-Datei geladen, Instanzen werden gewählt und

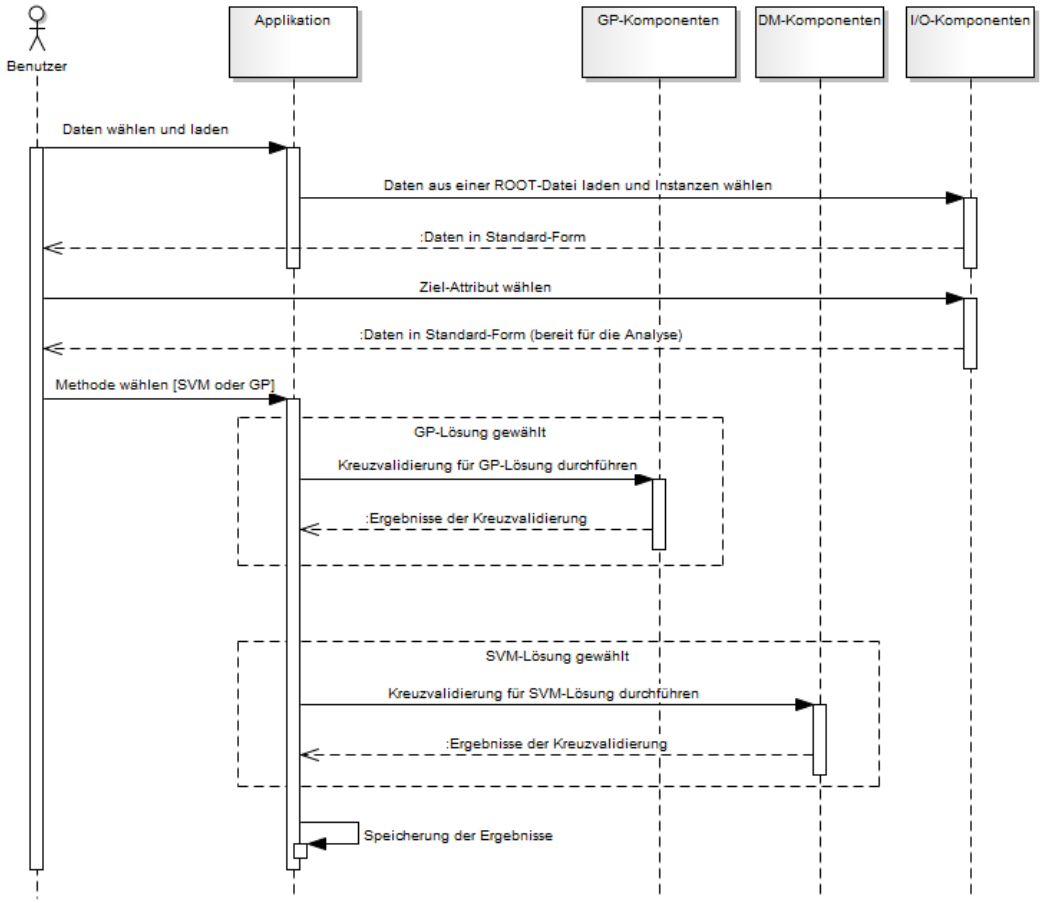

Abbildung 6.1: Vereinfachtes Sequenzdiagramm des Ablaufs der Hauptapplikation.

die Suchparameter werden so eingestellt, dass die Suche, entweder nach einer Stunde oder wenn die Fitness der Lösung weniger als 0,1 beträgt,[1] gestoppt wird. Dieses Beispiel ist im Listing 6.1 zu sehen.[2]

6.2 Ergebnisse

Die Software wurde zu diesem Zeitpunkt u. a. benutzt, um die Performance von Data-Mining-Lösung und bisher im Experiment verwendeter Formel mit-

[1] Die Fitness der Lösung ist in diesem Fall ein Wert, der eine Kombination aus den Ergebnissen der Fitness-Funktion (siehe Abschnitt 3.3) und der Komplexität der Lösung ist. Üblicherweise wird eine Fitness-Funktion benutzt, welche die mittlere absolute Abweichung minimiert.

[2] Die sonst nötigen *import*-Anweisungen wurden absichtlich entfernt.

```
# Konverter fuer ROOT-Dateien
converter = RootTreeConverter()
converter.set_file_parameters('file.root', 'T')
att_list = ['CCheight_EW', 'geomag_angle', 'Ze',
            'latMeanDistCC_EW', 'lgE']
converter.set_attributes_list(att_list)
converter.read_data_from_file()

# Daten in Standard-Form
data = converter.get_data()
data = data.select_instances('CCheight_EW < 1e-04 and lgE > 8')

# Suchparameter
search_parameters = EureqaSearchParameters()
search_parameters.set_timeout(1, 'h')
search_parameters.set_solution_fitness_threshold(0.1)
dependent_att = att_list
dependent_att.remove('lgE')
search_parameters.set_general_relationship(dependent_att, 'lgE')

# derzeitige Loesung
current_solution = Equation()
string_form = 'log10(pow((CCheight_EW * pow(10, 6) / 11
              * (1.16 - cos(geomag_angle)) * cos(Ze)
              * exp((-latMeanDistCC_EW / 236))), 1/0.95)) + 8'
current_solution.set_string_form(string_form)

# Kreuzvalidierung der GP-Loesung
gp_tester = EureqaTester()
gp_tester.test_performance(search_parameters, data, 10,
                           current_solution)
```

Listing 6.1: Beispiel der Anwendung der implementierten Python-Klassenbibliothek.

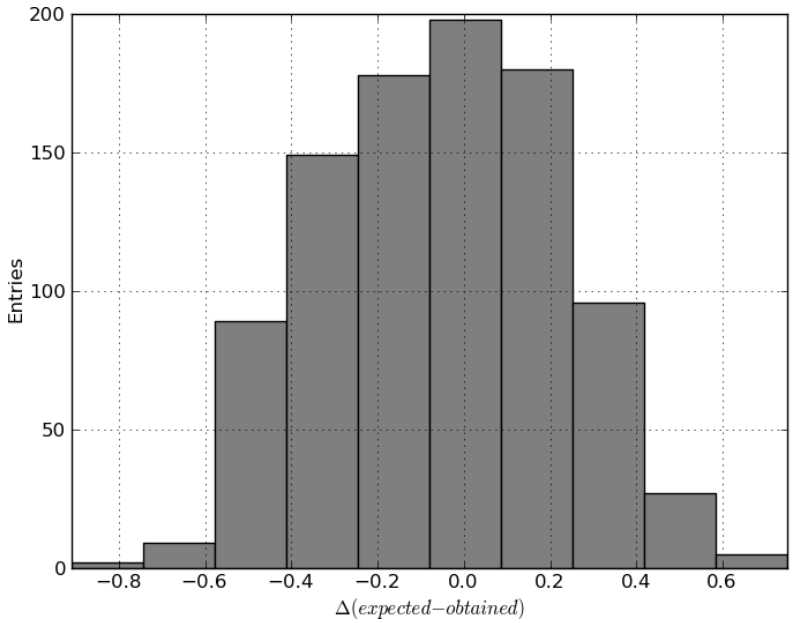

Abbildung 6.2: Histogramm des Fehlers für die derzeit verwendete Formel.

einander zu vergleichen. Die dargestellten Ergebnisse basieren auf der Formel zur Berechnung der Radiopulshöhe auf Grund der geschätzten Primärenergie, die mit der Gleichung 6.1 präsentiert und in der Quelle [Hor+07] genauer beschrieben wird.

$$\epsilon_{est} = 11 \cdot (1.16 - cos(\alpha)) \cdot cos(\theta) \cdot e^{\frac{-R_{SA}}{236}} \cdot \frac{E_p}{10^{17}} \quad \left[\frac{\mu\ V}{m\ MHz}\right] \tag{6.1}$$

Der Parameter α ist der geomagnetische Winkel, θ der Zenitwinkel und R_{SA} der durchschnittliche Abstand der Antennen zur Schauerachse. Diese Formel wurde umgestellt, um die Primärenergie zu berechnen. Das Ergebnis wurde mit dem Referenzwert aus dem KASCADE-Experiment verglichen.[3] Das Histogramm des Fehlers wird mit der Abbildung 6.2 präsentiert. Mit den selben Daten wurde eine Kreuzvalidierung für jeweils SVM und die Methode der genetischen Programmierung durchgeführt. Die Resultate sind in den Abbildungen 6.3 und 6.4 dargestellt. Die Abbildungen 6.2 und 6.4 wurden mit dem Quellcode aus Listing 6.1 produziert. Bzgl. des genannten Listings korrespondieren die Parameter der Gleichung mit den folgenden Attributen aus der erwähnten ROOT-Datei:

[3] Es wurden ca. 1.000 Instanzen benutzt.

- ϵ_{est} – $CCheight\_EW$
- α – $geomag\_angle$
- θ – Ze
- R_{SA} – $latMeanDistCC\_EW$
- E_p – 10^{lgE}

Es ist dem Autor nicht möglich, die Benutzbarkeit der Methode nur allein anhand ihrer Performance zu beurteilen. Sie muss stattdessen von den Forschern während der Alltagsarbeit bewertet werden. Wenn es um die gewählten Data-Mining-Lösungen selbst geht, sind die Ergebnisse, die mit der Software ermittelt werden auf Grund der Performance eindeutig nicht schlechter als die Ergebnisse, die mit der bisherigen Methode berechnet wurden.

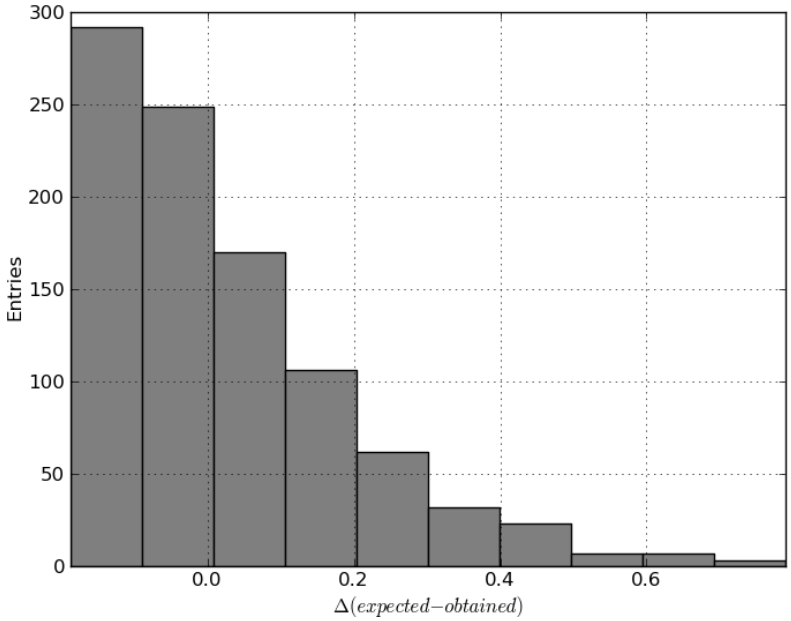

Abbildung 6.3: Histogramm des Fehlers der SVM-Lösung.

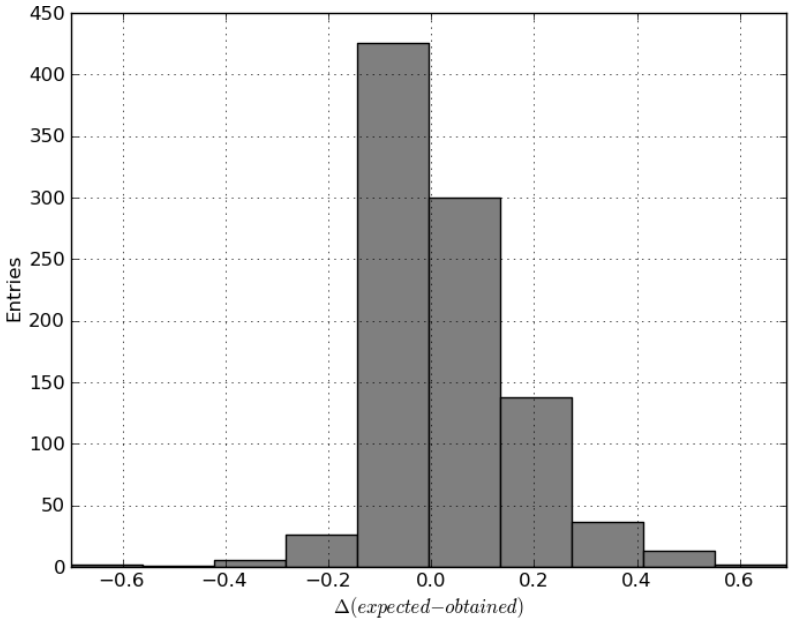

Abbildung 6.4: Histogramm des Fehlers der GP-Lösung.

Kapitel 7

Zusammenfassung

Das Ziel der Arbeit war, die Methoden des Data Mining in das LOPES-Experiment zu integrieren. Zuerst wurden verschiedene Data-Mining-Methoden mit den Daten aus LOPES getestet und dann einige brauchbare ausgewählt. Die entwickelte Software erfüllt dieses Ziel und die Python-Klassenbibliothek, die dafür geschrieben wurde, kann einfach für die Erstellung eigener Skripte benutzt werden. Die Hauptapplikation kann man benutzen, ohne weitere Kenntnisse weder von Programmierung noch von Data Mining zu besitzen, was als eine attraktive Alternative für das bisherige Analyse-Verfahren betrachtet werden kann.

Das Thema der Nutzung von Data-Mining-Methoden im Bereich der Physik ist zwar nicht neu, wurde aber vom Autor dahingehend neutral betrachtet, dass die Vorurteile von Quellen, die meist neuronale Netze als ein universelles Mittel für mathematische oder physikalische Probleme ansehen, außen vor gelassen wurden. Auch soll die Leserschaft nicht davon überzeugt werden,[1] dass mit der Methode der Entscheidungsbäume einfach die Datenmengen, die aus vielen Attributen und tausenden Instanzen bestehen, verständlich beschrieben werden können.[2]

Alle an die Software gestellten Anforderungen konnten erfüllt werden. Deswegen hofft der Autor, dass die Software von den Forschern des KIT häufig benutzt wird und dass sie zumindest implizit den Fortschritt im Bereich der Astrophysik vorantreibt. Auf jeden Fall war es eine Herausforderung diese

[1] Die Arbeit will keine negativen Beispiele anpreisen, deswegen wurden sie an dieser Stelle nicht aufgelistet.

[2] Die Methode der Entscheidungsbäume ist in der Software verfügbar und dient während der Datenvorbereitungsphase als ein Tool für die Attributreduktion.

Software zu planen, zu entwickeln und die damit verbundenen technischen Probleme zu lösen. Die Kooperation mit dem KIT war erfolgreich und der Autor hat sein Wissen in den Bereichen sowohl der Astrophysik als auch des Data Mining und der Software-Entwicklung vertiefen können.

Literaturverzeichnis

[Aho+06] A. V. Aho u. a. *Compiler. Principles, Techniques and Tools, Second Edition*. Addison-Wesley, 2006.

[Ale01] A. Alexandrescu. *Modern C++ Design. Generic Programming and Design Patterns Applied*. Addison-Wesley, 2001.

[Ale04] A. Alexandrescu. *C++ Coding Standards. 101 Rules, Guidelines, and Best Practices*. Addison-Wesley, 2004.

[BL04] M. J. A. Berry und G. S. Linoff. *Data mining techniques. For marketing, Sales and Customer Relationship Management, Second Edition*. Wiley, 2004.

[BL99] M. J. A. Berry und G. S. Linoff. *Mastering Data Mining. The Art and Science of Customer Relationship Management*. Wiley, 1999.

[Cli10] M. Cline. *C++ FAQ Lite*. 2010. URL: `http://www.parashift.com/c++-faq-lite/`.

[DAR10] B. Daves, D. Abrahms und R. Rivers. *Boost Library Documentation*. 2010. URL: `http://www.boost.org/doc/libs/`.

[Fal+05] H. Falcke u. a. „Detection and imaging of atmospheric radio flashes from cosmic ray air showers“. In: *Nature* 435 (2005). Nature Publishing Group, Seiten 313–316.

[Fow+99] M. Fowler u. a. *Refactoring. Improving the Design of Existing Code*. Addison-Wesley, 1999.

[Fra10] M. Franc. „Data Mining in industriellen Prozessen“. Studienarbeit (nicht öffentlich verfügbar). Fachhochschule Brandenburg, 2010.

[Gam+95] E. Gamma u. a. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[HL02] M. Howard und D. LeBlanc. *Writing Secure Code. Second Edition*. Microsoft Press, 2002.

[Hof08] T. Hoff. *C++ Coding Standard*. 2008. URL: `http://www.possibility.com/Cpp/CppCodingStandard.html`.

[Hor06] A. Horneffer. „Measuring Radio Emission from Cosmic Ray Air Showers with a Digital Radio Telescope“. Dissertation. Rheinische Friedrich-Wilhelms-Universität Bonn, 2006.

[Hor+07] *Primary Particle Energy Calibration of the EAS Radio Pulse Height*. 30th ICRC. Merida, Mexiko, 2007.

[Ise98] P. Isensee. *C++ Optimization Strategies and Techniques*. 1998. URL: `http://www.tantalon.com/pete/cppopt/main.htm`.

[Jel+65] J. V. Jelley u. a. „Radio Pulses from Extensive Cosmic-Ray Air Showers“. In: *Nature* 205 (1965). Nature Publishing Group, Seiten 327–328.

[JM01] H. Jiawei und K. Micheline. *Data Mining. Concepts and Techniques*. Academic Press, 2001.

[KL03] S. S. Keerthi und C.-J. Lin. „Asymptotic behaviors of support vector machines with Gaussian kernel“. In: *Neural Computation* 15 (2003). MIT Press, Seiten 1667–1689.

[Koz07] J. R. Koza. *The home page of Genetic Programming Inc.* 2007. URL: `http://www.genetic-programming.com`.

[McC04] S. McConnell. *Code Complete. Second Edition*. Microsoft Press, 2004.

[Mey98] S. Meyers. *Effective C++. Second Edition*. Addison-Wesley, 1998.

[Ous98] J. K. Ousterhout. „Scripting: Higher-Level Programming for the 21st Century“. In: *Computer* 31 (1998). IEEE Computer Society, Seiten 22–30.

[Roc04] M. J. Rochkind. *Advanced UNIX Programming. Second Edition*. Addison-Wesley, 2004.

[RW09] G. van Rossum und B. Warsaw. *Style Guide for Python Code*. 2009. URL: `http://www.python.org/dev/peps/pep-0008`.

[Sar97] W. S. Sarle. *Neural Network FAQ*. 1997. URL: `ftp://ftp.sas.com/pub/neural/FAQ.html`.

[Sav97] V. Savikko. *Design Patterns in Python*. 1997. URL: http://www.python.org/workshops/1997-10/proceedings/savikko.html.

[SC03] International Organization for Standardization und International Electrotechnical Commission. *ISO/IEC 14882:2003. Programming languages – C++*. 2003. URL: http://www.iso.org/iso/catalogue_detail.htm?csnumber=38110.

[Sch10] B. Schäling. *The Boost C++ Libraries*. 2010. URL: http://en.highscore.de/cpp/boost.

[SL09] M. Schmidt und H. Lipson. „Distilling Free-Form Natural Laws from Experimental Data“. In: *Science* 324 (2009). American Association for the Advancement of Science (AAAS), Seiten 81–85.

[Str01] B. Stroustrup. *The C++ Programming Language. Third Edition*. Addison-Wesley, 2001.

[Str09] B. Stroustrup. *Bjarne Stroustrup's C++ Style and Technique FAQ*. 2009. URL: http://www2.research.att.com/~bs/bs_faq2.html.

[Vap95] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[Wei02] K. Weicker. *Evolutionäre Algorithmen*. Teubner, 2002.

[Wei08] M. Weigend. *Python GE-PACKT. Schneller Zugriff auf Module, Klassen und Funktionen. Tkinter, Datenbanken und Internet-Programmierung. Für die Versionen Python 3.0 und 2.x*. Mitp-Verlag, 2008.

[WF05] I. H. Witten und E. Frank. *Data Mining. Practical Machine Learning Tools and Techniques, Second Edition*. Elsevier, 2005.

[WI98] S. M. Weiss und N. Indurkhya. *Data Mining. A practical guide*. Morgan Kaufmann Publishers, 1998.

[Zel04] I. Zelinka. *Symbolic regression - an overview*. 2004. URL: http://www.mafy.lut.fi/EcmiNL/older/ecmi35/node70.html.

Abbildungsverzeichnis

Danksagung

Für meine Masterarbeit schulde ich vielen Menschen einen herzlichen Dank. Danken möchte ich vor allem Herrn Prof. Dr. Jochen Heinsohn und Herrn Prof. Dr. Johannes Blümer, den Gutachtern dieser Arbeit. Für die jederzeit vorhandene Hilfsbereitschaft und wissenschaftliche Unterstützung bei allen auftauchenden Fragen möchte ich mich auch bei Herrn Ingo Boersch, Herrn Dr. Andreas Haungs und Herrn Frank Schröder bedanken.

Selbstständigkeitserklärung

Ich erkläre hiermit, dass die vorliegende Arbeit von mir selbst und ohne fremde Hilfe verfasst wurde. Alle benutzten Quellen sind im Literaturverzeichnis angegeben. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Marcin Franc, 18. August 2010