Vergleich der Machine Learning API Anbieter BigML, Google, Prediction.io und Weka

Martin Macheiner

BACHELORARBEIT

Nr. 1310237022-A

eingereicht am
Fachhochschul-Bachelorstudiengang

Mobile Computing

in Hagenberg

im Juni 2016

Diese Arbeit entstand im Rahmen des Gegenstands

Mobile Sports

im

Sommersemester 2016

Betreuer:

Stephan Selinger, FH-Prof. DI.

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 13. Juni 2016

Martin Macheiner

Inhaltsverzeichnis

Kurzfassung

Die Welt ist digital und jeder Mensch hinterlässt einen digitalen Fußabdruck. Ist es kein Geheimnis, dass die Big Player und da allen voran Google ihr Geld mit der Auswertung von Nutzerdaten lukriert. Jeder Benutzer hinterlässt bei jeglicher Interaktion mit Apps oder Webanwendungen auswertbare Daten. Mithilfe von Machine Learning ist es möglich, die gespeicherten Daten in Zusammenhang zu setzen. Voraussetzung ist nur ein möglichst großes Datenset für den jeweiligen Anwendungsbereich. Zu diesen Anwendungsbereichen zählen unter anderem: Email-Spam-Erkennung, in der Sportinformatik oder das Diagnostizieren von Erkrankungen.

Die Arbeit beschäftigt sich zum einen mit den Grundlagen von Machine Learning und andererseits mit der Implementierung von mehreren Machine Learning Lösungen. Die Grundlagen umfassen die Algorithmen, die verschiedenen Lernansätze, Big Data, sowie eine Aufzählung der bereits bestehenden API Anbieter. Bei der Implementierung werden verfügbare Machine Learning API Anbieter miteinander aufgrund von verschiedenen Kriterien verglichen. Im Zuge der Evaluierung wird eine Java-Applikation entwickelt, um die verschiedenen Anbieter anhand von Genauigkeit und Laufzeiten miteinander zu vergleichen. Die API Anbieter lassen sich über ein modulares Plugin-System in die Applikation einbinden.

Abstract

We live in a digital world and every person leaves a digital footprint. It's no secret, that the big players and there especially Google are making their money by analyzing the user's data. Every user generates valuable data in the interaction with mobile or web apps. With the help of machine learning it's possible to wire up all the already stored data and predict or classify new data. The only prerequisite is a preferably big data set for the particular use case. Use cases could be: email spam detection, sport informatics or the diagnosis of diseases.

The work includes the fundamentals of machine learning and also the implementation of several machine learning solutions. The basics chapter contains the applied algorithms, the learning conceptions, big data and a listing of already available API provider. The implementation compares different machine learning API provider based on multiple criteria. As part of the evaluation there will be implementation of a Java application, with which it's possible to compare the different provider by means of accuracy and runtime. The API provider can be easily integrated into the application by a modular plugin system.

Kapitel 1

Einleitung

1.1 Motivation

Durch die tagtägliche Benutzung von Smartphones und Computern hinterlassen wir einen digitalen Fußabdruck quer durch das Internet. Google und Co. gewinnen aus diesen Daten wertvolle Informationen über die Benutzer und – in Googles Fall – erstellen damit maßgeschneiderte Werbung. Deswegen ist das Sammeln von personenbezogenen Nutzerdaten in Verruf gekommen. Dabei wird aber meist der Nutzen, den eine solche Datenanalyse haben kann, außer Acht gelassen. *Machine Learning* und *Data Prediction* finden unter anderem in der Medizin Anwendung. Natürlich handelt es sich dabei um sensible Daten, aber die Auswertung dieser Daten könnte helfen, Krankheiten besser vorherzusagen und zudem würde es die Möglichkeit bieten den Arzt bei der Diagnostizierung zu unterstützen [24]. Hier wird ein echter Mehrwert für den Menschen generiert. Eben das, was Technologie in der heutigen Zeit leisten soll. Die Medizin ist aber nur ein Anwendungsgebiet, in welchem Machine Learning verwendet werden kann. Amazon als Beispiel – das ebenfalls ein API für ihre Machine Learning Lösung anbieten – floriert so prächtig, da die Vorschläge für den Kunden interessante Produkte über Machine Learning generiert werden. Ein weiterer Blick auf den mobilen Bereich offenbart, dass zurzeit die persönlichen Assistenten die wichtigsten Begleiter am Smartphone sind: Siri, Google Now oder Cortana. Kontextbasierte Suche löst mehr und mehr die Stichwortsuche ab. Das wäre alles ohne Machine Learning nicht möglich [13]. Außerdem schlagen diese Apps auch Beiträge vor, die dem Benutzer gefallen könnten, im Grunde eben Data Prediction. Die Apps funktionieren deswegen meist sehr exakt, weil Google, Apple und Microsoft einen entsprechend großen Datenpool besitzen, um ihre Algorithmen damit zu trainieren.

1.2 Problemstellung

Viele Entwickler und Firmen können von Machine Learning profitieren. Im mobilen Kontext kann Machine Learning etwa zur Erkennung von bösartigen Apps in den App Stores verwendet werden [16]. Dafür wird aber eine Infrastruktur und ein Algorithmus benötigt. Ein solches System aufzusetzen kostet Zeit, Ressourcen und Geld. Außerdem kann jedes Modell ausschließlich für einen einzigen Anwendungsbereich verwendet werden. Hier kann die Überlegung angestellt werden, ob die Implementierung direkt am Gerät stattfindet (was vor allem bei mobilen Geräten zulasten der Akkulaufzeit geht), oder online über einen Server. Meistens rentiert es sich nicht selbst einen Server zu hosten. Einige namhafte Firmen bieten Machine-Learning-Systeme für Kunden an. Unter diesen Firmen befinden sich unter anderem Google, Amazon und Microsoft. Es gibt aber auch Firmen, deren Geschäft das Anbieten von cloud-basierter Machine-Learning-Infrastruktur ist. Bei diesen Firmen kann man davon ausgehen, dass sie auch durchgehend verfügbar sind. Zum Beispiel bietet Google eine Verfügbarkeit von 99.9%[1].

Das Problem dabei ist oft wie man diese Anbieter nun in den Workflow der Applikation einbindet. Durch die Anzahl der Anbieter kann der Entwickler den für seinen Anwendungsfall besten wählen. Die Zielsetzung der praxisbezogenen Kapiteln 5 - 7 ist es, auf die Einbindung des API Anbieter in eine bestehende Applikation einzugehen. Zusätzlich werden noch weitere Faktoren berücksichtigt um zu sehen, welcher API Anbieter für den jeweiligen Anwendungsfall am besten geeignet ist.

1.3 Gliederung

Zu Beginn der Arbeit werden die Begriffe *Machine Learning* und *Data Prediction* erläutert und die Grundlagen des Themas beschrieben. Dabei werden die Anwendungsgebiete und Vorzüge aufgelistet, aber auch die Probleme, die mit *Machine Learning* einhergehen. Mit Fortlauf der Arbeit werden der Stand der Forschung und der Begriff *Big Data* erörtert. In den Kapiteln 3 und 4 werden Algorithmen und die von den Anbietern bereitgestellten APIs beschrieben. Nach der Einführung in die Theorie wird das eigene Konzept und der eigene Lösungsansatz vorgestellt. Des Weiteren wird im Zuge des Konzepts auch die Implementierung und Umsetzung im Detail betrachtet. Am Ende wird die Evaluierung der erschlossenen Kenntnisse durchgeführt, welche die gewonnenen Daten des Konzepts miteinander vergleicht.

[1] https://cloud.google.com/prediction/pricing#paid-usage

Kapitel 2

Machine Learning / Data Prediction

Beim Fachausdruck *Data Prediction* handelt es sich um eine Datenvorhersage aufgrund bereits vorliegender Daten. Er wird oft synonym mit dem Begriff *Machine Learning* verwendet. Die Maschine kann angelernt werden, um Muster und Zusammenhänge in diesen sogenannten Lern- bzw. Trainingsdaten zu erkennen. Ist nun eine Maschine trainiert, so kann er neue Daten selbständig klassifizieren oder zuordnen.

In diesem Kapitel geht es darum, die Begriffe Data Prediction, Prediction Model, Big Data, Data Mining und Machine Learning zu trennen und Bewusstsein für die Anwendungsgebiete von den genannten Begriffen zu schaffen. Außerdem werden die benötigten Lerndaten genauer betrachtet. Immerhin sind die Lerndaten notwendig, um das Prediction Model zu erstellen. Dabei ist vor allem wichtig, wie genau die Lerndaten sind und wie viele Attribute sie besitzen (Die Anzahl der Attribute variiert nach Anwendungsfall). In diesem Zusammenhang werden auch die Probleme von *Machine Learning* beschrieben, die zwangsläufig damit einhergehen.

2.1 Begriffsklärung

Wenn es um Daten und um ihre Verarbeitung geht, dann fällt gerne der Begriff der Datenanalyse(QUELLE). Auch der Begriff Big Data wird – vor allem in Verbindung mit Firmen wie Google, Facebook oder Amazon – immer wieder verwendet. Meist sind diese Begriffe nur teilweise richtig und werden fälschlicherweise synonym für Begriffe wie Data Prediction, Data Mining und Machine Learning verwendet. Aber was verbirgt sich hinter diesen Begriffen?

Der Begriff *Data Prediction* bezieht sich auf die Vorhersage von Ereignissen, welche aufgrund von ausgewerteten Daten ermittelt werden [20, S. 2]. Ein-

fach ausgedrückt nutzt Data Prediction als Grundlage ein Datenset (nicht zwingend Big Data) und verwendet Machine Learning um Ergebnisse vorherzusagen. Um eine Datenvorhersage zu treffen werden folgende Punkte benötigt:

- Datenset an Lerndaten
- Machine Learning Algorithmus
- Prediction Model

Die Lerndaten sollte möglichst umfangreich und statistisch gut verteilt sein, um das Problem von Overfitting zu vermeiden (sieh 4.2). Hierfür bietet sich gewissermaßen Big Data an: Eine möglichst große, unabhängige Datenmenge für einen speziellen Anwendungsbereich. Als Beispiel hierfür könnte man alle Einkäufe aller Kunden von Amazon anführen.

Es gibt mehrere Machine-Learning-Algorithmen, die je nach Anwendungsfall im Bezug auf Vor-und Nachteile verwendet werden können. Auf diese wird in Kapitel 3.1 näher eingegangen. Der Machine Learning Algorithmus erzeugt und trainiert ein sogenanntes Prediction Model [9, S. 17]. Nach diesem Training is es möglich, Anfragen an das Prediction Model zu stellen und dieses antwortet mit der plausibelsten Vorhersage (Prediction). Das Prediction Model ist ein integraler Bestandteil von Data Prediction. Prediction Models können auch als PMML-Dateien exportiert und so in anderen Anwendungen verwendet werden.

Bei *Data Mining* handelt es sich um die Analyse von Datenbeständen. Data Mining ist ein Prozess, bei dem Muster in Daten erkannt werden [8, S. 5]. Der Prozess geschieht meist automatisiert, oder zumindest semi-automatisiert. Mittels Data Mining können beispielsweise die Lerndaten für Data Prediction extrahiert werden. Dabei sieht man klar, dass diese Begriffe oft ineinandergreifen und auch meist dieselbe Materie behandeln.

Laut Witten [8, S. 5] geht man davon aus, dass sich alle 20 Monate die Menge an gespeicherten Daten verdoppelt. Somit liegen unzählige Datensätze vor, welche an und für sich keinen Mehrwert besitzen. Aber im Verbund kann man in ihnen Muster erkennen und in diesen Fällen kann Data Mining angewendet werden.

Spricht man von *Big Data*, spricht man von Datensätzen in der Größenordnung mehrerer Terabyte [18, S. 2]. Trotzdem wird der Bezeichnung öfters falsch ausgelegt, wodurch Big Data sehr schnell – aufgrund von Privacy-Bedenken – negativ behaftet ist. Hinter Big Data stehen im Grunde zwei sich in der Informatik immer wiederkehrende Gedanken: Datenspeicherung und Datenanalyse [10, S. 1]. Die genaue Spezifikation von Big Data ist dahingehend schwer, weil das Wort *Big* Interpretationsspielraum bietet. Geht es dabei nun um Wichtigkeit, Komplexität oder nur um Quantität? [10, S. 1] Wie schon erwähnt gibt es mehrere Definitionen dafür, einige kritische

Faktoren treffen jedoch immer auf Big Data zu:

- Größe
- Komplexität
- Technologie

Die Größe beschreibt die Anzahl der Datensätze, welche bei Big Data gewissermaßen umfangreich ist [10, S. 1]. Wie komplex Big Data ist, wird vor allem durch die Struktur und das Verhalten der Datensätze beschrieben. Zudem zeichnet sich der Begriff auch durch die verwendete Technologie aus. Immerhin haben sich einige Softwareprodukte auf die Verarbeitung von großen Datenmengen spezialisiert, wie zum Beispiel NoSQL [10, S. 2]. Um die sich ändernden Eigenschaften von Big Data zu beschreiben werden oftmals die 4Vs verwendet.

- Volume
- Variety
- Velocity
- Veracity

Auf deutsch: Volumen, Vielfalt, Geschwindigkeit und Richtigkeit [5, S. 5]. Die 4Vs besagen, dass Big Data immer weiter an Volumen gewinnt, die gewonnenen Daten immer vielfältiger – und damit schwerer zu interpretieren – werden, die Geschwindigkeit für die Erzeugung neuer Daten zunimmt und die Daten noch auf Richtigkeit überprüft werden müssen, um die Datenanalyse nicht zu verfälschen.

Der Begriff *Machine Learning* bedeutet, dass ein Algorithmus Muster aus Daten erkennt und daraus Prediction Models erzeugen kann [9, S. 3], mit welchem es möglich ist Vorhersagen zu treffen. Alles nur in einem gewissen Grad versteht sich.

Machine Learning kann aus den gegeben Trainingsdaten selbst – meist sehr gute – Trainingsergebnisse erzielen und ein solides Prediction Model erstellen. Die Aufgabenbereiche dafür sind vielseitig. Man nehme an, es geht um ein Kreditinstitut. Dem Kreditinstitut liegen die Daten des Kreditnehmers vor, wie etwa das Einkommen. Dabei könnte man den Entschluss fassen, wenn das Einkommen kleiner als 1500€ ist, wird der Kredit abgelehnt. Hierfür werden keine weiteren Faktoren miteinbezogen, obwohl Faktoren wie Alter, Ersparnisse und Grundbesitz eine Rolle spielen könnten. Die Aufgabenstellung ist nicht mehr trivial, wenn weitaus mehr Daten miteinbezogen werden. Hier spielen Machine-Learning Algorithmen ihre Stärke aus.

Diese Algorithmen fußen auf mathematischen Lernmodellen, je nach Lernansatz. Sehr verbreitet ist der Ansatz von Entscheidungsbäumen [9, S. 121], welche den Ansatz von informationsbasiertem Lernen wählen. Dabei gibt es vier populäre Lernansätze:

- Informationsbasiertes Lernen

- Ähnlichkeitsbasiertes Lernen
- Wahrscheinlichkeitsbasiertes Lernen
- Fehlerbasiertes Lernen

Neben den Lernansätzen gibt es auch noch verschiedene Lerntypen. Für Data Prediction wird fast ausschließlich *Supervised Learning* verwendet [9, S. 3]. In manchen Fällen wird auch *Unsupervised Learning* verwendet [22, S. 6]. Daneben gibt es noch – für die Vollständigkeit – *Semi-supervised Learning* und *Reinforcement Learning*. Supervised Learning bedeutet, dass aufgrund von beschreibbaren Attributen und einem Zielattribut eine Entscheidung getroffen wird [19, S. 4], welche dem System in Form von Trainingsdaten hinzugeführt wird.

2.2 Anwendungsgebiete

Zunächst stellt sich die Frage, ob die Integration von Data Prediction in kleinen Projekten überhaupt sinnvoll erscheint, oder ob nur große Firmen mit einem dementsprechenden Datenpool davon profitieren. Es gibt einige Anwendungsgebiete, die auch für App-Entwickler interessant sein können, deswegen gibt es auch eine Vielzahl an API Anbieter [29]. Es beziehen sich sehr viele Anwendungsbereiche [9, S. 1] auf Enterprise-Produkte. Es müssen in jedem Fall Ressourcen vorhanden sein, um Daten zu erhalten und verarbeiten zu können.

Ein Anwendungsgebiet ist *Price Prediction*. Es bezieht sich auf die Preiskalkulation je nach Nachfrage und auf weitere Einflussfaktoren. Unternehmen, wie Fluglinien, Tankstellenbetreiber und Online-Händler passen ihre Preise den gegebenen Umständen an. Erst jüngst gab Amazon bekannt, dass sich die Preise mehrmals pro Tag ändern können, je nach Kundeninteresse [32]. Auch Risikoanalyse fällt in dieses Anwendungsgebiet.

Auch in der Medizin ist Machine Learning stark vertreten, wobei es sich hier um sehr interessante Anwendungsgebiete für den Menschen handelt. Es kann für die Reduktion der Risiken bei der Medikamentenverabreichung nützlich sein. Es kann aber auch eine Hilfestellung für raschere Entwicklungszyklen von neuen Medikamenten sein.

Zum anderen wird Machine Learning bei Diagnosen angewendet, wobei der Algorithmus auf eine Vielzahl von bereits diagnostizierten Krankheiten und die dazugehörigen Eigenschaften zurückgreifen kann (Näheres in Kapitel 5.3. Diese Methode ist üblicherweise in bereits bestehende Diagnoseprozesse eingebunden [9, S. 2].

Ein spannender Anwendungsbereich trägt den Namen *Suspicious activity reporting*. Dabei geht es darum, Aktivitäten aufzudecken, welche nicht vom

eigentlichen Inhaber getätigt worden sind. Kurz gesagt, wenn verdächtige Aktivitäten erkannt werden, die in der Regel auf Missbrauch hindeuten. Hier geht es vor allem um Banküberweisungen oder Kreditkartenabrechnungen [17]. Jedoch verwenden immer mehr Software-Anbieter ebenfalls einen solchen Algorithmus, um zu erkennen, ob der Login von dem dazugehörigen Benutzer ausgeführt wurde, oder ob sich ein Fremder Zugriff zu einem Konto verschaffen wollte[2].

Bei *Recommendation Systems* handelt es sich um einen Empfehlungsdienst. Solche Systeme werden tagtäglich benutzt [11]. In App Stores werden nach dem Herunterladen einer App Vorschläge für andere Apps angezeigt. Amazon wendet dasselbe Modell an seinen Produkten an, was im besten Fall zu einer Umsatzsteigerung führt, da der Konsument mehrere ähnliche Produkte vergleichen und kaufen kann.

Ein Großteil der Mail-Anbieter stellt einen *Spam-Filter* zur Verfügung. Im Grunde ist ein Spam-Filter nichts anderes, als ein Prediction Model, welches entscheidet, ob eine Mail als Spam klassifiziert werden soll oder nicht. Ein einfacher Spam-Filter müsste nicht einmal Machine Learning verwenden, sondern könnte aufgrund von Mail-Adressen und Wörtern auf einer Blacklist jene als Spam klassifizieren. Ausgereiftere Spam-Filter verwenden Machine Learning. Mail-Anbieter wie Yahoo, Google oder Microsoft können wieder auf eine große Anzahl von Mails zurückgreifen. Aus dieser Anzahl von Mails kann den Maschinen angelernt werden, welche Mails als Spam zu klassifizieren sind und welche nicht.

Die *Klassifizierung von Dokumenten* hat eine starke Ähnlichkeit mit dem Filtern von Spam-Mails. Der Ansatz ist derselbe. Zum Beispiel Gmail kategorisiert Mails in drei Kategorien: Primary, Social und Promotions. Bei Promotions handelt es sich vor allem um Werbung und Newsletters. Aber nicht nur Mails, sondern auch Dokumente können je nach Inhalt klassifiziert und zugeordnet werden [9, S. 2].

2.3 Notwendige Daten

Jeder Mensch benötigt Übung um etwas zu erlernen. Übung macht den Meister. Je besser der Trainer, desto besser die Lernfortschritte des Schülers. Je mehr man lernt, desto besser wird man dabei. Und so verhält es sich auch im Bereich von Machine Learning. Eine Maschine kann nur so intelligent werden, wie es der Trainer – der Mensch – zulässt. Damit ein Prediction Model erstellt werden kann, muss dem Machine Learning Algorithmus zuerst Intelligenz antrainiert werden. Das passiert mit den sogenannten Trainingsdaten.

[2] U.a. Facebook und Google verwenden Suspicious Activity Reporting

Dabei handelt es sich um vollständige Datensets, die alle Attribute (Auslassungen sind möglich) sowie die zu bestimmende Klasse beinhalten. In der Tabelle 2.1 ist ein Ausschnitt aus einem Trainingsbeispiel des Google Prediction APIs[3] abgebildet. Es beinhaltet ein Attribut und die Klasse. Die Sprache, welche später auch erkannt werden soll, und einen Textausschnitt aus dieser Sprache. Die Maschine kann den Textausschnitt nun eindeutig der Sprache zuordnen, weil es die Trainingsdaten so beschreiben. In der Regel haben Trainingsdatensets zwischen 100 und mehreren Millionen Einträgen [33]. Die große Anzahl ist notwendig, damit statistische Fehler und falsches Anlernen ausgeschlossen werden können.

English	This version of the simple language detection
French	M. de Troisvilles, comme s'appelait encore sa
Spanish	En efeto, rematado ya su juicio, vino a dar en
French	L'hôte, qui n'était pas doué d'une grande

Tabelle 2.1: Beispiel Trainingsdaten für Sprachtraining.

Aufgrund der oben angeführten Daten kann der Algorithmus jetzt ein Prediction Model anlernen. Die Korrektheit dieser Lerndaten ist essentiell. Würden die Daten nicht stimmen, also die Sprache nicht zu den Textausschnitten passen, so würde das Prediction Model keine vernünftigen Vorhersagen liefern können. Aus diesem Grund sind qualitativ hochwertige Trainingsdaten zwingend notwendig.

Die Güte von Datensets hängt auch von ihrer Anzahl an Attributen oder Features ab, wobei mehr Attribute nicht gleichbedeutend mit einem besseren Datenset ist. Unterschieden wird zwischen *descriptive features* und *target features* [9, S. 5]. Target features sind Zielattribute, auch Klassen genannt. Descriptive features sind beschreibende Attribute, die den Zustand eines Systems abbilden. Ein Prediction Model kann aufgrund der gegebenen beschreibenden Attribute das passende Zielattribut auswählen. Im Beispiel 2.1 gibt es sowohl nur ein Zielattribut als auch nur ein beschreibendes Attribut. Für viele Anwendungsgebiete ist das aber nicht ausreichend. Um beispielsweise ein Herzsignal abzubilden werden bis zu 75 Attribute benötigt[4].

In der Tabelle 2.2 befindet sich ein Auszug von Trainingsdaten für eine balancierte Waage[5]. Das erste Attribut ist das Zielattribut. Die Waage kann entweder balanciert (B), rechtshängend (R) oder linkshängend (L) sein. Das zweite und das vierte Attribut beschreiben das Gewicht auf der linken bzw. rechten Seite. Die Attribute drei und fünf stellen den Abstand des jeweiligen

[3] https://cloud.google.com/prediction/docs/language_id.txt

[4] https://archive.ics.uci.edu/ml/datasets/Heart+Disease

[5] https://archive.ics.uci.edu/ml/machine-learning-databases/balance-scale/balance-scale.data

Arms dar und die Attribute haben einen Wertebereich von 1-5.

R	1	1	5	4
R	1	1	5	5
L	1	2	1	1
B	1	2	1	2
R	1	2	1	3

Tabelle 2.2: Beispiel Trainingsdaten für ausbalancierte Waage.

Der Aufbau der Trainingsdaten im Beispiel ist sehr einfach. Die zusätzliche Meta-Information muss jedoch in einer anderen Datei beschrieben werden. Das Weka Toolkit ist eine Software, die speziell für die Bereiche Data Mining, Data Prediction und Machine Learning ausgelegt ist [34]. Dabei handelt es sich um ein frei verfügbares Tool, das auf der Universität von Waikato entwickelt wurde. Weka verwendet ein eigenes Dateiformat für Trainingsdaten: ARFF [8, S. 53]. Das Format erlaubt die Deklaration von Attributen, sowie deren Wertebereiche und die dazugehörigen Datensätze. Der Aufbau ist dabei äußerst übersichtlich gehalten. Die Datei beginnt mit der Annotation *@relation* und dem dazugehörigen Namen. Attribute werden mit der Annotation *@attribute* gekennzeichnet, gefolgt vom Attributnamen und dessen Wertebereich. Falls es sich um einen numerischen Wertebereich handelt, kann das mit dem Wort *numeric* festgelegt werden. Die Attribute sind sortiert, d.h. die n-te Attributsbezeichnung bezieht sich auf die n-te Spalte des Datensets. Danach folgen die Trainingsdaten der Tabelle 2.2, welche mit der Annotation *@data* eingeleitet werden. Jedoch sind diese mit einem Beistrich getrennt. Das ARFF-Format für die balancierte Waage wird in Listing 2.1 abgebildet. ARFF-Dateien schaffen so einen eleganten und einfachen Weg, die Attribute direkt in der Datei mit Lerndaten zu beschreiben. Durch diese Beschreibung ist es auch möglich, zu überprüfen, ob die vorliegenden Daten überhaupt valide sind.

```
%Attribute-Relation file format (ARFF) for relation scale
@relation scale

@attribute class_balance { B, L, R }
@attribute left_weight { 1, 2, 3, 4, 5 }
@attribute left_distance { 1, 2, 3, 4, 5 }
@attribute right_weight { 1, 2, 3, 4, 5 }
@attribute right_distance { 1, 2, 3, 4, 5 }

@data
R, 1, 1, 5, 4
R, 1, 1, 5, 5
L, 1, 2, 1, 1
B, 1, 2, 1, 2
R, 1, 2, 1, 3
```

Listing 2.1: ARFF-Datei für ausbalancierte Waage.

2.4 Probleme

Eine genaue Datenvorhersage muss sich mit zwei Problemstellungen auseinandersetzen. Es müssen korrekte, qualitativ hochwertige Trainingsdaten vorliegen. Die Attribute spielen eine ebenso wichtige Rolle. Zu wenige Attribute können das Ergebnis ebenso verfälschen, wie zu viele Attribute. Die Anzahl hängt vom jeweiligen Anwendungsgebiet ab. Ein gleichverteiltes Datenfeld wäre optimal. Durch die Gleichverteilung kann die Vorhersage nicht von Ausreißern in der Verteilung manipuliert werden.

Auf der anderen Seite gibt es das Prediction Model, das falsche Ergebnisse hervorrufen kann. Es gibt mehrere Algorithmen für Machine Learning. Jedoch gibt es keinen Algorithmus, der für jedes Anwendungsgebiet vernünftige Ergebnisse liefert. Die Schwierigkeit besteht darin, den richtigen Algorithmus für das jeweilige Anwendungsgebiet zu finden. Das ist Aufgabe des Datenanalysten. Durch ein falsches Prediction Model kann es zu zwei Arten von Komplikationen kommen. Das *Underfitting*: Das Prediction Model ist zu einfach aufgebaut, um die Zusammenhänge zwischen den beschreibenden Attributen und dem Zielattribut darzustellen. Wobei das häufigere Problem *Overfitting* ist: Das Prediction Model ist so kompliziert, dass es zu genau am Datenset liegt. Daraus resultiert, dass keine genaue Vorhersage möglich ist. Schon ein Rauschen in den Daten [9, S. 11] kann eine Änderung der Vorhersage herbeiführen. Overfitting ist ein komplexes Thema und wird in Kapitel 4.2 genauer beschrieben.

Kapitel 3

Stand der Forschung

Erst kürzlich gab Google den Source Code der Software *TensorFlow* frei [36]. Bei *TensorFlow* handelt es sich um Googles Artificial Intelligence Engine. Sie findet Verwendung in der Google Photos App, bei der Sprachübersetzung oder hilft bei der Verbesserung der Suchresultate. Chris Nicholson (Gründer des Startup Skymind) meint, dass Google der Konkurrenz mit TensorFlow fünf bis sieben Jahre voraus sei [36]. Google selbst glaubt, dass durch die Offenlegung des Codes der gesamte Machine Learning Prozess weiter beschleunigt werden könnte, denn jeder hätte nun Zugriff auf eine ausgereifte Software. Aber dadurch erkennt man welch wichtige Rolle Machine Learning auch in der Zukunft spielen wird. Dabei gibt es bereits viele etablierte Algorithmen und Herangehensweisen für das Problem. Auch für das Problem wie eine Maschine lernt gibt es mehrere Herangehensweisen. Es gibt aber keinen idealen Algorithmus der für jede Problemstellung die passenden Ergebnisse liefert. Hier muss vom Datenanalysten abgewogen werden, welcher Algorithmus verwendet wird. Außerdem gibt es zurzeit eine Vielzahl an Anbietern, die ihre Dienste für Machine Learning über das Internet bereitstellen. Einige Anbieter legen die verwendeten Algorithmen transparent offen. Bei solchen Systemen spricht man von einer Whitebox. Andere Anbieter lassen nicht durchblicken, welcher Algorithmus nun wirklich verwendet wird. Diese Systeme werden auch Blackbox genannt.

3.1 Bestehende Techniken und Algorithmen

In diesem Kapitel wird auf die bereits existierenden Algorithmen eingegangen. Es werden auch die verschiedenen Lernansätze beschrieben. Einige API Anbieter verwenden für die Klassifizierung Entscheidungsbäume [27]. Deswegen werden Entscheidungsbäume in diesem Kapitel ein wenig genauer betrachtet. Es gilt auch zwischen kategorischen und und numerischen Attributen zu unterscheiden. Zum Beispiel das kategorische Attribut Motorentyp kann vom Typ Benzin, Diesel oder Hybrid sein. Im englischen wird es *cate-*

gorical feature genannt. Die numerischen Attribute *continuous features*. In den folgenden Erklärungen werden die zwei Begriffe öfter verwendet. Nicht jeder Algorithmus kann standardmäßig mit beiden Arten von Attributen umgehen.

3.1.1 Entscheidungsbäume

Entscheidungsbäume oder Decision Trees gehören zum Konzept des informationsbasierten Lernens. Der Grundgedanke von Entscheidungsbäumen ist, die beschreibenden Attribute des Datensets auf ihren Informationsgehalt zu überprüfen und diese in einer Baumstruktur abzufragen. Als Beispiel dient ein Pokerkartenset. Es wird eine Karte gezogen und man muss erraten um welche Karte es sich handelt. Ein völlig ineffizienter Ansatz wäre von jede Karte jeder Farbe von der Karte zwei bis zum Ass durchzugehen. Oder man fällt die Entscheidung mithilfe eines Entscheidungsbaumes. Jede Karte hat mehrere beschreibende Attribute, wie etwa die Farbe, ob sie eine Bildkarte ist, und einen konkreten Wert, entweder zwischen zwei und zehn, oder Bube, Dame, König und Ass. Hier muss gegeben sein, dass man alle Zustände (Klasse) der Karten kennt. Die erste Frage könnte lauten ob es sich um eine Bildkarte handelt oder nicht. Danach kann die Farbe der Karte ermittelt werden und zum Schluss ihr konkreter Wert. Bei Entscheidungsbäumen ist die Reihenfolge der gestellten Fragen essentiell. Um die Reihenfolge der Entscheidungen zu bestimmen wird der Informationsgehalt der beschreibenden Attribute herangezogen.

Es gibt insgesamt 52 Karten bei vier Farben zu je 13 Karten. Davon sind neun Zahlenkarten und vier Bildkarten. Die Information ob es sich um eine Bildkarte handelt oder nicht, schränkt die Suche von 52 Karten auf nur mehr 16 Karten ein. Ist es jedoch keine Bildkarte, so muss man immerhin noch 36 Karten durchsuchen. Die Information ob es sich bei der gesuchten Karte um eine Herzkarte handelt, schränkt die Suche auf 13 Karten ein. Trifft das aber nicht zu, so müssen 39 Karten durschucht werden.

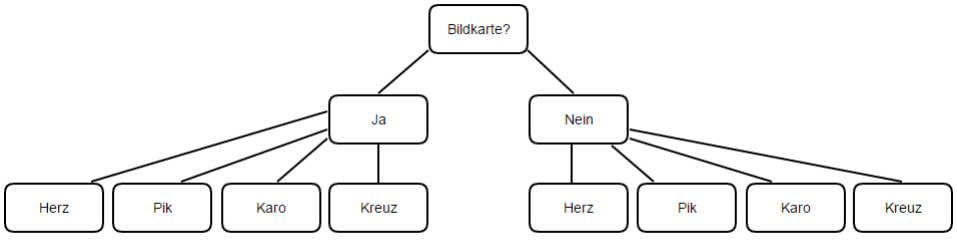


Abbildung 3.1: Beispiel Entscheidungsbaum für Pokerkarten.

Wenn die erste Frage wie in Abbildung 3.1 die Frage nach einer Bildkarte

ist, dann bleiben bei einem Nichtzutreffen der Frage noch immer 36 Karten übrig. Falls es sich aber um eine Bildkarte handelt kann aufgrund der Farbe weiter entschieden werden. Trifft die Farbe auch zu, so gibt es wieder vier Entscheidungsmöglichkeiten um welche Bildkarte es sich genau handelt.

Dabei wird ersichtlich, dass Entscheidungsbäume immer zu einem Ergebnis kommen. Die Reihenfolge der Entscheidungen beeinflusst lediglich die Laufzeit der Vorherssage. Ein Entscheidungsbaum startet mit dem sogenannten Wurzelknoten[6]. Daneben gibt es innere Knoten und Blätter. Jeder Knoten trifft seine Entscheidungen aufgrund eines beschreibenden Attributs. Beschreibende Attribute können in mehreren Knoten vorkommen. Die Blätter beschreiben die Klasse, die der Datensatz mit den beschreibenden Attributen wahrscheinlich besitzt [9, S. 117-122].

Des Weiteren gibt es die Begriffe *Tree Pruning* und *Tree Ensemble*. Bei Tree Pruning handelt es sich um eine Beschneidung des Entscheidungsbaums. Ein zu komplexer Entscheidungsbaum kann zu falschen Vorhersagen führen und kann durch Pruning beschnitten werden. Den Begriff *Occam's Rasiermesser* findet man oft in der Medizin, aber er ist im Grunde in allen Anwendungsbereichen gültig. Im übertragenen Sinne meint man damit, *dass die einfachste Lösung meistens die plausibelste Lösung ist* [23, S. 167]. Bei der Datenverarbeitung bezieht sich das auf das zugrundeliegende Modell. Das einfachste Modell, dass für die Daten passt, ist auch das beste. Ein zu kompliziertes Modell führt oft zu Overfitting4.2. Beim Beschneiden gibt es zwei Herangehensweisen. *Pre-pruning*beschneidet den Baum schon während der Erzeugung. Das ist laufzeittechnisch effizient, da keine Subbäume gebildet werden müssen, die womöglich später beschnitten werden. Beim Erzeugen des Elternknoten kann aber noch nicht vorhergesagt werden, wie effizient und gut die Kinderknoten sein werden. Deswegen können diese womöglich weggeschnitten werden, obwohl die Kinderknoten sehr gute Entscheidungen treffen würden. *Post-pruning* beschneidet dazu im Gegensatz erst, nachdem der Entscheidungsbaum schon komplett erzeugt wurde. Der Nachteil bei dieser Variante ist, dass der Baum komplett erstellt werden muss und erzeugte, aber nicht benötigte Subbäume wieder verworfen werden. Dafür treten aber die vorher erwähnten Fehler nicht auf [9, S. 158-162]. Es liegt auf der Hand, dass ein beschnittener Baum nicht mehr ident mit dem Baum ist, der aufgrund der Trainingsdaten erstellt wurde. Das ist aber dahingehend kein Problem, da der Baum effizient mit den zu prädiktierenden Daten arbeiten soll und nicht mit den Trainingsdaten.

Von *Ensembles* spricht man, wenn mehrere Entscheidungsbäume zusammen eine Vorhersage treffen. Dafür gilt, dass jeder Entscheidungsbaum unabhän-

[6]Entscheidungsbäume ähneln vom Aufbau her der korrespondierende Baum-Datenstruktur

gig von den anderen Bäumen arbeiten muss, damit sie sich nicht gegenseitig beeinflussen [9, S. 163]. Der Vorteil von Ensembles ist, dass sie hervorragende Vorhersagen treffen können, selbst wenn die einzelnen Entscheidungsbäume unpräzise Vorherssagen treffen. Alle Entscheidungsbäume im Ensemble werden mit dem gleichen Datenset erstellt, jedoch wird jeder Baum mit einem modifizierten Datenset trainiert. Für das Erstellen von Ensembles gibt es zwei Herangehensweisen: *Boosting und Bagging*. Boosting sollte für Datensets mit bis zu 4000 beschreibenden Attributen verwendet werden. Bei mehr als 4000 beschreibenden Attributen sollte Bagging verwendet werden, da Boosting leichter zu Overfitting neigt [9, S. 166].

Der *ID3-Algorithmus* gilt als der Standardalgorithmus für das erzeugen von Entscheidungsbäumen [9, S. 167]. ID3 steht für *Iterative Dichotomizer 3* und wurde von J.Ross Quinlan 1985 entwickelt [9, S. 167]. Der Algorithmus ist rekursiv, arbeitet zuerst jeden Pfad einzeln ab (depth-first) und erzeugt den Baum von der Wurzel hin zu den Blättern (top-down). Wie bereits erwähnt besitzt jeder Knoten ein beschreibendes Attribut, welches für eine Entscheidung verwendet wird. Der Algorithmus wählt die Attribute aufgrund ihres Informationsgehalts. Der Wurzelknoten beinhaltet das Attribut mit dem größten Informationsgehalt. Der Informationsgehalt wird mithilfe von Shannons Entropieformel berechnet. Die beschreibenden Attribute werden in absteigender Reihenfolge des Informationsgehalts für die Baumerzeugung verwendet.

Die Blätter sind das Ergebnis des Rekursionsankers, hier bricht der Algorithmus die weitere Erzeugung des Pfades ab. Jedes Blatt beinhaltet den Wert einer Klasse. Der Algorithmus erzeugt ein neues Blatt, wenn:

- alle Instanzen des partionierten Datensets das gleiche Zielattribut besitzen
- alle beschreibenden Attribute bereits für einen Knoten in diesen Pfad verwendet werden
- das partionierte Datenset leer ist (Blatt wird Wert von Elternknoten zugewiesen) [9, S. 136]

Der ID3-Algorithmus erzielt gute Resultate, benötigt jedoch ein sauberes, vollständiges Trainingsdatenset und kann nur kategorische Zielattributen verarbeiten [9, S. 167].

Beim *C4.5-Algorithmus* handelt es sich um eine Weiterentwicklung vom ID3-Algorithmus, welcher ebenfalls von J.Ross Quinlan entwickelt wurde. C4.5 kann im Gegensatz zu ID3 mit fehlenden Werten umgehen. Fehlende Werte werden nicht zur Entropieberechnung verwendet. Außerdem können numerische Attribute verwendet werden. Post-pruning (Subtree-Rasing [8, S. 193]) wird gegen Overfitting eingesetzt. Außerdem ist es möglich kontinuerliche Zielattribute zu bestimmen. Es wird fast ausschließlich C4.5 für die Erzeu-

gung von Entscheidungsbäumen verwendet [8, S. 188]. J48 ist die Open-Source Java-Implementierung, welche im Weka Toolkit verwendet wird [9, S. 167].

Der große Vorteil von Entscheidungsbäumen ist, dass sie für den Menschen lesbar und interpretierbar sind, also das getroffene Vorhersagen nachvollzogen werden können [9, S. 167].

3.1.2 Nächster Nachbar Algorithmus

Der *Nächste-Nachbar-Algorithmus* (Nearest Neighbor) gehört zum Konzept des ähnlichkeitsbasiertem Lernen (Similarity-Based Learning oder Instance-Based Learning). Es geht darum, Ähnlichkeiten mit bereits bekannten Informationen zu verknüpfen. Dazu ein Beispiel: Im Fernsehen wird ein Ballsportspiel übertragen, jedoch erkennt man den Sport am Anfang nicht. Man beginnt gewisse Attribute zu vergleichen. Es befinden sich zwei Spieler am Platz. In der Mitte ist ein Netz gespannt. Der Ball ist ungefähr faustgroß. Die Spieler verwenden Schläger. Es sieht so aus wie Tennis, also wird es auch Tennis sein. Dieses zugegeben sehr banale Beispiel gibt jedoch einen Einblick welches Prinzip ähnlichkeitsbasiertes Lernen folgt [9, S. 179].

Ein wichtiger Begriff für ähnlichkeitsbasiertes Lernen ist *Feature Space*. Der Feature Space projiziert jedes beschreibende Attribut auf eine Achse in einem Koordinatensystem. Man setzt voraus, dass es sich um numerische Attribute handelt. Feature Spaces lassen sich für zwei bis drei beschreibende Attribute auch hervorragend visuell als Koordinatensystem darstellen. Feature Spaces sind jedoch nicht auf eine Attributanzahl begrenzt [9, S. 181].

Ein weiterer Begriff ist das *Maß der Ähnlichkeit (Measure of Similarity)*. Durch Feature Spaces erhalten Attribute einen räumlichen Bezug. Das Maß der Ähnlichkeit misst den Abstand zwischen zwei Instanzen im Raum. Der Abstand kann auch über mehrere Dimensionen errechnet werden. Befindet man sich im zweidimensionalen Raum kann der Satz des Pythagoras verwendet werden. Im mehrdimensionalen Raum wird die Summe der Teilabstände gebildet. Wird der pythagoräische Lehrsatz zur Abstandsrechnung verwendet, spricht man von der euklidschen Distanz [9, S. 183].

Der Algorithmus wird trainiert, indem alle Trainingsinstanzen in den Speicher geladen und in einer Datenstruktur (vorzugsweise Liste) gespeichert werden. Dadurch ist der Algorithmus schon angelernt. Neue Trainingsdaten können jederzeit zur Laufzeit hinzugefügt werden. Gegeben ist das Datenset Q. Die Prädiktionsinstanz I berechnet sich im Feature Space von Q das Maß der Ähnlichkeit zu allen Trainingsinstanzen. I wird dasselbe Zielattribut zugewiesen, wie der Trainingsinstanz mit dem kürzesten Abstand, dem nächsten Nachbarn, mit dem höchsten Maß an Ähnlichkeit. Ein Nachteil ist, dass das Datenset eine bestimmte Größe nicht überschreiten darf, da es im Hauptspeicher gehalten werden sollte. Gegeben sei folgendes Beispiel:

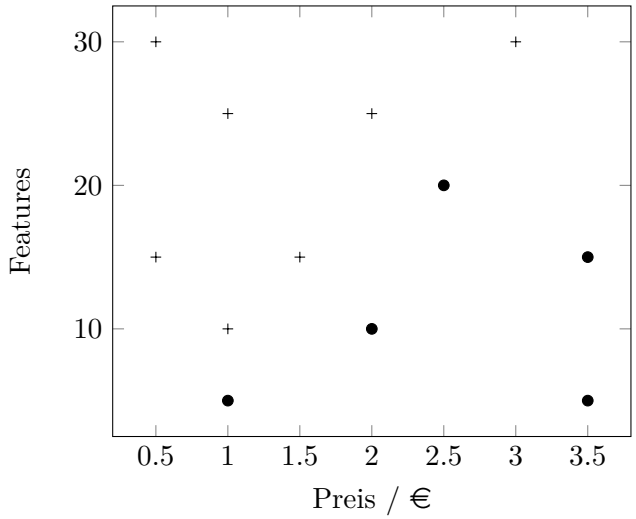


Abbildung 3.2: Feature Space für nearest neighbor Algorithmus.

Eine App wird im Appstore veröffentlicht. Dabei soll die Akzeptanz der Kunden vorausgesagt werden, ob sie dafür bereit sind Geld auszugeben. Die Trainingsdaten wurden bereits in einen Feature Space gebracht (Diagramm 3.2). Die Punkte bedeuten, dass die Benutzer nicht bereit waren dafür zu zahlen. Plussymbole beschreiben zahlende Benutzer. Wird nun eine neue App veröffentlicht, kann vorausgesagt werden, ob die Kunden diese auch kaufen werden. Angenommen eine App wird für 4 Euro angeboten, bietet aber nur 10 Features. Der nächste Nachbar besitzt das Zielattribut *nein*, deswegen ist es wahrscheinlich, dass Kunden die App nicht kaufen werden.

Das größte Problem beim nearest-neighbor-Algorithmus ist, dass er sehr sensitiv für verfälschte Datensätze ist. D.h., dass ein möglicherweise falscher Datensatz inmitten von richtigen Datensätzen ausgewählt werden kann, weil es sich dabei um den nächsten Nachbarn handelt. Der *k-nearest-neighbor-Algorithmus* ist eine Variation des ursprünglichen Algorithmus', der genau dieses Problem beseitigt. Dabei wird nicht der unmittelbar nächste Nachbar zur Bestimmung hinzugezogen, sondern die k nächsten Nachbarn, wobei k variabel ist. Das Zielattribut wird aus der Mehrheit der k nächsten Nachbarn bestimmt. Die Variable k muss dafür richtig gewählt werden. Ist k zu hoch tritt Underfitting (siehe 2.4) auf und die Genauigkeit der Vorhersagen nimmt stark ab. Um diesen Problem entgegenzuwirken gibt es den Begriff der *gewichteten Distanz*. Die Zielattribute von nahen Nachbarn werden mehr gewichtet als die Zielattribute von entfernten Nachbarn [9, S. 192].

3.1.3 Bayes Prognose

Das wahrscheinlichkeitsbasierte Lernen fußt – wie der Name vermuten lässt – auf der Wahrscheinlichkeitsrechnung. Und dabei fast ausschließlich auf dem Theorem nach Bayes. Jede Vorhersage ist im Grunde ein Ereignis, das mit der höchsten Wahrscheinlichkeit aller Ereignisse auftritt. Wahrscheinlichkeitsbasiertes Lernen geht den Ansatz nach, dass jedes Auftreten von neuer Information zu einer anderen Verteilung der Wahrscheinlichkeiten führt. Deswegen gibt es auch kein festes Modell, sondern das Modell wird geändert, sobald neue Informationen vorliegen. Damit setzt sich der Lernansatz von den anderen ab. Um diesen Lernansatz besser verständlich zu machen wird ein Geschäftsgebäude als Beispiel hinzugezogen. Das Geschäftsgebäude hat fünf Stockwerke. Es gilt zu ermitteln, in welchem Stock die Kunden ein Geschäft betreten. Die Daten werden einen Monat lang erhoben und liegen wie folgt vor: 1. Stock 20%, 2. Stock 35%, 3. Stock 15%, 4. Stock 20%, 5. Stock 10%. Aus folgenden Daten lässt sich ableiten, dass im 2. Stock die meisten Personen ein Geschäft betreten. Jedoch kann sich diese Information ändern, denn sobald eine Person den 1. Stock verlässt, fällt die Wahrscheinlichkeit für dieses Stockwerk weg. D.h., die Wahrscheinlichkeit wird nun für die restlichen vier Stockwerke neu berechnet. Würde die Person weiter in den 3. Stock gehen, so würde sich die Wahrscheinlichkeitsverteilung wieder anpassen. Ein Nachteil des Ansatzes ist jedoch, dass er nur mit kategorischen Attributen arbeiten kann.

Oft wird die bayes'sche Vorhersage synonym mit dem Wahrscheinlichkeitsbasierten Lernen verwendet. Das Theorem wird folgendermaßen beschrieben Die Formel beschreibt die Wahrscheinlichkeit, dass das Ereignis X auf-

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

Abbildung 3.3: Theorem nach Bayes.

tritt, wenn das Ereignis Y bereits aufgetreten ist, ist gleich groß wie die Wahrscheinlichkeit, dass Y aufgrund von X eintritt, multipliziert mit der Wahrscheinlichkeit des Ereignisses X gebrochen durch die Wahrscheinlichkeit, dass Y auftritt [9, S. 252]. Bei $P(X)$ $P(Y)$ handelt es sich um sogenannte A-priori-Wahrschein-lichkeiten. Das Gegenteil davon sind A-posteriori-Wahschein-lichkeiten. Solche werden durch ein *generalisiertes Bayes Theorem* errechnet. Davon spricht man, wenn man das Bayes Theorem erweitert, um alle beeschreibenden Attribute in der Formel abzubilden. A-posteriori-Wahrscheinlichkeiten sind jene Wahrscheinlichkeiten, mit welchem die eigentlichen Ereignisse auftreten. Der einfachste Ansatz ist der Maximum a-Posteriori-Ansatz (MAP). Dabei wird die Vorherssage aufgrund der größten

A-posteriori-Wahrscheinlichkeit in Abhängigkeit der a-priori-Wahrscheinlichkeiten der beschreibenden Attribute gefällt [9, S. 259].

Der *naive Bayes Klassifizierer* arbeitet nach dem MAP-Prinzip. Naiv deswegen, weil der Klassifizierer annimmt, dass die beschreibenden Attribute sich nicht gegenseitig beeinflussen und keine Abhängigkeiten besitzen. In den meisten Fällen trifft das aber nicht zu, was aber vernachlässigt werden kann. Der naive Bayes Klassifizierer klassifiziert ausreichend gut, ist sehr kompakt und robust gegen Störungen. Der naive Ansatz hat auch seine Nachteile. Zwar mag die Berechnung für kategorische Zielattribute ausreichen, aber für numerische Zielattribute ist der Algorithmus nicht ausgelegt [9, S. 267-268].

Neben dem naiven Bayes Klassifizierer gibt es auch noch *Bayes'sche Netzwerke*. Dabei handelt es sich um einen azyklischen Graphen. Sie bieten eine bessere Möglichkeit um die Unabhängigkeit der Attribute zu beschreiben und unterstützen auch numerische Attribute. Jeder Knoten im Graphen besitzt eine Tabelle, in welcher die Wahrscheinlichkeiten der Attribute der anderen Knoten abgebildet sind [9, S. 294]. Es gibt Ähnlichkeiten zum nearest-neighbor-Algorithmus, denn dort werden im Grunde A-priori-Wahrscheinlichkeiten zur Vorhersage verwendet. Mit ihnen ist es auch möglich eine künstliche Intelligenz zu erschaffen [9, S. 314-315].

3.1.4 Regression

Lineare und logistische Regression fallen in die Kategorie des fehlerbasierten Lernens. Die beste Lösung ist jene, mit dem geringsten Fehler. Der Fehler wird mittels einer Funktion beschrieben. Beim fehlerbasierten Lernen wird ein sogenanntes parametrisiertes Modell verwendet [9, S. 326]. Ein parametrisiertes Modell besitzt mehrere Parameter, die am Beginn zufällig gewählt werden. Ziel ist es, die Parameter so zu wählen, dass der Fehler zu den Trainingsinstanzen möglichst gering ist. Die *Lineare Regression* verwendet als parametrisiertes Modell eine Gerade. Die Funktion einer Gerade ist $y = k * x + d$, wobei k und d die Parameter des Modells sind. Dabei wird eine Gerade durch die Trainingsdaten gelegt, wobei k und d so gewählt werden sollen, dass der Fehler minimal ist. Als Fehlerfunktion wird die Summe der Fehlerquadrate verwendet. Für diese Funktion wird der durch die Gerade errechnete Wert mit dem eigentlichen Wert aus den Trainingsdaten subtrahiert. Das Ergebnis wird anschließend quadriert, damit es keine negativen Fehlerwerte gibt, da sich diese durch das anschließende Aufsummieren auslöschen würden. Als Beispiel kann die Automobilindustrie verwendet werden. Es gibt Aufzeichnungen vom Spritverbrauch von neu gebauten Autos zwischen 1990 und 2015. Das Baujahr ist in diesem Fall das beschreibende Attribut und der durchschnittliche Spritverbrauch das Zielattribut. Nun soll mithilfe von fehlerbasierten Lernen ein Modell erzeugt werden. Bei linearer Regression werden die Parameter k und d nach der Trial-And-Error-Methode

ermittelt. Zuerst wird k und d zufällig gewählt. Diese Werte werden geändert und der Fehler wird errechnet. Das passiert so lange, bis der Fehler minimal ist. Durch diese Methode ergibt sich für das Beispiel (Abbildung 3.4)

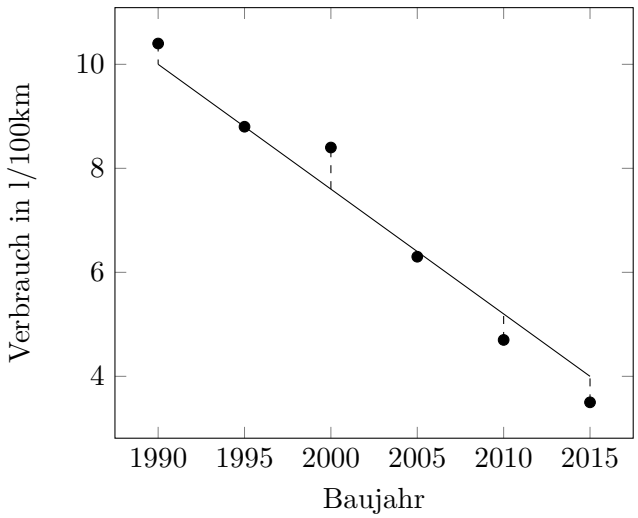


Abbildung 3.4: Lineare Regression für Kraftstoffverbrauch.

für k = -1.2 und d = 10. Das parametrisierte Modell für den Kraftstoffverbrauch in Abhängigkeit vom Baujahr lässt sich folgendermaßen beschreiben: $Verbrauch = -1.2 * Baujahr + 10$. Ein Nachteil von linearer Regression ist, dass keine kategorischen beschreibenden Attribute verarbeitet werden können.

3.1.5 Künstliche neuronale Netze

Künstliche neuronale Netze gehören ebenfalls zum fehlerbasierten Lernansatz [9, S. 387]. Der Mensch kopiert oft bei neuen Erfindungen die Natur. Und ein mustergültiges Beispiel der Informationsverarbeitung und Objekterkennung ist das menschliche Gehirn. Das menschliche Gehirn arbeitet jedoch auf biologischer Basis. Künstliche neuronale Netze basieren auf einem mathematischen Modell, welches ein menschliches Gehirn simuliert. Deswegen der Begriff *künstlich*, um sie von biologischen neuronalen Netzen abzugrenzen (in der Literatur wird meistens nur von neuronalen Netzen gesprochen). Laut Gupta [7, S. 1] sind die großen Vorteile von neuronalen Netzen:

- Enorme Parallelität
- Verteilte Berechnung
- Lernfähigkeit
- Fähigkeit zur Abstrahierung
- Anpassungsfähigkeit

- Informationsverarbeitung
- Fehlertoleranz
- Niedriger Energieverbrauch

Neuronale Netze bestehen aus einer Menge von künstlichen Neuronen, welche den biologischen Neuronen nachgebildet sind. Neuronen besitzen eine Eingangsgröße und ein Gewicht. Die Ausgangsgröße ist die Eingangsgröße multipliziert mit dem Gewicht. Meistens wird ein Schwellwert verwendet um zu überprüfen, ob das Neuron aktiviert wird und durchschaltet [7, S. 1]. Als Aktivierungsfunktion wird dafür ein *Sigmoid (logistische Funktion)* verwendet [4, S. 6]. Neuronale Netze spezialisieren sich auf ein Anwendungsgebiet, weil sie eben nur mit genau dafür vorgesehen Trainingsdaten trainiert werden. Bei der Menge an Neuronen muss das Gewicht bestmöglich eingestellt werden um den gewünschten Lerneffekt zu erzielen. Mithilfe von fehlerbasiertem Lernen werden die Gewichte für jedes Neuron optimal eingestellt. Zu Beginn werden die Gewichte zufällig angenommen und der Algorithmus probiert durch Trial-And-Error den geringsten Fehler zu ermitteln.

Man unterscheidet grundsätzlich zwischen 2 Arten: *Feedforward-Netze* können als statische Graphen ohne Rückkopplung angesehen werden. Sie verbrauchen weniger Speicher als sogenannte *Feedback-Netze*. Feedback-Netze besitzen sehr wohl eine Rückkopplung, benötigen aber eben mehr Speicher und durch die Rückkopplungen sind sie auch abhängig vom vorherigen Status [7, S. 2]. Die Hauptanwendung von neuronalen Netzen besteht in der Mustererkennung und im Kategorisieren (Clustering) von Daten [1, S. 10]. Für das Training kann entweder Supervised-, Unsupervised- oder Reinforcement Learning verwendet werden. Unsupervised Learning eignet sich hervorragend für Clustering (siehe 3.2.2. Ein sehr beliebter Ansatz ist das *Multilayer Perceptron*. Dabei handelt es sich um ein mehrschichtiges Feedforward Netze, wobei jeder Knoten im Layer mit jedem Knoten im nachfolgenden Layer verbunden ist (siehe 3.5). Als Trainingsalgorithmus wird Back-Error-Propagation verwendet [4, S. 2]. Neuronale Netze besitzen eine Eingangs- und Ausgangsschicht (engl. Input-Output-Layer) und eine beliebige Anzahl von versteckten Schichten (engl. Hidden layer). Die versteckten Schichten werden benötigt um Funktionen zu approximieren. Das neuronale Netz erzielt bei einer Anstieg der versteckten Schichten dementsprechend besser. Die Komplexität (die Anzahl der versteckten Schichten und die Anzahl der Knoten pro Schicht) des neuronalen Netzes hängt von der eigentlichen Problemstellung ab. [15, S. 2].

Neuronale Netze verstehen vier Lernregeln. Das *Error-correcting Learning* minimiert bei jeder Iteration den Fehler des gesamten Netzwerkes. Die meisten Netze werden nach diesem Typen gelernt. Daneben gibt es noch *das Lernen nach Boltzmann* [1, S. 11] und das *Hebb'sche Lernen*. Bei *Competitive Learning* geht es darum, dass sich verschiedene Ausgangsknoten um eine Aktivierung bemühen müssen. Pro Iteration wird somit auch nur ein Aus-

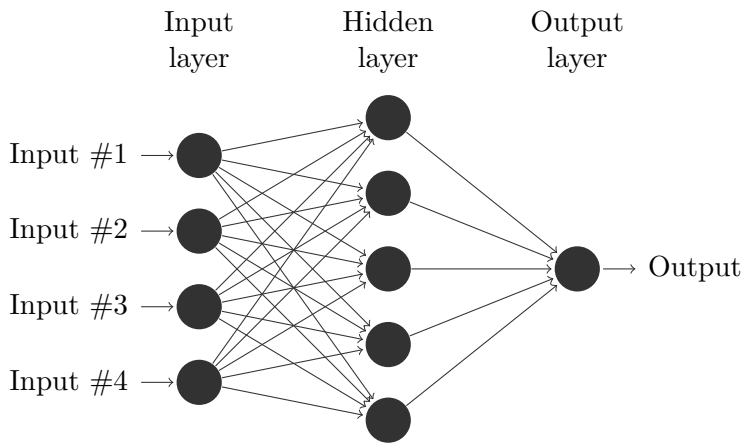


Abbildung 3.5: Multilayer Perceptron (MLP).
Quelle: http://www.texample.net/tikz/examples/neural-network

gangsknoten aktiviert. Diese Lernregel wird auch in biologischen neuronalen Netzen beobachtet [7, S. 4].

3.2 Machine Learning Konzepte

Im Grunde gibt es vier Machine Learning Konzepte, nach denen ein Prediction Model vorgeht: Klassfizierung, Assoziierung, Clustering und Regression. Eine Klassifizierung bedeutet exakt ein Zielattribut des Datensatzes zu bestimmen, was in vielen Fällen ausreicht. Das Konzept Assoziierung versucht Verbindungen zwischen den Datensätzen zu finden. Dabei wird nicht nur ein Zielattribut bestimmt, sondern es können mehrere Zielattribute unabhängig voneinander bestimmt werden. Das Clustering beschreibt das Zusammenfassen von Gemeinsamkeiten der Datensätze. Die Regression ähnelt der Klassifizierung, jedoch wird kein diskretes Zielattribut bestimmt, sondern ein numerischer Wert. Je nach Konzept wird der dazu passende Lernansatz ausgewählt. Bei der Klassifizierung geht man davon aus, dass bereits klassifizierte Trainingsdaten vorliegen. Solche Daten werden im Englischen als *labeled data* bezeichnet [9, S. 7]. Trainingsdatensets bestehen aus sogenannten Instanzen. Eine Instanz ist ein Datensatz innerhalb des Sets. Trainingsdaten spielen vor allem bei Supervised Learning eine entscheidende Rolle. Dafür gibt es andere Lernansätze, die auf Trainingsdaten verzichten, wie etwa das Reinforcement Learning3.2.3.

3.2.1 Supervised Learning

Der Begriff *Supervised Learning* bedeutet übersetzt überwachtes Lernen. Supervised Learning findet vor allem bei der Klassifizierung Anwendung. Der

Algorithmus wird mit Trainingsdaten angelernt, wobei die Klassifizierung der Daten bereits stattgefunden hat. Der Maschine wird schon beim Training mitgeteilt welche Daten wie einzustufen sind. Und meistens ist es genau so, dass bereits erhobene Daten der Maschine als Lerngrundlage dienen. Die Maschine lernt nur aus dem, was ihr der Überwacher (Supervisor) zu lernen gibt. Bei der Klassifizierung sind meistens einzelne – nicht vorhandene – Attribute von Datensets relevant. Wie gut eine Maschine durch Supervised Learning funktioniert kann sehr einfach bestimmt werden. Dabei werden der Maschine Datensätze zugeführt, welche ebenfalls bereits klassifiziert wurden, aber diese Ergebnisse werden der Maschine nicht mitgeteilt. Die Ergebnisse der Maschine werden im Nachhinein mit den bereits erhobenen Daten verglichen [8].

Supervised Learning erzeugt das Modell nur mithilfe der gelieferten Trainingsdaten. Das Trainingset beinhaltet einen oder mehrere Eingangsparameter X und einen Ausgangsparameter Y. Der Ausgangsparameter wird auch als Klasse bezeichnet. Ziel des angelernten Systems ist es nun aufgrund von gegebenen Eingangsparametern X auf einen sinnvollen – nicht gegebenen – Ausgangsparameter Y zu schließen. Voraussetzung dafür ist, dass es sich um gültige Eingangsparameter handelt und dass das System diese auch richtig verarbeiten kann. Diese Methode eignet sich nur für die Konzepte Klassifizierung und Regression [22].

3.2.2 Unsupervised Learning

Unsupervised Learning wird vor allem bei *Clustering* verwendet. Der große Unterschied zu Supervised Learning besteht darin, dass bei Unsupervised Learning keine expliziten Ergebnisse der Trainingsdaten vorliegen. Das System verarbeitet Datensets, die keine zu bestimmende Klasse besitzen. Das Modell wird aufgrund von Beobachtungen der Daten erzeugt. Das Problem bei Unsupervised Learning ist, dass es im Gegensatz zu Supervised Learning und Klassifizierung keine definierten Klassenwerte gibt. Wie der Name schon sagt werden die Datensets bei Clustering in verschiedene Gruppierungen (Cluster) unterteilt. Die Unterteilung erfolgt aufgrund des Modells [22]. Wie gut Unsupervised Learning funktioniert kann nur subjektiv beurteilt werden [8].

3.2.3 Reinforcement Learning

Bei *Reinforcement Learning* setzt man auf das Prinzip *Trial-And-Error*. Der Agent agiert interaktiv mit einer dynamischen Umgebung. Es stehen keine Trainingsdaten zur Verfügung. Die einzige Möglichkeit zu lernen, besteht darin, das man verschiedene Möglichkeiten auszuprobiert.

Wie in Abbildung 3.6 ersichtlich, interagiert der Agent mit seiner Umwelt. Er sendet eine Aktion aus und die Umwelt – als Gegenspieler – sendet

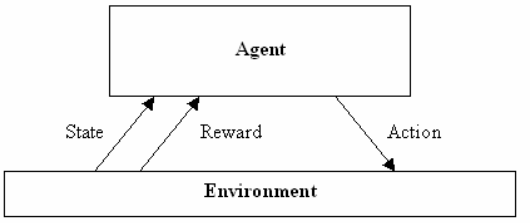


Abbildung 3.6: Reinforcement Learning Modell.
Quelle: http://www2.hawaii.edu/čhenx/ics699rl/grid/rl.html

zwei Signale zurück. Das Signal S ist die Ausgangsgröße, das Signal R steht für Reward und wird auch Reinforcement Signal genannt. Beim Signal R handelt es sich um eine skalare Größe, meistens 0 oder 1 und beschreibt die Güte der Handlung. Ziel des Agenten sollte es sein, dass sich die Summe von Signal R langfristig erhöht. Damit ist gemeint, dass der Algorithmus immer die richtigen Entscheidungen treffen [12, S. 1-3].

Meistens spielt Reinforcement Learning eine untergeordnete Rolle, da sich ein Großteil der Probleme mit einer Klassifizierung lösen lässt. Zu den Hauptunterschieden zwischen Reinforcement Learning und Supervised Learning zählt aber, dass es keine konkret Ein- und Ausgangspaare gibt [14, S. 7-15] [21, S. 1-2].

Der große Vorteil von Reinforcement Learning gegenüber Supervised Learning ist jedoch, dass Reinforcement Learning in einer unbekannten Umwelt ohne jegliche Unterstützung eigenständig lernen kann. In der Literatur wird als Anwendungsfall eine interaktive Umgebeung genannt [14, S. 15].

3.2.4 Semisupervised Learning

Der Unterschied zwischen Supervised und Unsupervised Learning besteht darin, dass bei Supervised Learning die Trainingsdaten beschriftet sind. D.h., jede Instanz der Trainingsdaten beinhaltet eine bereits bekannte Klasse. Supervised Learning erzielt gute Resultate, aber die Aufbereitung der Trainingsdaten ist sehr aufwendig. Angenommen man hat mehrere Millionen nicht klassifizierte Datensätze, wie sie etwa für Unsupervised Learning verwendet werden. Diese Daten müssten zuallererst von Menschenhand klassifiziert werden. Aus diesem Grund geht die Forschung in die Richtung von *Semisupervised Learning*. Der Ansatz dahinter ist so simpel wie elegant: Man hat eine kleine Anzahl von klassifizierten Trainingsdaten. Diesen klassifizierten Trainingsdaten steht ein ungleich größeres Datenset an nicht klassifizierten Datensätzen gegenüber. Um diese Daten miteinander zu verknüpfen wird die EM-Methode[7] angewandt. Zuallererst wird ein Klassifizierer mit den

[7]Expectation-Maximization-Methode

klassifizierten Trainingsdaten angelernt. Dabei empfiehlt es sich auf einen Wahrscheinlichkeitsansatz3.1.3 zurückzugreifen. Der Klassifizierer wird verwendet um das riesiege nicht klassifizierte Datenset zu klassifizieren. Daraus entsteht ein neues Datenset, das nun eine enorme Anzahl an klassifizierten Datensätzen beinhaltet. Dieser Schritt wird bei der EM-Methode als der Expectation-Step bezeichnet. Das neue Datenset wird wiederum verwendet, um einen neuen Klassifizierer anzulernen (der Maximation-Step), der jetzt viel präziser als der erste arbeiten sollte [8, S. 337]. Die Umsetzung erweist sich aber nicht ganz so trivial, da einige Probleme auftreten können. Eine Klassifizierung durch einen Menschen ist in manchen Fällen genauer als eine automatische Klassifizierung. Deswegen wird diese in der Praxis oft mehr gewichtet, um die Genauigkeit der Vorhersagen zu erhöhen [25, S. 1].

3.3 Bestehende API Anbieter

Es gibt einige Firmen, die ihre Infrastruktur als ein Service anbieten (Infrastructure-as-a-service). Vor allem kleine Unternehmen oder generell Entwickler können von solchen Lösungen profitieren, da in diesen Fällen womöglich das Geld für eine eigene Infrastruktur fehlt. In Verbindung mit Machine Learning ergibt sich ein riesiges Potential, was jeder Entwickler für wenig Geld verwenden kann. Einige Lösungen werden auch als Open Source angeboten, wie etwa das Weka[8] Toolkit [34] oder die Plattform Prediction.io [30]. Diese Systeme setzen jedoch die Infrastruktur voraus, womit auch der Hostrechner zur Verfügung gestellt werden muss. Im folgenden wird auf die bereits bestehende API Anbieter eingegangen. Ein besonderes Augenmerk liegt dabei auf den jeweiligen Kosten und die Verfügbarkeit der Programmiersprache Java.

3.3.1 Google Prediction API

Googles Cloud Chef Urs Hölzle behauptete zuletzt, dass Google bis 2020 mehr Geld mit ihrer Cloud Plattform, als mit dem Kerngeschäft der Werbung lukrieren werde [35]. Weiters führt er an, dass sich Google in fünf Jahren als Cloud Konzern versteht. Google bietet schon eine Fülle von Diensten auf ihrer Cloud Plattform an. Allen voran die Google App Engine, welche die Datenspeicherung für Webseiten und mobile Anwendungen ermöglicht. Mit dabei ist auch ein Prediction API. Der große Vorteil dieser Plattform ist, dass beinahe alle Services über eine einheitliche Schnittstelle zugänglich sind. Die APIs variieren natürlicherweise je nach Anwendungsgebiet. Aber die Authentifizierung aller APIs erfolgt über das OAuth2 Protokoll. Dabei ist die Anmeldung für das Gmail API ident mit der des Prediction APIs. Dadurch ergibt sich eine vergleichsweise geringe Einarbeitungszeit für das API, wenn

[8] Waikato Environment for Knowledge Analysis

man bereits mit anderen APIs von Google gearbeitet hat. Wie von Google gewohnt wird für die (vor allem im Webbereich) geläufigsten Programmiersprachen ein API angeboten. Es gibt APIs für Go, Java, JavaScript, .NET, Node.js, Objective-C, PHP, Python und Ruby. Das Java-API ist reichlich wenig dokumentiert. Die Benutzung des Services ist nicht gänzlich gratis. Es gibt eine Freimenge von 100 Predictions und fünf Megabyte an Trainingsdaten pro Tag, begrenzt aber auf maximal 20000 Predictions überhaupt. Die Freimenge läuft nach sechs Monaten nach Aktivierung des Accounts automatisch ab. Wie aus der Dokumentation hervorgeht wird nur eine Regression und Klassifizierung unterstützt. Jedoch gibt es von Seiten Googles bereits zur Verfügung gestellte Prediction Models: für Spracherkennung (englisch, spanisch, französisch), Tag-Erkennung und Gefühlserkennung von Texten. Die bereitgestellten Modelle sind vielmehr für die Einarbeitung und Kennenlernen des Services gedacht, als wirklich für den produktiven Einsatz. Ein entscheidender Vorteil ist, dass der gesamte Workflow über das API abgewickelt werden kann und keine zusätzliche Software dafür benötigt wird.

3.3.2 BigML

Bei BigML handelt es sich um einen kommerziellen Dienst, der sich ausschließlich darauf spezialisiert hat, Machine Learning Infrastruktur anzubieten [26]. Grundsätzlich ist BigML kostenpflichtig, bietet aber einen sogenannten Entwicklermodus an. Der Entwicklermodus ist kostenlos, beschränkt sich aber auf Aufgaben die kleiner als 16 Megabyte sind. Die Bezahlung erfolgt entweder über fixe monatliche bis jährliche Abonnements, oder – wie es auch bei Googles ist – nach dem Pay-as-you-go-Prinzip. Dabei können Credit Points für Speicherplatz und Predictions gekauft werden. Die Preise starten bei 2$ für 10 Megabyte und 2000 Predictions und können im Terabytebereich bis zu mehreren tausend Dollar gehen. BigML bietet dafür sehr umfangreiche Features. So ist es möglich seine Prediction Models als PMML-Dateien zu exportieren. Die Prediction Models können über ein einfaches und intuitiv gestaltetes Webinterface erstellt und bearbeitet werden. Zusätzlich bietet BigML auch noch diverse APIs für Java, Python, Bash, C#, R, Ruby, Node.js, Objective-C/Swift und php an. Eines der interessantesten Features ist, dass es öffentlich zugängliche Datasets und Prediction Models zum kostenfreien Download gibt. Jeder Benutzer kann seine erstellten Prediction Models veröffentlichen. Ein weiterer Bonus ist der sehr umfangreiche Funktionsumfang. Neben den bereits erwähnten APIs gibt es auch ein Google Sheets Add-On und ein Kommandozeilen-Tool namens BigMLer. Nichtsdestotrotz ist das Java API wenig dokumentiert. Zudem verwendet BigML Entscheidungsbäume für ihre Prediction Models [27]. Das ist dahingehend interessant, da es sich bei Entscheidungsbäumen um Whitebox-Systeme handelt und der Entwickler mit einem transparentem System arbeiten kann.

3.3.3 Prediction.io

Prediction.io ist ein quelloffener Machine-Learning-Server. Das technisches Fundament bilden die Technologien Apache Spark, HBase und Spray [30]. Als Zielgruppe sind Entwickler und Datenanalysten angeführt. Diesen soll durch Prediction.io geholfen werden, prädiktive Anwendungen in einem Bruchteil der ansonsten dafür benötigten Zeit zu veröffentlichen. Es sind bereits viele Modelle von anderen Entwicklern bereitgestellt und können direkt in eigene Anwendungen integriert werden. Ein Nachteil ist, dass der Server manuell installiert werden muss, was aber ein unixoides Betriebssystem voraussetzt. Dafür gibt eine einfache Integration für die Amazon Web Services, damit der Server dort gehostet wird. Zusätzlich werden noch weitere Softwarekomponenten benötigt, wie etwa Apache Hadoop und Apache Spark, sowie zumindest Java 7. Da der Entwickler den Server selbst hostet, hat er natürlich volle Kontrolle über das System. Prediction.io unterstützt auch die native Bibliothek Spark MLlib[9], welche die verschiedensten Machine Learning Algorithmen implementiert. Hier kann nach eigenen Ermessen ein Algorithmus gewählt werden. Von Prediction.io selbst werden SDKs für Java, PHP, Python und Ruby zur Verfügung gestellt. Es gibt auch SDKs, die von der Community entwickelt werden. Darunter befinden sich die SDKs für Node.js, C# / .NET und Swift. Man sieht dem System aber an, dass es sich zur zum Verfassungszeitpunkt noch um keine finale Version handelt (10.12.2015 - Version: 0.9.5). Einige Beispiele sind nicht für alle APIs bereitgestellt, und die SDKs selbst sind dürftig bis mittelmäßig dokumentiert.

3.3.4 Weka Toolkit

Beim Weka Toolkit handelt es sich um eine Software für alle gängigen Betriebssyteme. Die Software ist in Java geschrieben und deswegen plattformunabhängig. Es wurde speziell für Machine Learning und Data Mining entwickelt. Mit dem Toolkit können verschiedene Algorithmen für Machine Learning angewendet werden. Es ist ein sehr umfangreiches Tool und wird vor allem an Universitäten und Fachhochschulen verwendet, deswegen wird es auch hier angeführt. Ein Nachteil von Weka ist, dass das Machine Learning sowie die Vorhersagen direkt am Gerät ausgeführt werden, was natürlich Rechenleistung benötigt. Weka bietet auch ein Java API an, um die Funktionalität in eigene Anwendungen einzubinden.

3.3.5 Amazon ML

Bei den Amazon Web Services handelt es sich um das Pendant zu Googles Cloud Plattform und Microsofts Azure Plattform. Die Anmeldung ist denkbar einfach und das Webinterface sehr übersichtlich und intuitiv. Die

[9] Machine Learning Library

Amazon Web Services sprechen vor allem Unternehmen an, da sie sehr flexibel und erweiterbar sind. Die Services sind in diverse Kategorien unterteilt, wie etwa Mobile und Datenbanken (um nur 2 zu nennen). Wie bei jeder größeren Cloud Plattform wird auch eine Machine Learning Lösung angeboten. Amazon hat hier gute Arbeit geleistet und stellt ein umfangreiches SDK zur Verfügung um auf alle Services zuzugreifen. Das SDK wird für Java, .NET, Python, PHP, Node.js, Ruby, Android und iOS angeboten. Außerdem gibt es ein Kommandozeilen-Tool und auch ein SDK für Unity. Neben den verfügbaren SDKs gibt es auch ein eigenes Eclipse-Plugin. Das API ist sehr gut dokumentiert, aber durch den Umfang verliert man relativ schnell den Überblick. Bei der Preisgestaltung wendet sich Amazon wie die Konkurrenz an das Pay-as-you-go-Prinzip. Datenanalyse und Modellaufbau schlagen mit 0,42$ pro Stunde zu Buche. 1000 Batch-Prognosen kosten 10 Cent und jede Echtzeit-Prognose kostet 0,01c plus einer stündlichen Gebühr. Das Konzept von Amazon wirkt sehr durchdacht und ausgereift. Deswegen ist Amazon sicherlich eine gute Anlaufstelle für Unternehmen, wenn es um cloud-basierte Lösungen geht.

3.3.6 Microsoft Azure ML

Bei Microsoft Azure handelt es sich um Microsofts Cloud Plattform. Zu dieser Plattform gehört auch der Dienst Azure ML, ein Online Tool für Machine Learning. Das Tool ist schön und einfach gestaltet und bietet damit auch technisch weniger versierten Nutzern eine Möglichkeit Machine Learning einzusetzen. Die meisten Funktionen basieren auf Drag-and-Drop. So ist es möglich, Datensets und Prediction Models per Hand hineinzuziehen. Dennoch ist im großen und ganzen die Microsoft Azure Struktur sehr unübersichtlich. Es gibt in der Cloud Plattform keine einheitliche Navigation. Zwar gibt es einen freien Zugang, welcher aber äußerst eingeschränkt ist. Gedrosselter Zugriff über das API kann nur während der Entwicklungsphase erfolgen. Die Anmeldung für Privatpersonen gestaltet sich als sehr umständlich. Hier kristallisiert sich, dass Microsoft mit Azure vor allem Unternehmen als Kunden gewinnen will. Microsoft kommt hier aber Studenten entgegen und bietet ihnen erleichterten Zugang zu ihren Diensten. Auch Microsoft bietet vorgefertigte Modelle an. Im speziellen für Bilderkennung, Spracherkennung, Recommendations, Übersetzer, Wettervorhersage und Gesichtserkennung. Das Problem dabei ist, dass Microsoft nur APIs für C# und Python anbietet. Grundsätzlich handelt es sich um rein REST-basiertes Webservice, so wäre es also möglich, ein API selbst zu schreiben, wovon aber aufgrund der Komplexität der Schnittstelle abgeraten wird. Bei eigenen Prediction Models ist der Zugriff über ein API ebenfalls sehr umständlich gelöst. Bevor ein API auf ein Prediction Model zugreifen darf, muss das Model zuerst als Webservice veröffentlicht werden. Für Unternehmenskunden mag Microsoft Azure womöglich die passende Wahl sein, aber für einen Betrieb außerhalb

des Microsoft/.NET Ökosystems scheint es eher weniger geeignet.

3.3.7 Wolfram Alpha API

Wolfram Alpha ist im erweiterten Sinne eine Suchmaschine. Der Online-Dienst versteht auch komplexe Fragen und kann diese beantworten. Zusätzlich kann Wolfram Alpha sehr umfangreiche mathematische Formeln lösen und die Ergebnisse grafisch darstellen. Eigentlich handelt es sich bei Wolfram Alpha um keinen API Anbieter, wie in den vorherigen Beispielen. Jedoch bietet der Dienst zwei implementierte Sprachbefehle für Machine Learning. Die Befehle *Predict* und *Classify* können für Machine Learning/Data Prediction verwendet werden. Die Anwendung sowie die Anwendungsbereiche sind eher mathematischer Natur, und deswegen auch meistens nicht für klassische Informatikprojekte geeignet. Wolfram Alpha stellt aber umfangreiche APIs zur Verfügung. So ist es möglich mittels Perl, Python, .NET, Ruby, PHP, Java und Mathematica auf den Online-Dienst zuzugreifen.

Kapitel 4

Big Data

Durch mobile Geräte und beinahe flächendeckenden Internetempfang steigt auch die Anzahl der aufzeichenbaren Daten. Firmen wie Google, Amazon, Facebook und Apple haben Zugang zu den enormen Datenmengen ihrer Benutzer. Meistens sind die erfassten Daten äußerst heterogen. Sie setzen sich aus strukturierten und unstrukturierten Daten zusammen [18, S. 2]. In Kapitel 2.1 wurde bereits der Begriff Big Data erklärt. In diesem Kapitel geht es vor allem um die Probleme und Konzepte von Big Data, wenn Big Data für Machine Learning und Data Prediction verwendet werden soll. Von Big Data spricht man von Datensets in einer Größenordnung von 30-50 Terabyte [18, S. 2]. Bei einfachen Machine Learning Algorithmen, wie etwa beim k-nächster-Nachbar-Algorithmus, werden alle Trainingsdaten im Hauptspeicher verwaltet. Die Größe des Hauptspeichers beläuft sich in der Regel zwischen 4 - 128 Gigabyte. Also nicht ausreichend um Terabyte an Trainingsdaten zu verarbeiten. Die meisten Algorithmen sind aufgrund dieser Limitierung für solche Probleme nicht ausgelegt. Es gibt generell drei Lösungen, wenn Datensets zu groß sind, um im Hauptspeicher gehalten werden zu können. Die erste Möglichkeit ist, dass ein Subset aus den gesamten Datenset entnommen wird. Wenn ein Datenset viel Redundanz enthält, reicht ein Subset aus um ohne Informationsverlust die Daten zu repräsentieren. Parallelisierung ist die zweite Möglichkeit um große Datensets zu verarbeiten. Ein Beispiel dafür sind Ensembles (siehe 3.1.1). Bei Ensembles werden mehrere Entscheidungsbäume mit verschiedenen Daten aus einem Datenset angelernt und das Ergebnis setzt sich aus allen Entscheidungsbäumen im Ensemble zusammen. Die dritte und letzte Möglichkeit sieht die Entwicklung neuer Algorithmen vor, die speziell auf große Datenmengen ausgelegt sind. Meistens ist die dritte Möglichkeit nicht zu realisieren. Im Grunde geht es aber darum, mithilfe von Hintergrundwissen die Daten auf das nötigste zu reduzieren [8, S. 346-348].

Neben dem Datenzugriff und der Datenspeicherung gibt es ein zweites, häufig auftretendes Problem bei Machine Learning: *Overfitting*. Overfitting

wurde schon in Kapitel 2.4 erwähnt. Es tritt bei zu kleinen und zu ähnlichen Datensätzen auf. Durch die enorme Anzahl von Datensätzen kann Overfitting vermieden werden, wobei der Effekt auch durch andere Faktoren beeinflusst werden kann. Overfitting ist ein nicht zu unterschätzender Fehlerfaktor.

4.1 Datenverarbeitung

Zu den beliebtesten Datenbanksystemen zählt das relationale Datenbanksystem SQL, obwohl es signifikante Einschränkungen bei der Verarbeitung von großen Datenmengen aufweist. In den Datenbanken werden Tabellen (Relationen) abgebildet. Jede Tabelle enthält beliebig viele zeilenweise Einträge, welche allem einem zuvor definierten Datenbankschema folgen. Seit Jahrzehnten dominiert dieser Ansatz. Doch relationale Datenbanken bieten einige gravierende Nachteile, wenn es darum geht, große Datenmengen effektiv abzubilden. Relationale Datenbanken lassen sich schlecht skalieren. Darüber hinaus bieten sie eine schlechte Schreib- und Lese-Performance, was der Datenkonsistenz geschuldet ist. Rasch ändernde Daten können nur schlecht verarbeitet werden. Und wahrscheinlich der größte Nachteil ist, dass diese Systeme einen vordefinierten Datenbankschema folgen, was sie insgesamt sehr unflexibel macht [2].

In den letzten Jahren tendierte der Trend deswegen in Richtung von NoSQL-Datenbanken[10]. Generell sind NoSQL-Datenbanken um einiges flexibler, können dynamisch neue Attribute zu Aufzeichnungen hinzufügen und halten sich nicht an das ACID-Prinzip[11] [2, S. 1]. Während bei relationalen Datenbanken ein zentraler Server die bevorzugte Variante ist, können NoSQL-Datenbanken über mehrere Server hinweg linear skalieren und Daten verteilen. Dadurch lassen sich sehr viele einfache Lese- und Schreibbefehle ausführen, was bei relationalen Datenbanksystemen mit einem enormen Performance-Verlust einhergehen würde. Dokumentorientierte NoSQL-Datenbanken wie MongoDB oder CouchDB erfreuen sich immer größerer Beliebtheit. Sie speichern Dokumente, in den meisten Fällen JSON-Dateien [2]. Der Vorteil von diesen System ist, dass sie auch unstrukturierte Daten, wie etwa menschliche Sprache, Bilder oder Filme speichern können. Wenn man von großen Datenmengen spricht, dann bestehen diese sowohl aus strukturierten bzw. unstrukturierten Daten. Das Speichern von den unstrukturierten Daten stellt in vielen Fällen das größere Problem dar.

Firmen, deren Tagesgeschäft das Auswerten von Daten ist, haben natürlich gänzlich andere Anforderungen für ein Datenbanksystem. Es ist nicht verwunderlich, dass Google ein eigenes verteiltes Datenbanksystem namens *Bigtable* verwendet. Die Entwickler beschreiben *Bigtable* als ein verteiltes,

[10] Not only SQL

[11] Atomicity, Consistency, Integrity, Duralibility

hochskalierbares Speichersystem für strukturierte Daten, womit mehrere Petabyte an Daten über mehrere tausend Server verteilt verarbeitet werden können [3, S. 1]. Der Fokus wurde dabei auf Skalierbarkeit, Performance, und Verfügbarkeit gelegt. Als Datenmodell kommt eine verteilte multidimensionale sortierte Map zum Einsatz. Neben Googles eigener Lösung gibt es die freie Software *Apache Hadoop*. Bei Apache Hadoop handelt es sich um ein verteiltes, skalierbares System [28]. Auf diesem Grundgerüst wird die Funktionalität in mehrere Schichten abstrahiert. Das Herzstück ist der *Hadoop Storage Layer*, wobei das Dateisystem *HDFS*[12] die wichtigste Komponente ist. Bei HDFS handelt es sich um ein hochskalierbares Dateisystem, das sich wie Bigtable über eine Vielzahl von Maschinen spannen kann [18, S. 14]. Typischerweise gibt es mehrere Schichten oder Layer. Der *Ingestion Layer* speißt neue (strukturierte sowie unstrukturierte) Daten in das System. Der *Hadoop Platform Management Layer* handhabt den eigentlichen Zugriff auf die verteilte Datenbank [18, S. 10]. Dabei wird ein von Google entwickelter Algorithmus verwendet: MapReduce. MapReduce besteht aus zwei Methoden: Map und Reduce. Ziel des Algorithmus' ist, dass mehrere Anfragen parallel in der Map-Methode behandelt werden können. Danach kümmert sich die Reduce-Methode damit, die Ausgaben der Map-Methode zu einer Antwort zu kombinieren [18, S. 17]. Die Verarbeitung folgt dem funktionalen Paradigma. Es gibt noch weiter Layer, wie etwa der *Analytics Engine Layer* und der *Data Warehouse Layer*. Hier können die Daten mittels Machine Learning-Algorithmen verarbeitet werden.

4.2 Overfitting

Für Yaser [23, S. 119] ist die Fähigkeit mit *Overfitting* umzugehen, der Unterschied zwischen einem profesionellen Datenanalysten und einem Amateur. Das Problem des Overfitting tritt sehr häufig auf, wobei das Erkennen von Overfitting schon die erste Schwierigkeit ist. Der Effekt beschreibt wortwörtlich, dass das mathematische Modell zu eng an den eigentlichen Daten anliegt. Overfitting kann aus mehreren Gründen auftreten. Der Hauptgrund ist ein unzureichendes Datenset. Im Idealfall hat man Zugriff auf Big Data, oder Datensätze mit einer zumindest ausreichend guten statistischen Verteilung. Mehrere Datenpunkte senken das Risiko für Overfitting, während fehlerhafte Daten und ein zu kompliziertes Modell andererseits Overfitting begünstigen [23, S. 124]. Was versteht man aber unter einem zu komplizierten Modell? Man nehme als Beispiel eine Regression. Es liegen 20 Datenpunkte vor. Nun hängt es von den Freiheiten des Algorithmus' ab, ob er zu Overfitting führt. Angenommen es würde eine Gerade ausreichen um die beste Prädiktion zu erzielen. Erlaubt man den Algorithmus jedoch eine Polynomfunktion bis zum Grad x zu verwenden, so wird er diese Freiheit ausnutzen eine Funktion zu

[12] Hadoop distributed file system

berechnen, welche durch alle 20 Datenpunkte verlaufen würde. Die Funktion liegt dabei in diesen Punkten direkt am eigentlichen Wert. Und obwohl eine Gerade für eine ausreichend gute Prädiktion besser geeignet wäre, wird eine Polynomfunktion x. Grades verwendet. Um den Fehler zu messen unterscheidet man zwischen *In-Sample-Fehler*[13] und *Out-Of-Sample-Fehler*[14]. Der In-Sample-Fehler kann während der Berechnung ermittelt werden. Es ist die Abweichung des errechneten Punktes zum eigentlichen Datenpunkt. Die Gerade hat immer einen gewissen In-Sample-Fehler, da sie nur selten direkt durch die Punkte geht. Eine Polynomfunktion hat im idealen Fall einen In-Sample-Fehler von 0. Errechnet man aber den Out-Of-Sample-Fehler, also den Fehler, der außerhalb der Datenpunkte auftritt, so weist die Gerade einen weitaus geringeren Fehler auf, als die Polynomfunktion, da eine Polynomfunktion zu starken Überschwingen neigt. Im Abbildung 4.1 erkennt man, dass die linke Lösung im Mittel den geringeren Fehler besitzt, die rechte Lösung aber enger an den eigentlichen Datenpunkten anliegt. Dadurch ergibt sich in Folge ein größerer Out-Of-Sample-Fehler. Das eigentliche Pro-

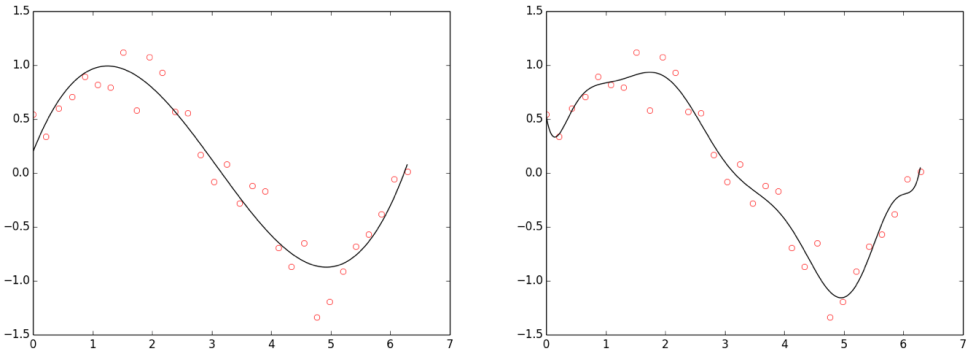


Abbildung 4.1: Good fit vs. Overfitting.
Quelle: www.tdda.info/generalized-overfitting-errors-of-applicability.html

blem bei Overfitting ist nicht, dass zu wenig Datensätze vorliegen. Meistens scheitert ein Modell an der Komplexität. In Kapitel 3.1.1 wurde bereits der Begriff *Occam's Rasiermesser* erwähnt. Occam's Rasiermesser besagt, dass die einfachste Lösung die wahrscheinlich passendste Lösung ist [23, S. 167]. Bei Entscheidungsbäumen gibt es das Pruning, welches die Bäume beschneidet und dadurch vereinfacht. Bei Regression gibt es den Begriff der *Regulierung*.

Bei der Regulierung wird ein zusätzlicher Wert Ω eingeführt. Ω beschreibt die Komplexität des mathematischen Modells. Anstatt wie im obigen Beispiel nur den In-Sample-Fehler zu reduzieren, wird bei der Regulierung der In-Sample-Fehler und Ω minimiert. Dadurch ist gewährleistet,

[13] In der Literatur als E_{in} bezeichnet
[14] In der Literatur als E_{out} bezeichnet

dass der ermittelte In-Sample-Fehler auch die geringste Komplexität aufweist. Geringere In-Sample-Fehler mit höherer Komplexität werden vom Algorithmus nicht als besser eingestuft. Eine Möglichkeit Ω zu berechnen ist mithilfe des *Weight Decay*. Dabei wird die Komplexität des Modells aufgrund der Anzahl der verwendeten Koeffizienten errechnet [23, S. 126].

Overfitting ist in der Tat ein sehr häufiges Problem, lässt sich aber durch die Wahl der richtigen Herangehensweise sehr gut eliminieren. Einige Ansätze neigen jedoch häufiger zu Overfitting, wie etwa das Boosting bei Entscheidungsbäumen. Bei Overfitting ist vor allem die Evaluierung enorm wichtig, da ohne genaue Evaluierung und ohne Visualisierung nur sehr schwer Overfitting überhaupt entdeckt werden kann. Durch den geringen In-Sample-Fehler kann es rasch übersehen werden, da fälschlicherweise angenommen wird, dass es sich um ein gutes Modell handelt.

Kapitel 5

Konzept

Zu Beginn stellt sich die Frage, wie man die verschiedenen API Anbieter am besten miteinander vergleichen kann. Natürlich gibt es mehrere Faktoren die bei einem Vergleich wichtig sind. Alle Faktoren werden genauer in Kapitel 7 beschrieben. Die wichtigsten Eigenschaften der API Anbieter sind Genauigkeit und Geschwindigkeit, jedoch müssen diese in Benchmarktests erhoben werden. Unter Genauigkeit versteht man wie präzise die Predictions sind. Der einfachste Weg die Präzision zu bestimmen ist, die Ergebnisse mit bereits erhobenen Ergebnissen zu vergleichen. Die Geschwindigkeit bezieht sich zum einen auf Trainingszeit für neue Modelle, sowie auf die durschnittliche Dauer von einzelnen Predictions.

Dabei ist es notwendig, dass alle Tests auf den gleichen Datensets ausgeführt werden. D.h., dass sowohl die Trainingsdaten und die Predictions für die Vergleiche aus dem selben Datenset stammen. Für die Tests werden dafür Datensets benötigt, welche im Grunde nur Trainingsdaten beinhalten müssen, da die Testdaten selbst erhoben werden können. Dafür müssen die Trainingsdaten so aufbereitet werden, dass alle API Anbieter damit umgehen können. Es werden 3 verschiedene Datensets für die Tests verwendet, was natürlich eine zu kleine Menge ist, um daraus einen zuverlässigen Vergleich zu erstellen. Die Ergebnisse sind zwar nur bedingt repräsentativ, aber der Vergleich wird genügend Aussagekraft über den jeweiligen Anbieter bieten.

Um die Tests durchzuführen wird dafür eine Java-Applikation entwickelt. Mit diesem Programm soll es möglich sein die verschiedenen APIs der Anbieter möglichst komfortabel miteinander zu vergleichen und die Ergebnisse in tabellarischer Form gegenüberzustellen. Aufgrund der Ergebnistabelle soll es möglich sein Erkenntnisse für die Evaluierung über die Trainingsdauer, die durschnittliche Prädiktionszeit und Korrektheit zu gewinnen.

5.1 Zielsetzung

Die Zielsetzung des Programmes ist folgende: Es soll mit dem Programm möglich sein, mehrere Machine Learning API Anbieter miteinander auf ihre wichtigsten Charakteristiken zu vergleichen. Die wichtigsten Charakteristiken sind Genauigkeit und Geschwindigkeit. Die anderen Faktoren – wie etwa Kosten, Konfigurierbarkeit, etc... – können ohne zusätzliches Programm verglichen werden. Folgende Kriterien muss das Programm erfüllen, um einen Vergleich der Anbieter anzustellen:

- Programmiersprache, die von Anbieter unterstützt wird
- Modularer Aufbau, um weitere Anbieter hinzuzufügen
- Einfache und für Anbieter einheitliche Oberfläche
- Grafisches Aufbereiten von Trainingsdaten
- Messen von Zeit und Genauigkeit

Als Programmiersprache wird Java gewählt. Sie zählt zu den beliebtesten Programmiersprachen und beinahe jeder Anbieter in Aufführung 3.3 stellt eine Java-Schnittstelle zur Verfügung. Außerdem hängt die Entscheidung ebenfalls mit den oben genannten Punkten 2, 3 und 4 zusammen, welche dann näher in Kapitel 5.2.1 beschrieben wird.

Punkt 5 „Messen von Zeit und Genauigkeit“ bedarf einer genaueren Erörterung. Während sich die Zeit für das Training und für jede einzelne Prädiktion noch leicht bestimmen lassen, so muss der Term *Genauigkeit* exakter erläutert werden. Um die Genauigkeit zu bestimmen benötigt man bereits klassifizierte Prädiktionen, sprich die Ergebnisse müssen schon vorliegen. Die Ergebnisse werden mit den ausgeführten Prädiktionen verglichen und je nach Übereinstimmung in korrekt oder inkorrekt eingeteilt. Die Ausgabe des Programmes sollte alle Informationen wie in Tabelle 5.1 beinhalten. Die

API provider name	Training	Avg. time	Correct	Incorrect	Percentage
Google	100s	400ms	20	80	20%
BigML	200s	300ms	40	60	40%
Prediction.io	300s	200ms	60	40	60%
Weka	400s	100ms	80	20	80%

Tabelle 5.1: Entwurf des Vergleichs in Tabellenform.

Tabelle beschreibt die oben erwähnten Eigenschaften der API Anbieter, die für eine weitere Evaluierung benötigt werden. Die eingefügten Daten sind nur als Platzhalter zu verstehen und haben keine weitere Bedeutung. Das wohl wichtigste Kriterium von den Anbietern ist die Genauigkeit. Deswegen wird der Genauigkeit auch die höchste Priorität zugemessen. Nichtsdestotrotz ist die Zeit auch ein wichtiges Kriterium. Genauigkeit und Zeit sollten sich in etwa die Waage halten.

Tabelle 5.1 dient als eine wichtige Grundlage für die Evaluierung. Der mobile Aspekt sollte aber nicht vernachlässigt werden. Und auch die grundsätzliche Frage sollte gestellt werden: Welche Anwendungsgebiete für Machine Learning gibt es auf mobilen Plattformen? Wann ist es sinnvoll diese direkt am Gerät zu implementieren und ist es überhaupt möglich? Grundsätzlich ist es schon möglich, aber die Internetanbindung der Geräte macht es meistens obsolet. Machine Learning benötigt Rechenleistung und Rechenleistung am mobilen Gerät geht zulasten der Akkulaufzeit. Die meisten API Anbieter stellen dazu ein Web-Service zur Verfügung. Es ist natürlich auch möglich mit seinen eigenen Webserver zu kommunizieren und von dort das API zu verwenden. Die Anwendungsbereiche für mobile Geräte sind vorallem Musikerkennung, Gesundheitsapps und unter anderem Empfehlungssysteme. Oftmals verrichtet der Server die Aufgabe und das Smartphone dient nur als Ausgabegerät. Hier wurde ebenfalls darauf geachtet, dass mit Java eine Programmiersprache zum Einsatz kommt, mit welcher ebenfalls das Machine Learning API am Server eingebunden werden kann.

Die verschiedenen API Anbieter gehen hier unterschiedliche Wege. Google und BigMl bieten ihre Lösung inklusive ihrer Infrastruktur an. Prediction.io lässt den Entwickler sogar einen eigenen Prediction-Server hosten und Weka führt Code lokal am Gerät aus. Für das zu entwickelnde Programm macht das aber keinen Unterschied. Es soll nämlich über ein Plugin-System möglich sein, die Schnittstelle soweit zu abstrahieren, dass die Applikation ohne Mühe weitere Anbieter unterstützen kann.

5.2 DPAT - Data Prediction API Analysis Tool

Im Rahmen des Konzeptes wurde die Applikation *Data Prediction API Analysis Tool* entwickelt, welches zur Evaluierung des Projektes benötigt wird. Das Programm implementiert alle genannten Anforderungen um die API Anbieter zu vergleichen. Der Fokus der Implementierung ist eine möglichst benutzerfreundliche Oberfläche und ein funktionierendes Plugin-System. Dem Plugin-System wird auch höchste Wichtigkeit zugemessen, weil es eine hervorragende Kapselung zwischen der Applikation und den eigentlichen APIs bietet. Dadurch können neue API Anbieter ohne weiteren Aufwand in das Programm hinzugefügt, aber ebensogut wieder entfernt werden. Die Plugins können aber auch von anderen Programmen verwendet werden, da die Schnittstellendefinition der Plugins offen liegt[15].

Es wird hier immer wieder das Wort *Plugin* verwendet. Wobei noch nicht gänzlich geklärt ist, um was es sich nun bei einem Plugin handelt und warum überhaupt ein Plugin-System verwendet werden soll. Grundsätzlich besteht ein Plugin aus einer Schnittstellendefinition (im Falle Java um ein

[15]Die Schnittstellendefinition wird in Kapitel 6 behandelt

Interface), die alle Informationen bezüglich eines Anbieters – die DPAT für die Einbindung benötigt – über Methoden zur Verfügung stellt. Die konkrete Implementierung hängt vom zugrundeliegenden API ab. Das Interface wird als Vertrag zwischen dem Hauptprogramm DPAT und dem Plugin (Anbieter-Abstrahierung) angesehen. Über dieses Plugin-System ist es möglich, ein Programm zu entwickeln, dass die konkrete API-Implementierung nicht kennt und zusätzlich noch skalierbar ist. Die Applikation selbst beinhaltet keine Machine-Learning-Logik, sondern bietet die Oberfläche um die Daten so zu konfigurieren, damit sie an die Logik in Form des Plugins weitergegeben werden kann.

5.2.1 Verwendete Technologien

Java

Die Applikation wird in Java implementiert. Das hat mehrere Gründe, wobei die beiden ausschlaggebendsten Gründe die Möglichkeit einer raschen GUI-Implementierung sowie die Verfügbarkeit von Machine Learning APIs waren. Eine weitere Alternative wäre C# gewesen, aber hier gab es das Problem, dass nicht alle Anbieter APIs für C# zur Verfügung stellten sind und sich C# nur auf Windows beschränkt. Dafür wäre ein ähnlich einfacher GUI-Entwurf wie mit Java möglich gewesen. Außerdem ist Java plattformunabhängig. Es gibt einige C++ Frameworks wie *Qt*, die ebenfalls plattformunabhängig sind und eine ebenso komfortable GUI-Entwicklung anbieten, aber keiner der erwähnten Anbieter macht sich die Mühe ein C++ API anzubieten. Durch die Plattformunabhängigkeit ist es auch möglich, dass Java am Server und am Client (inklusive Android) ausgeführt werden kann. Damit könnte der entwickelte Code ebenfalls mit gewissen Einschränkungen direkt auf einem Android-Smartphone ausgeführt werden.

Im Kapitel 5.1 wird ein modularer Aufbau als Ziel angestrebt. Ein modularer Aufbau erfolgt mithilfe von Plugins. Sozusagen ist jede API-Implementierung als ein einzelnes Plugin zu betrachten. In Java gibt es nun 3 Möglichkeiten ein Plugin-System zu realisieren. Eine Möglichkeit wäre das *Java Plugin Framework*, was aber relativ schlecht dokumentiert ist und seit 2007 nicht mehr weiterentwickelt wird[16]. Die zweite Möglichkeit wäre das Plugin-System von OSGi[17] zu verwenden, was wiederum äußerst komplex ist und auf Enterprise-Produkte abzielt. Die dritte und vermutlich eleganteste Möglichkeit ist ein Interface zu erzeugen, welches alle Plugins implementieren und diese dann zur Laufzeit über das Java Reflection API zu laden.

Aufgrund der überwiegenden Vorteile wurde Java als Programmiersprache für die Applikation gewählt.

[16] http://jpf.sourceforge.net/
[17] Open Services Gateway initiative

Externe Abhängigkeiten

Es wird darauf geachtet, dass die Applikation die Anzahl der Abhängigkeiten so gering wie möglich hält. Aufgrund des Plugin-Systems werden die einzelnen Plugins mitsamt deren gesamten benötigten Bibliotheken geladen, und somit sollen redundante Abhängigkeiten so gut es geht vermieden werden. Die Applikation benötigt nur 2 Bibliotheken:

- pdfbox
- json

PDFBox ist eine von Apache entwickelte Bibliothek, mit der es möglich ist PDF-Dateien zu erzeugen. Die ausgewerteten Predictions können als PDF exportiert werden. Und für Datenspeicherung und Datenaustausch wird die Standard-JSON-Bibliothek von org.json verwendet. Je nach Umfang von Plugin kann die Anzahl der benötigten Abhängigkeiten variieren. Eine genaue Auflistung ist in Kapitel 6.3.3 angeführt.

5.2.2 Programmaufbau

Das grundlegende Oberflächenkonzept wurde von dem Weka Toolkit inspiriert. Das Weka Toolkit selbst ist eine Java Applikation, die zusätzlich ein API zur Verfügung stellt. Das Programm ist in mehrere Tabs gegliedert, die den Arbeitsfluss bestimmen. Zugegebenermaßen ist die Software viel umfangreicher als Data Prediction API Analysis Tool. Doch vom Arbeitsfluss kann DPAT als eine vereinfachte Variante vom Weka Toolkit angesehen werden (siehe Abbildung 5.1).

Dabei wird die Oberfläche in DPAT in drei Tabs unterteilt:

- Daten aufbereiten
- Anbieter konfigurieren
- Verarbeiten

Im Tab *Daten aufbereiten* sollen die Trainingsdaten für die verschiedenen Plugins bereitgestellt werden. In *Anbieter konfigurieren* können die eingebundenen Plugins über eine Oberfläche initialisiert und konfiguriert werden und im Tab *Verarbeiten* kann die Verarbeitung der verschiedenen Ergebnisse eingestellt werden. Dabei ist wichtig anzumerken, dass es sich bei DPAT um keine Kopie von der Software Weka Toolkit handelt, sondern dass DPAT sich lediglich vom Oberflächenkonzept inspirieren hat lassen.

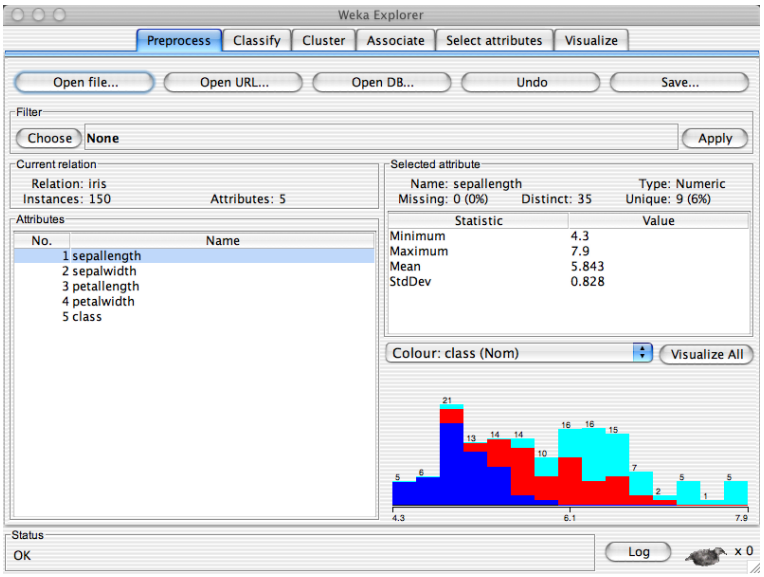


Abbildung 5.1: Weka Toolkit.
Quelle: http://weka.sourceforge.net/explorer_screenshots/PreprocessPanel.png

5.3 Verwendete Datensätze

Die Auswahl der verwendeten Datensätze wurde durch zwei Kriterien begrenzt. Aus Komplexitätsgründen wurden Datensätze mit weniger als zehn Attributen bevorzugt, da diese leichter zu analysieren und zu überprüfen sind. Der jedoch wichtigere Grund war, dass es sich bei allen Datensätzen um Klassifizierungen handeln muss, damit die verschiedenen Anbieter damit umgehen können. Unter anderem unterstützt BigML lediglich Entscheidungsbäume und ist somit für Clustering oder Regression nicht geeignet. **Als Quelle für die Datensets diente die freie Datenbibliothek von BigML**[18]. Einzige Voraussetzung ist, dass man bei BigML angemeldet sein muss um die Datensets unentgeltlich zu erwerben. Es wurden aus der Vielzahl an vorliegenden Daten drei Datensets ausgewählt, welche allesamt einen medizinischen Bezug aufweisen. In der Medizin gibt es zahlreiche Anwendungsfälle für Machine Learning. Für richtige medizinische Forschungszwecke besitzen die unten angeführten Datensätze aber keineswegs den benötigten Umfang.

Erkennung des Iris-Typs Ein eher einfach gestricktes Beispiel für einen medizinischen Anwendungsfall ist die Erkennung des Iris-Typs einer Person. Zu bestimmen gilt es zwischen *Iris setosa*, *Iris versicolor* und *Iris virginica*. Diese drei Typen bilden die Klasse des Datensets. Die vier beschreibenden

[18] https://bigml.com/gallery/datasets

Attribute sind die sepale und petale Länge und Breite. Aus diesen vier Attributen wird der Iris-Typ bestimmt. Das Datenset besitzt 150 Einträge.

Erkennung einer Diabeteserkrankung Die Erkennung von Diabetes benötigt im Vergleich zum ersten Beispiel mehrere beschreibende Attribute, unter anderem Alter, BMI, Insulinspiegel, Plasma Gluckose, Blutdruck und die Diabetes Familienhistorie. Die bestimmende Klasse gibt Aufschluss darüber, ob die Person an Diabetes erkrankt ist oder nicht. Bei der Klasse handelt es sich in diesem Fall um einen binären Wert: wahr oder falsch. In sehr vielen Anwendungsgebieten reicht ein binärer Wert als Klasse aus. Mit über 750 Einträgen ist das Datenset auch weitaus umfangreich als noch die Iris-Typerkennung.

Erkennung einer Brustkrebswiedererkrankung Wiederum ein medizinisches Beispiel, das als zu bestimmende Klasse einen binären Wert besitzt. Mithilfe des Datensets soll erkannt werden, ob eine Frau, die bereits an Brustkrebs erkrankt war, ein Risiko besitzt wieder daran zu erkranken. Interessant dabei ist, dass es sich um ein sehr eingeschränktes Anwendungsgebiet handelt, da es nur für Personen zutrifft, die bereits an Brustkrebs erkrankt waren. Das Datenset beinhaltet neun beschreibende Attribute und einen Umfang von mehr als 250 Einträgen. Zu den beschreibenden Attributen zählen u.a. Alter, Menopause, Tumorgröße, Lymphknotenbefall und die befallene Brust.

5.4 Verwendete API Anbieter

Die bestehenden API Anbieter wurden in Kapitel 3.3 bereits aufgelistet. Aufgrund der verschiedenen beschriebenen Anforderungen wurden 4 API Anbieter ausgewählt, die als Plugin gekapselt für DPAT implementiert werden sollen. Der große Nachteil jedes Plugin-Systems ist der damit einhergehende Kompromiss der Vereinheitlichung. Die Schnittstellendefinition muss abstrakt gehalten werden, um die Kompatibilität zu den einzelnen Plugins sicherzustellen. Modelltraining und die eigentliche Prädiktion sind hier die beiden Hauptprobleme. Es gibt kein einheitliches Datenformat dafür, mit welchem alle APIs kompatibel sind. Das Programm DPAT muss hier einen Kompromiss wählen um die Daten so aufzubereiten, dass alle Plugins damit arbeiten können.

5.4.1 Google Prediction API

Als erster Anbieter wurde das Prediction API von Google ausgewählt. Ähnlich wie Amazon bietet Google eine mächtige und umfangreiche Cloud-Plattform mit einer Vielzahl von Diensten an. In der Regel sind die Produkte qualitativ hochwertig und gut dokumentiert. Außerdem gewährleistet

Google, dass die Services auch auf mobilen Betriebssystemen wie Android und iOS performant arbeiten. Auf der Homepage gibt es einige Sample-Projekte, auf welche man mit dem API zugreifen kann. Unter anderem gibt es bereits ein Modell, mit dessen Hilfe man Sprachen erkennen kann, was als hervorragender Anwendungsfall für mobile Geräte gesehen werden kann.

5.4.2 BigML

Durch die Vorabsondierung der verschiedenen Anbieter stach BigML als das wohl beste kommerzielle Produkt hervor. Wenig verwunderlich, da sich die Firma ausschließlich auf Machine Learning spezialisiert hat. Die Weboberfläche ist intuitiv gehalten und es werden reichlich Features angeboten. Nun wird überprüft, ob auch die Qualität entsprechend dem ersten Eindruck stimmt.

5.4.3 Prediction.io

Die Lösung Prediction.io überzeugt vor allem durch das Argument, dass es möglich ist seinen eigenen Prediction-Server zu hosten und nicht von der Infrastruktur von anderen Anbietern abhängig ist. Auch wenn der Trend vermehrt in Richtung von cloud-basierten Lösungen geht, so ist es in manchen Fällen äußerst nützlich die Infrastruktur selbst zur Verfügung zu stellen und dafür volle Kontrolle über das System und die eigenen Daten zu behalten.

5.4.4 WEKA Toolkit

Als im Konzept Java als Programmiersprache festegelegt wurde, war die Einbindung von Weka naheliegend. Immerhin handelt es sich um ein Standardwerkzeug für die Forschung im Bereich Machine Learning[6]. Der interessante Aspekt dabei ist sicherlich, dass dank der Java-Schnittstelle auch eine Implementierung für Android möglich wäre. Damit könnte ein Smartphone die Machine Learning Algorithmen direkt am Gerät ausführen. Außerdem ist es möglich Weka im Backend mit Java zu implementieren und so über einen eigenen Server zugänglich zu machen.

5.5 Vergleichsansatz

Die wichtigsten Vergleichskriterien sind Geschwindigkeit und Genauigkeit. Die Genauigkeit ist dabei das Hauptkriterium. Die Geschwindigkeit ist aber nicht minder wichtig. Beispielsweise wenn ein API Anbieter um 3% genauere Ergebnisse liefert, aber das fünffache der Zeit für das Ergebnis benötigt, kann es nicht pauschal als besser klassifiziert werden. Die exakte Beschreibung des Evaluierungsprozesses erfolgt später in Kapitel 7. Der Gesamtvergleich muss die einzelnen Kriterien miteinander kombinieren und darf diese

nicht separat betrachten.

Im Kontext des Konzeptes und der dazu benötigten Evaluierungssoftware DPAT bezieht sich der Vergleichsansatz jedoch ausschließlich auf Geschwindigkeit und Genauigkeit. Die anderen Kriterien lassen sich nicht in Tests miteinander vergleichen. Die Geschwindigkeitsmessung erfolgt über Zeitmessung für und nach der Prädiktion sowie vor und nach dem Training. Die Genauigkeitsmessung benötigt bereits klassifizierte Daten um diese mit den Ergebnissen zu vergleichen. Beide Messungen können zeitgleich erfolgen. Zusätzlich ist es möglich, dass die Zeitmessung nur alleine erfolgt. Die Ergebnisse sind für ein Modell konstant. Ein trainiertes Modell wird für diesselben Eingangsgrößen das gleiche Ergebnis zurückliefern, die Laufzeiten können aber stärker variieren. Mit mehreren Zeitmessungen kann ein exakterer Mittelwert errechnet werden.

Kapitel 6

Implementierung

In diesem Kapitel wird die Implementierung des in Kapitel 5 erwähnten Konzepts beschrieben. Konkret geht es um die Implementierung des Programms Data Prediction API Analysis Tool, das mittels Java realisiert wird. Die Benutzeroberfläche wurde mithilfe der Bibliothek *Swing* erstellt. Das Hauptaugenmerk liegt in diesem Kapitel auf der Benutzeroberfläche und dem bereits mehrmals erwähnten Plugin-System.

6.1 Beschreibung

Das Programm ist nicht konzipiert, dass es von ungeschulten Benutzern verwendet werden kann. Eine Grundkenntnis im Bereich von Machine Learning und dem Umgang mit den verschiedenen Anbietern wird vorausgesetzt. Nichtsdestotrotz wurde auf eine intuitive und benutzerfreundliche Oberfläche Wert gelegt. Dadurch, dass Java betriebssystemunabhängig ist, ergibt sich dadurch zwangsläufig ein Problem mit den GUI-Elementen. Standardmäßig wird der Look *Nimbus* verwendet, der aber eine sterile Oberfläche bietet und sich nicht den anderen Programmen des Betriebssystems anpasst. Um diesen Problem entgegenzuwirken bietet Java die Möglichkeit, den nativen Look des Betriebssystems zu verwenden.

```
// Set native look and feel
UIManager.setLookAndFeel(UIManager
            .getSystemLookAndFeelClassName());
```

Listing 6.1: Ändern von Look-and-Feel in Java.

Das Code-Snippet wird zu Beginn des Programms in der *main*-Methode von *MainWindow* aufgerufen.

Das Programm ist in mehrere Packages aufgeteilt. Das Root-Package, in welchem sich die sechs Packages befinden lautet *at.fhooe.mc.dpat*.

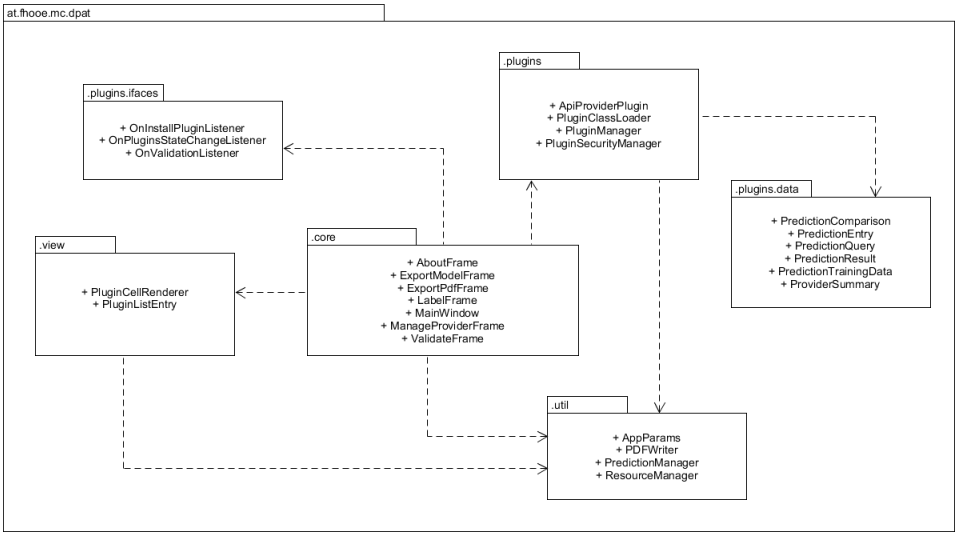


Abbildung 6.1: Package-Diagramm der Applikation.

.core Hier befinden sich alle GUI-Elemente, also alle Subklassen, die vom Typ *JFrame* ableiten. Einstiegspunkt für das Programm ist die Klasse *MainWindow*.

.plugins Mitunter das wichtigste Package des Plugin-Systems. Während im Package *.plugins.data* die Hilfsklassen für die Plugins implementiert werden, beinhaltet dieses Package alle Klassen, welche das Plugin-System überhaupt ermöglichen. *PluginManager, PluginSecurityManager* und *PluginClassLoader* ermöglichen dem Programm das Laden und Einbinden von Plugins aus JAR-Dateien. Zusätzlich befindet sich das Interface *ApiProviderPlugin* in diesem Package. Dieses Interface wird in Kapitel 6.3.2 genauer beschrieben.

.plugins.data Das Package ist eine der Kernkomponenten des Plugin-Systems. Die hier befindlichen Klassen sind alle Hilfsklassen der Plugins. Das eigentliche Herzstück, die Klasse *ApiProviderPlugin* befindet sich jedoch im Package *.plugins*. Dieses Package, sowie die Klasse *ApiProviderPlugin* müssen als JAR-Datei exportiert werden, um von den Plugins verwendet werden zu können.

.plugins.ifaces Die hier definierten Interfaces dienen nur als Callback für pluginspezifische Aktivitäten, wie etwa für die Plugin-Installation oder für Plugin-Statusänderungen.

.util Wie der Name vermuten lässt befinden sich hier Utility-Klassen. Programmweite Paramater werden in der Klasse *AppParams* über statische Variablen zugänglich gemacht. Des Weiteren befindet sich der *PDFWriter* und weitere Manager-Klassen in diesem Package.

.view In diesem Package befinden sich Hilfsklassen für GUI-Elemente. Das Programm verwendet eigene Renderer für *JList* Listeneinträge.

6.2 Benutzeroberfläche

Generell wurde der Arbeitsablauf des Programms an die Implementierung des Weka Toolkits angelehnt. Zur Navigation gibt es mehrere Tabs, die voneinander abhängig sind. Jeder Tab geht davon aus, dass der vorherige Tab bereits richtig konfiguriert ist. Um das Programm benutzerfreundlicher zu gestalten, wurden vermehrt Icons für Tabs, Buttons und Menüeinträge verwendet. Wichtige Elemente sind über Shortcuts im Menü zugänglich. Im Menü befinden sich unter anderem der PDF-Export, Prediction-Model-Export, der Start von Predictions und die Verwaltung von API Anbieter.

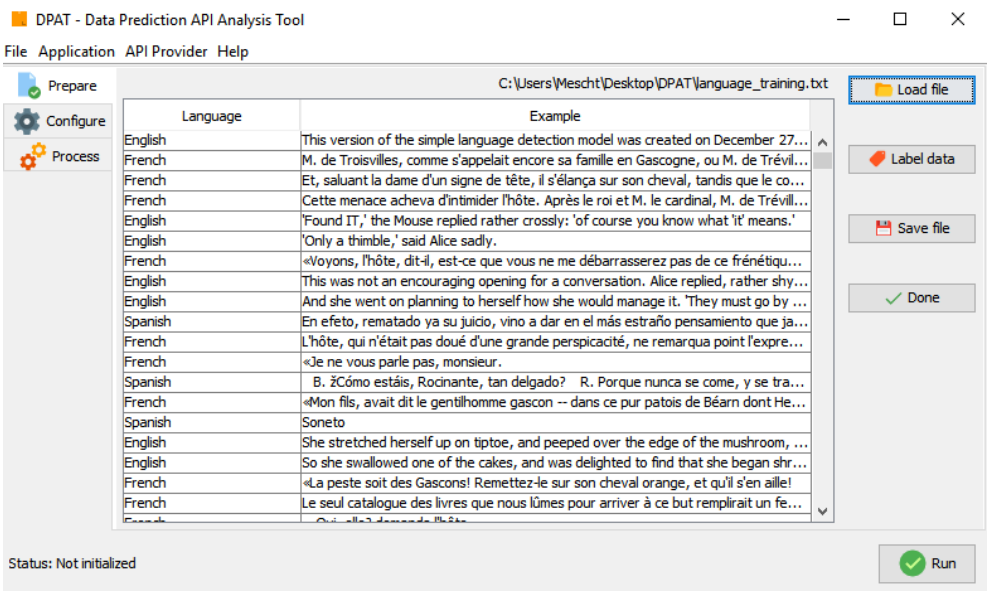


Abbildung 6.2: Daten aufbereiten.

Beim Programmstart erscheint der Tab für das Aufbereiten der Trainingsdaten. Dieser Tab kann übersprungen werden, wenn bereits trainierte Modelle verfügbar sind. Im Tab befinden sich auf der rechten Seite vier Buttons: *Laden, Label, Speichern* und *Fertig*. Es können Trainingsdaten als CSV-, TXT- oder ARFF-Datei geladen werden. Über den Button *Label* wird

ein eigenes Fenster geöffnet, in welchem eine komfortablere Oberfläche zur Bearbeitung der Daten zur Verfügung steht. Sobald die Daten aufbereitet sind, muss explizit der Button *Fertig* gedrückt werden, damit sich der Status des Programms ändert und somit die API Anbieter konfiguriert werden können. Die Plugins müssen aber bereits zu Beginn installiert sein, können aber jederzeit ohne Neustart nachgeladen werden. Neue API Anbieter können über den Menüeintrag *Provider verwalten* (oder mit dem Shortcut Strg+M) installiert werden.

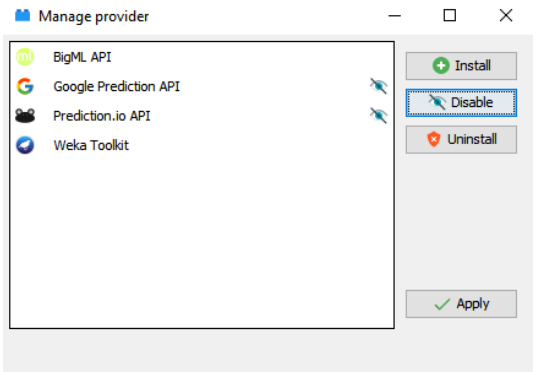


Abbildung 6.3: Provider verwalten.

Das Fenster bietet nicht nur die Funktionalität neue Plugins zu installieren. Es ist auch möglich Plugins zu deinstallieren und Plugins vorübergehend auszuschalten. Der Button *Übernehmen* führt die getätigten Änderungen durch und schließt das Fenster. Bei der Installation werden alle externen Bibliotheken in einen gemeinsamen Ordner kopiert. Dies kann bei größeren Bibliotheken einige Sekunden dauern, daher erscheint während der Installation ein Fortschrittsbalken am unteren Ende des Fensters. Bei der Deinstallation werden alle zuvor hinzugefügten Bibliotheken wieder entfernt.

Die Konfiguration erfolgt für jedes Plugin einzeln. Wie in Abbildung 6.4 gezeigt, müssen die Plugins zuerst ausgewählt werden. Dadurch ist möglich, für einzelne Predictions nur gewisse Anbieter zu verwenden. Die einzelnen Plugins bieten alle eine Benutzeroberfläche für die Konfiguration an. Diese werden im Plugin in einem *JPanel* implementiert. Die grundsätzlichen Funktionalitäten der Plugin-Oberfläche sind das Authorisieren des APIs und das Auswählen sowie das Erzeugen neuer Modelle. Einige Anbieter bieten schon vordefinierte Modelle. In diesem Fall ist es nicht notwendig das Modell zu trainieren. Für untrainierte Modelle gibt es den Button *Modelle trainieren.* Damit werden alle ausgewählten Modelle mit den aufbereiteten Trainingsdaten trainiert. Da die Trainingszeit variieren kann und es nicht möglich ist, den Status des Modells abzufragen, muss gewartet werden, bis alle Modelle fertig trainiert sind.

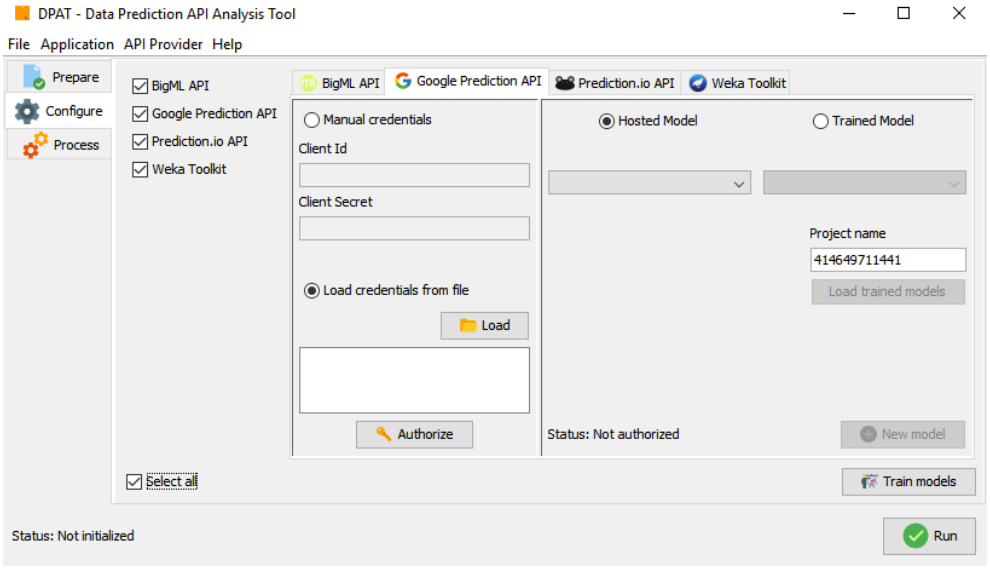


Abbildung 6.4: Provider konfigurieren.

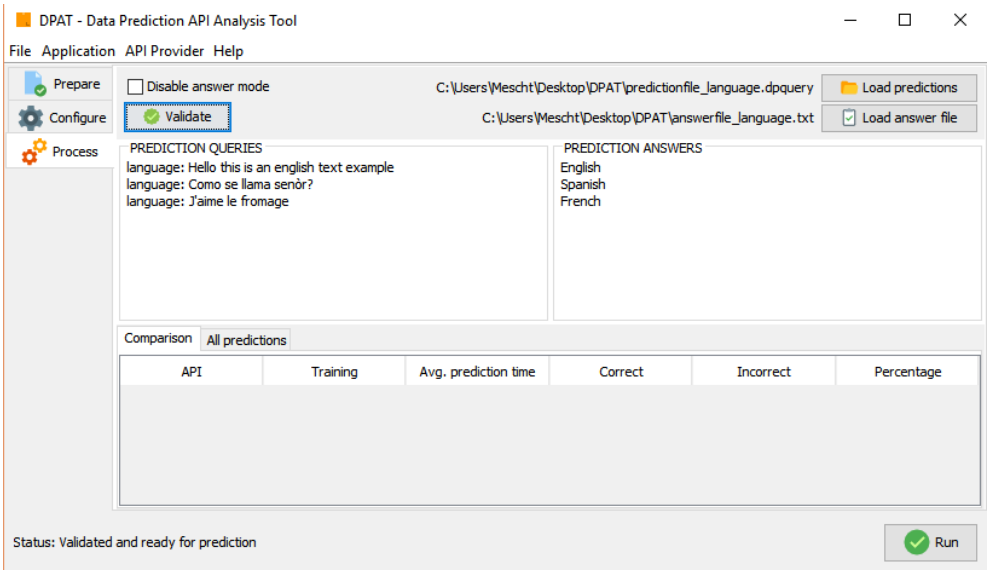


Abbildung 6.5: Daten verarbeiten.

Für die Prediction-Daten wurde das Dateiformat *.dpquery* eingeführt[19]. Im Vordergrund steht die Richtigkeit der Daten. Das Programm unterstützt nur .dpquery-Dateien als Eingabe, damit diese auch richtig eingelesen und verarbeitet werden können. Alternativ können die Antworten auf die Predic-

[19] Jede Zeile beinhaltet einen Datensatz im JSON-Format

tions ebenfalls über eine Datei geladen werden. Falls dies nicht erwünscht ist, kann der Antwortenmodus auch ausgeschalten werden. Die Daten können auch per Hand eingegeben werden. Bevor nun aber Predictions ausgeführt werden, müssen alle Daten zuerst validiert werden. Das stellt den letzten Schritt vor den eigentlichen Predictions dar.

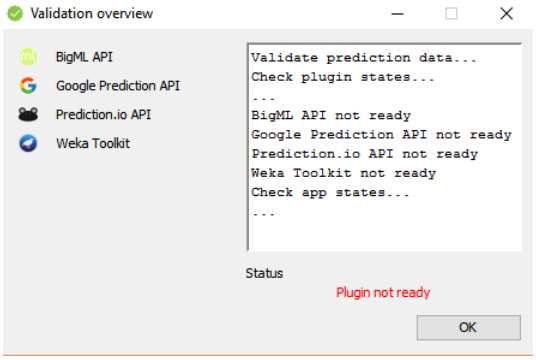


Abbildung 6.6: Fehlgeschlagene Validierung.

Die Validierung stellt sicher, dass alle Daten richtig eingegeben wurden und die Plugins richtig konfiguriert sind. Auf der linken Seite werden die ausgewählten Plugins angezeigt, auf der rechten Seite die getätigten Abläufe. Zu Beginn werden die Predictions überprüft. Im Anschluss werden die Plugins überprüft, ob diese schon initialisiert sind. Wie in Abbildung 6.6 angezeigt sind die Plugins nicht konfiguriert, wodurch der Validierungsprozess abgebrochen wird. Solange der Validierungsprozess nicht erfolgreich abgeschlossen ist, ist es nicht möglich Predictions auszuführen.

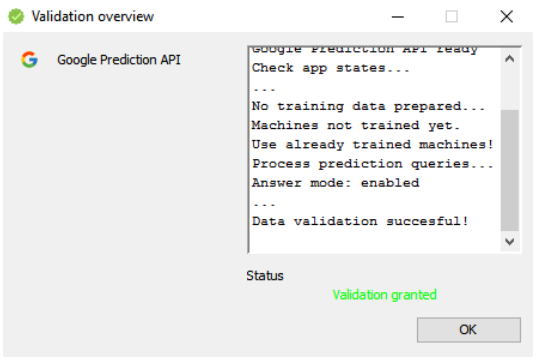


Abbildung 6.7: Erfolgreiche Validierung.

Wenn die Plugins bereits konfiguriert sind, werden die Trainingsdaten überprüft und der Trainingsstatus der Modelle abgefragt. Dieser Schritt kann übersprüngen werden, wenn bereits trainierte Modelle verwendet wer-

den. Im letzten Schritt werden noch die *Prediction Queries* mit den *Prediction Antworten* abgeglichen. Sollte ein Schritt fehlschlagen, schlägt der gesamte Validierungsprozess fehl.

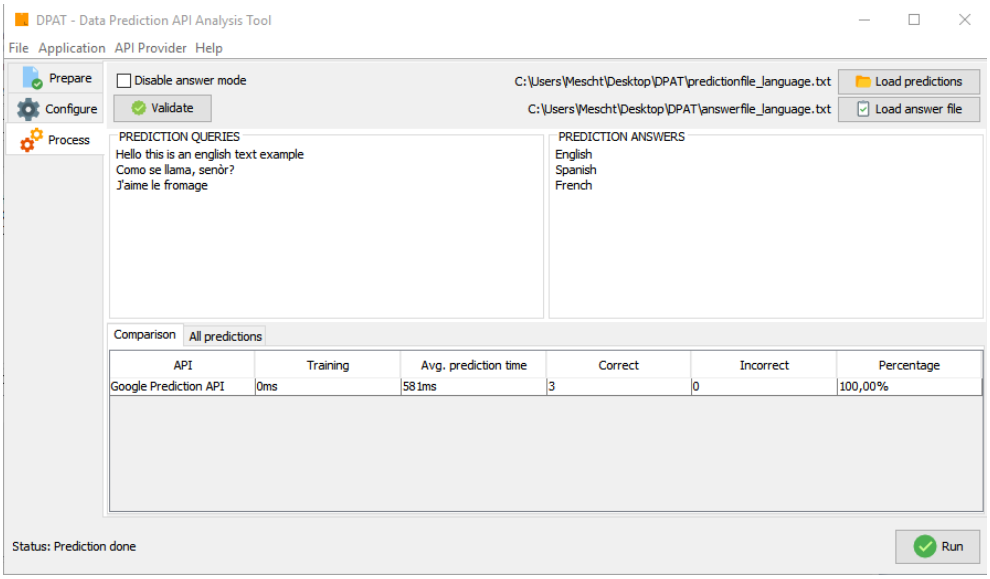


Abbildung 6.8: Predictions ausgeführt.

Bei erfolgreich Validierung können die eigentlichen Predictions ausgeführt werden. Die Predictions werden in der Tabelle nach Anbietern zusammengefasst. Diese Tabelle wird unter anderem als Ausgangspunkt für die Evaluierung in Kapitel 7 verwendet. Im zweiten Tab werden alle ausgeführten Predictions aufgelistet.

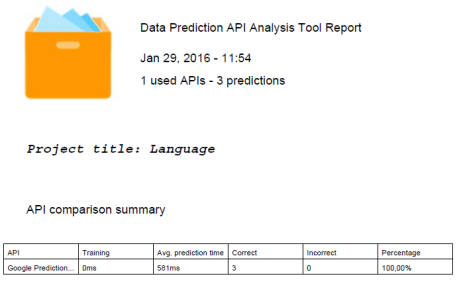


Abbildung 6.9: Ergebnisse exportieren.

DPAT bietet zwei zusätzliche Features für den Export der Ergebnisse. Zum einen ist es möglich, dass Prediction Model als PMML-Datei zu exportieren (nur von BigML unterstützt). Zum anderen können die ausgeführten Predictions und die Zusammenfassung als PDF-Datei exportiert werden.

6.3 Das Plugin-System

Schon von Beginn des Projektes an wurde großer Wert auf ein modulares, erweiterbares System gelegt. Dabei sollte die Schnittstelle auch möglichst einfach und intuitiv gehalten werden. Ausgangspunkt des Plugin Frameworks ist das Interface *ApiProviderPlugin*. Grundsätzlich ist die Struktur sehr simpel gehalten. Es gibt keinerlei Vererbung. Die abgebildeten Klassen werden ausschließlich vom Interface verwendet. Die weiteren Klassen des Frameworks sind: *PredictionQuery, PredictionTrainingData, PredictionEntry, PredictionResult, PredictionComparison* und *PredictionSummary*. Die Klassen *PredictionResult, PredictionComparison* und *PredictionSummary* werden verwendet um die Resultate der Prediction zu verarbeiten und visuell aufzubereiten.

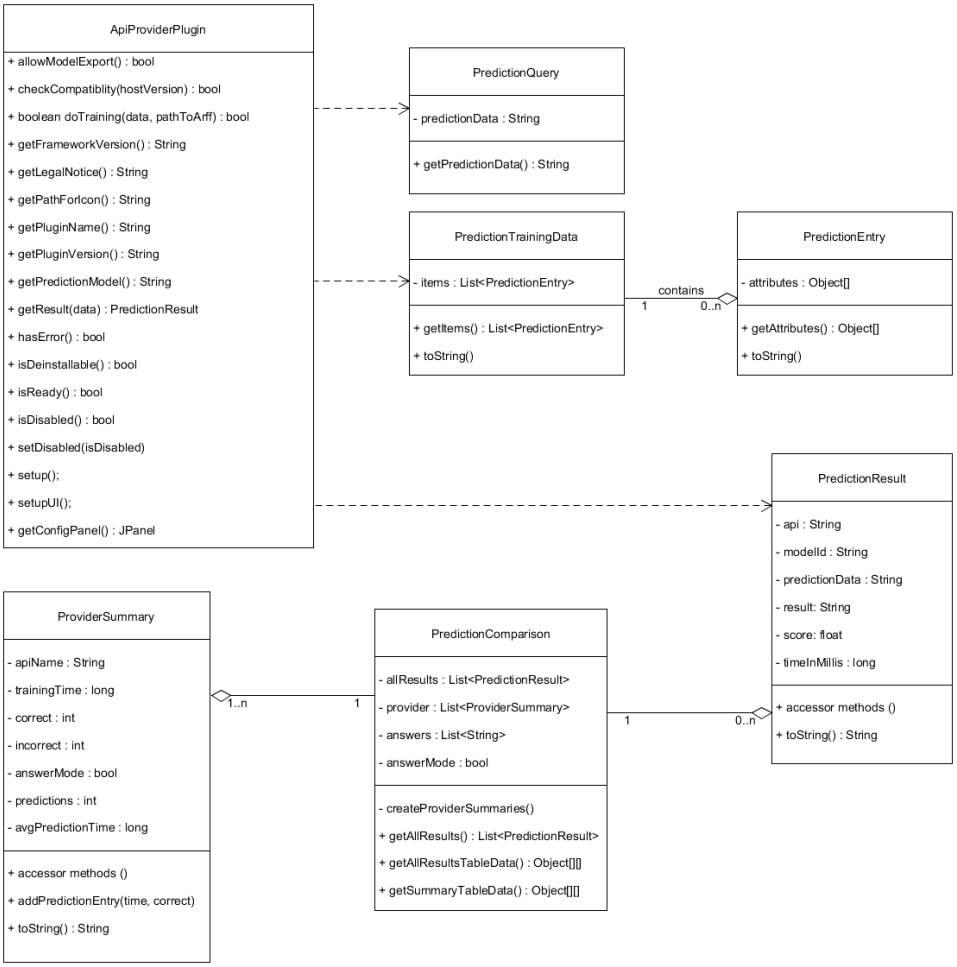


Abbildung 6.10: Klassendiagramm des Plugin-Systems.

Die Klasse *ApiProviderPlugin* sollte möglichst generisch gehalten werden, um die verschiedensten Trainingsdaten verarbeiten zu können. Die Generizität ist mitunter das Hauptproblem des Projekts. Deswegen wird bei der Schnittstelle vermehrt auf das JSON-Format gesetzt, um die Daten möglichst generisch weiterzureichen. Dies geht allerdings zu Lasten der Komplexität der Hauptanwendung, da die Daten zuallererst in das passende Format konvertiert werden müssen.

6.3.1 Plugin-Struktur

Um ein Plugin für DPAT zu schreiben wird die Pluginbibliothek *dpatp-x.y.jar*(Data Prediction API Analysis Tool Plugin) benötigt. Diese Bibliothek beinhaltet alle in Abbildung 6.10 abgebildeten Klassen. Im Quellordner muss sich eine Klasse befinden, welche das Interface *ApiProviderPlugin* implementiert. Ansonsten wird das exportierte Plugin vom Hauptprogramm nicht als Plugin erkannt und kann infolgedessen nicht eingebunden werden. Die Konfiguration des Plugins kann über eine Subklasse von *JPanel* visualisiert werden. Es wird empfohlen, dass sich diese Klasse ebenfalls im Quellordner des Plugins befindet. Außerdem ist es wichtig, dass sich alle Klassen im default-package befinden. Plugin-Packages führen zu Fehlern beim Einbinden in das Hauptprogramm.

Daneben werden noch drei zusätzliche Ordner auf Ebene des Quellordners benötigt, damit das Plugin einwandfrei funktioniert.

/images In diesem Ordner befinden sich alle vom Plugin verwendeten Bilder wie Plugin Icon, Button-Icons, etc...

/libs Alle Bibliotheken, welche das Plugin benötigt, werden in diesen Ordner abgelegt. Bei der Installation wird der gesamte Ordnerinhalt in den Bibliothekenordner des Hauptprogramms kopiert. Zur Laufzeit werden alle Abhängigkeiten der Plugins geladen. In diesem Ordner befindet sich ebenfalls die Bibliothek *dpatp-x.y.jar*, wird aber bei der Installation des Plugins nicht kopiert. Alle benötigten Klassen befinden sich bereits im Hauptprogramm.

/res Der Ordner kann für diverse Ressourcen verwendet werden. Hier werden unter anderem Prediction Models oder Client Secrets abgelegt. Hauptsächlich wird dieser Ordner aber zur Identifikation der Pluginnamen zur Installationszeit benötigt. Um eine saubere Installation zu gewährleisten, muss jedes Plugin eine Datei namens *Pluginname.dpat* im /res-Ordner abgelegen. Bei der Installation werden alle benötigten Bibliotheken aus dem /libs-Ordner für das jeweilige Plugin registriert. Bei einer Deinstallation des Plugins können so auch alle Bibliotheken entfernt werden.

6.3.2 Das ApiProviderPlugin Interface

Das Kernstück des Pluginsystems ist das bereits erwähnte Interface *ApiProviderPlugin*. Das Interface bietet alle benötigten Methoden um die Plugins ins System zu integrieren. Ein Großteil der Methoden wird für die korrekte und ansprechende visuelle Darstellung benötigt. Insgesamt stehen 19 Methoden zur Verfügung, wobei lediglich sechs Methoden für die Verwendung essentiell sind. Mittels *checkCompatibilityVersion()* wird zuallererst überprüft, ob das Plugin mit der DPAT-Version kompatibel ist. Die Methoden *setup(), setupUI()* sowie *getConfigPanel()* werden benötigt um die grundsätzliche Funktionalität des User Interfaces des Plugins aufzusetzen. Die zwei wichtigsten Methoden sind *doTraining()* und *getResult()*, also das Prediction Model trainieren und die Predictions auszuführen. Wie bereits oben erwähnt ist es das Hauptproblem diese beiden Methoden so generisch wie möglich zu halten.

```
import javax.swing.JPanel;
import org.json.JSONObject;

public interface ApiProviderPlugin {
  public boolean allowModelExport();
  public boolean checkCompatiblity(String hostVersion);
  public void createModel(String name);
  public boolean doTraining(PredictionTrainingData data,
                String pathToArff);
  public String getFrameworkVersion();
  public String getLegalNotice();
  public String getPathForIcon();
  public String getPluginName();
  public String getPluginVersion();
  public String getPredictionModel();
  public PredictionResult getResult(JSONObject data);
  public boolean hasError();
  public boolean isDeinstallable();
  public boolean isReady();
  public boolean isDisabled();
  public void setDisabled(boolean isDisabled);
  public void setup();
  public void setupUI();
  public JPanel getConfigPanel();
  @Override boolean equals(Object other);
}
```

Listing 6.2: Das Interface ApiProviderPlugin.

Neben all diesen Einschränkungen dürfen die Klassen, die das Interface implementieren, nur parameterlose Konstruktoren aufweisen. Die Plugins werden über das Java Reflection API geladen und dabei wird der Default-Konstruktor aufgerufen. Die Instanziierung der Plugins erfolgt über die Methode *Class.newInstance()*.

```
public void loadPlugins() {

   String[] files = mPluginsDirectory.list();
   for (int i = 0; i < files.length; i++) {
    try {

    if (!files[i].endsWith(".class")) {
      continue;
    }
    Class<?> c = cl.loadClass(files[i].substring(0,
   files[i].indexOf(".")));
    List<Class<?>> intfs =
   Arrays.asList(c.getInterfaces());
    for (int j = 0; j < intfs.size(); j++) {
      if (intfs.get(j).getName()
      .equals(ApiProviderPlugin.class.getName())) {
        ApiProviderPlugin pp = (ApiProviderPlugin)
                c.newInstance();
        mPlugins.add(pp);
        continue;
        }
      }
    } catch (Exception e) {
      System.err.println("File " + files[i] + " does not
   contain a valid ApiProviderPlugin implementation!");
    }
  }
}
```

Listing 6.3: Die Methode loadPlugins() in PluginManager.

In Listing 6.4 werden die wichtigsten Methoden der Implementierung des BigML Plugins abgebildet. Die Plugins sind zu umfangreich um sie in voller Länge abzubilden, weswegen auf die beiliegende CD verwiesen wird, welche die vollumfassende Implementierung enthält. Die Methode *doTraining()* wurde in der Implementierungsphase durch die Methode *createModel()* abgelöst, da die Trainingsdaten meist schon beim Erzeugen eines neuen Modells benötigt werden.

```
public class DPATBigMLPlugin implements ApiProviderPlugin {

  private BigMLClient mClient;
  @Override
  public void setup() {
    try {
      mClient =
    BigMLClient.getInstance(mConfigPanel.getUsername(),

    mConfigPanel.getApiKey(), true);
      retrieveAvailableModels();
      mIsReady = true; mHasError = false; mIsDisabled =
    false;
    } catch (AuthenticationException e) {
      e.printStackTrace();
      mIsReady = false; mHasError = true; mIsDisabled =
    true;
    }
  }
  private void retrieveAvailableModels() {
    mModels = new ArrayList<BigMLModel>();
    mModelNames = new ArrayList<String>();
    // Get available models from server
    JSONObject object = mClient.listModels(null);
    JSONArray remoteModels = (JSONArray)
    object.get("objects");
    for (int i = 0; i < remoteModels.size(); i++) {
      // Get name and modelId from JSON
      String name = (String) ((JSONObject)
    remoteModels.get(i)).get("name");
      String modelId = (String) ((JSONObject)
    remoteModels.get(i)).get("resource");
      mModels.add(new BigMLModel(name, modelId));
      mModelNames.add(name);
    }
    mConfigPanel.setModels(mModelNames);
  }
  @Override
  public PredictionResult getResult(org.json.JSONObject
    query) {
    BigMLModel selectedModel =
    mModels.get(mConfigPanel.getSelectedModelIndex());
    // Convert from org.json to org.json.simple object
    JSONObject inputData = convertToSimpleJSON(query);
```

```
    try {
      // Use local predictive model
      JSONObject model =
    mClient.getModel(selectedModel.getModelId());
      LocalPredictiveModel lpm = new
    LocalPredictiveModel((JSONObject)
    (model.get("object")));
      long start = System.currentTimeMillis();
      Prediction localPrediction = lpm.predict(inputData,
    true);
      long time = System.currentTimeMillis() - start;
      return new PredictionResult(getPluginName(),
    selectedModel.getModelId(),
          query.toString(), (String)
    localPrediction.getPrediction(),
          localPrediction.getConfidence().floatValue(),
    time);
    } catch (Exception e) {
      e.printStackTrace();
    }
    return null;
  }
  public void createModel(String name) {
    JFileChooser chooser = new JFileChooser("Pick data
    source");
    chooser.setFileFilter(new
    FileNameExtensionFilter(".csv, .txt, .arff files"));
    chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
    int approval = chooser.showOpenDialog(mConfigPanel);
    if (approval == JFileChooser.APPROVE_OPTION) {
      String filepath =
    chooser.getSelectedFile().getAbsolutePath();
      long startTime = System.currentTimeMillis();
      // Create the data source
      JSONObject datasource =
    mClient.createSource(filepath, name, (JSONObject) null);
      // Get resourceId
      String sourceId = (String)
    datasource.get("resource");
      // Create dataset
      JSONObject dataset = mClient.createDataset(sourceId,
    (JSONObject) null, null, null);
      String datasetId = (String) dataset.get("resource");
      // Create model
```

```
        JSONObject model = mClient.createModel(datasetId,
      (JSONObject) null, null, null);
        String modelId = (String) model.get("resource");
        long trainingTime = System.currentTimeMillis() -
      startTime;
       }
    }
  }
```

Listing 6.4: Implementierung des BigML Plugins.

6.3.3 Verwaltung von externen Abhängigkeiten

Die Plugins sind so konzipiert, dass sie autonom arbeiten und in jede beliebige Anwendung integriert werden können. Kein Plugin besitzt eine Abhängigkeit zur Hauptanwendung, da alle Abhängigkeiten in den dafür vorgesehenen Ordner abgelegt sind. Das Hauptprogramm lädt zur Laufzeit lediglich die Plugin-Abhängigkeiten in den *Classpath*.

```
try {

  Method method =
    URLClassLoader.class.getDeclaredMethod("addURL", new
    Class[] { URL.class });
  method.setAccessible(true);
  method.invoke(ClassLoader.getSystemClassLoader(), new
    Object[] { file.toURI().toURL() });

} catch (Exception e) {
  return false;
}
```

Listing 6.5: Laden von externen Abhängigkeiten.

Bei der Installation werden alle externen Abhängigkeiten des Plugins vom Ordner /libs in den dafür vorgesehenen Zielordner /plugins/libs der Hauptanwendung kopiert. Bei jedem Programmstart werden alle Dateien in diesem Ordner über Reflection (siehe 6.5) in den Classpath geladen.

Die Plugins beinhalten verschiedene Abhängigkeiten und es ist möglich, dass mehrere Plugins diesselbe Abhängigkeit besitzen. Die Plugins sollen aber unabhängig voneinander arbeiten. Deswegen ist es nötig, dass die Plugins alle Abhängigkeiten beim Exportieren mitliefern. Alle Plugins sind neben den unten angeführten Abhängigkeiten ebenfalls von der eigentlichen Plugin-Bibliothek *dpatp.jar* abhängig.

Data Prediction API Analysis Tool

- PdfBox 1.8.10
- org.json Json Library

Google Prediction API

- org.json Json Library
- google-api-services-prediction 1.6 rev52
- commons-logging 1.1.1
- Diverse Google Client APIs
- Gson 2.1
- httpclient 4.0.1
- httpcore 4.0.1
- jackson-core 2.1.3
- jdo2 2.3
- jetty 6.1.26
- jetty-util 6.1.26
- jsr305 1.3.9
- transaction-api 1.1
- javax.servlet API 3.1.0

BigML

- bigml-binding 1.4.1
- json-simple 1.1.1
- org.json Json Library
- slf4j 1.7.14

WEKA Toolkit

- Weka API 3.7
- org.json Json Library

Prediction.io

- Prediction.io Client 0.9.5
- async-http-client 1.8.16
- Gson 2.1
- Guava 18
- Netty 3.6.0
- slf4j 1.7.14
- Joda Datetime 2.9.3

Kapitel 7

Evaluierung

Das abschließende Kapitel befasst sich mit der Evaluierung der API Anbieter aufgrund der erfassten Daten mittels . Außerdem werden noch weitere Kriterien hinzugezogen. Die Evaluierungsergebnisse der API Anbieter werden in Kapitel 7.9 in einer Tabelle gegenübergestellt. Ein Hauptproblem bei der Evaluierung ist, dass hier sehr viele Faktoren berücksichtigt werden müssen, wobei jeder Anbieter unterschiedlich gut abschneidet. Das hat zur Folge, dass bei der Evaluierung auch auf die Anwendungsfälle Rücksicht genommen werden muss.

7.1 Ansatz

Die Evaluierung der verschiedenen API Anbieter erfolgt aufgrund der folgenden Faktoren: Kosten, Setup, Komplexität, Konfigurierbarkeit, Portierbarkeit, Laufzeiten für Training und Predictions und Genauigkeit. Das Ziel der Evaluierung soll sein, dass ein Entwickler die Anbieter aufgrund der Kriterien, die für sein Projekt wichtig sind, miteinander vergleichen kann. Die Prioritäten der Kriterien sind für jeden Anwendungsfall neu zu bewerten.

Der gewählte Ansatz betrachtet jedes Kriterium in sich geschlossen. Es werden keine anderen Faktoren als nur jene, die das Kriterium betreffen, zugelassen. Dadurch ergibt sich ein möglichst transparentes Bild der evaluierten Kriterien. Für jedes Kriterium kann ein API Anbieter zwischen 0-2 Punkte bekommen, wobei es hier 0,5 Punkte Schritte gibt. 0 Punkte bedeutet, dass ein Kriterium überhaupt bis wenig erfüllt wird. 2 Punkte bedeuten, dass es das Kriterium sehr gut erfüllt. Es kann möglich sein, dass bei manchen Kriterien mehrere Anbieter die Höchstzahl erreichen. Die Skala wurde bewusst von 0-2 gewählt, da eine höhere Genauigkeit nicht nötig ist und nur Komplexität hinzufügen würde. Außerdem kann die Bewertung auf ein Notensystem umgelegt werden, da es genau 5 Möglichkeiten der Bewertung gibt.

7.2 Kosten

Kosten können ein nicht unerheblicher Faktor bei der Entscheidung eines Anbieters sein. Wobei man nicht annehmen sollte, dass teuerere Softwareprodukte eine bessere Qualität als Open-Source-Produkte besitzen.

BigML Bei BigML handelt es sich um ein kommerzielles Produkt, bietet aber im Vergleich zu Google eine deutlich übersichtlichere Preispolitik an. Für Entwickler gibt es einen *Development mode*, mit dem kleine Tasks unter 16 Megabyte gratis ausgeführt werden können. Daneben können entweder verschiedene Abonnements abgeschlossen, oder Credit Points gekauft werden. Es gibt 3 verschiedene Abonnements, die sich nur in der Größe der Tasks und die Anzahl der parallel laufenden Tasks unterscheiden. Der Tarif *Standard* bietet max. Task-Größe von 64 Megabyte und 2 parallelen Tasks um 30$ / Monat. Der Tarif *Boosted* bietet max. Task-Größe von 1 Gigabyte und 4 parallelen Tasks um 150$ / Monat und der Tarif *Pro* kostet 300$ / Monat und bietet 8 parallele Tasks und eine maximale Task-Größe von 4 Gigagbyte. Dabei werden Studenten, öffentlichen Forschern und NGOs ein 50-prozentiger Nachlass geboten. Dagegen kostet jeder Credit Point 5 Cent und folgt dem *Pay-as-you-go*-Prinzip, wobei nur soviele Credit Points eingekauft werden müssen, wie benötigt werden. Wobei man hier für jeden Megabyte für Datensets und Trainingsdaten, sowie die Anzahl der getätigten Predictions bezahlt.

1 Punkt

Google Prediction API Google verfolgt mit seiner Cloud Plattform eine relativ unübersichtliche Preispolitik, was bei der Anzahl von Dienste auch weiters nicht verwunderlich ist. Für die Prediction API gibt es ein Freikontingent, aber nur 6 Monate gültig und auf insgesamt 20.000 Predictions begrenzt. Zudem weist es tägliche Einschränkungen von 100 Predictions und 5MB an Trainingsdaten pro Tag auf. Die bezahlte Variante sieht 10$ Grundgebühr für jedes Projekt vor. Dabei sind 10.000 Predictions beinhaltet. Danach kosten 1000 Predictions 50 Cent. Weiters zahlt man pro Megabyte Trainingsdaten 0,2 Cent, wobei die maximale Größe 2,5 Gigabyte an Trainingsdaten beträgt. Jedoch gibt es auch hier Einschränkungen: Benötigt man mehr als 40.000 Predictions pro Tag, muss zuerst Google kontaktiert werden.

0,5 Punkte

Prediction.io Prediction.io wurde erst zuletzt von SalesForce übernommen[31], das Produkt bleibt aber weiterhin als Open Source bestehen und

kann jederzeit über Github[20] betrachtet werden.

2 Punkte

Weka Toolkit Bei Weka handelt es sich wie bei Prediction.io um Open Source Software, die unter der GNU General Public License lizensiert ist[34]. Der Source Code ist auf ihren Servern über SVN zugänglich[21]

2 Punkte

7.3 Setup

Der Setup-Prozess beschreibt den Ablauf, beginnend bei der Anmeldung des Produkts, bis hin zur eigentlichen Verwendung des APIs im eigenen Programm. Ein unübersichtlicher Setup-Prozess kann sehr viel Zeit kosten und ist meistens gleichbedeutend mit einer unzureichenden Dokumentation des Produkts.

BigML Das Setup von BigML ist äußerst intuitiv gestaltet. Die Weboberfläche führt anschaulich alle Funktionen auf und die diversen APIs befinden sich auch auf einer gemeinsamen Seite. Außerdem benötigt man für die Registrierung keine Kreditkarte. Der gesamte Ablauf ist einfach und kurz gehalten.

2 Punkte

Google Prediction API Ein Punkt, den man Google zugute halten muss, ist die Konsistenz ihrer Cloud Produkte. Das bedeutet, dass das Setup und die APIs produktübergreifend beinahe identisch sind, und somit der Wechsel zwischen den verschiedenen Produkten schnell vonstatten geht. Über die Google Cloud Console muss zuerst ein neues Projekt erstellt und das Prediction API explizit hinzugefügt werden. Danach müssen noch Zugangsdaten erzeugt und exportiert werden und schon ist der Setup-Prozess fast beendet. Das Prediction API für Java umfasst eine Vielzahl von Googles Bibliotheken, es wird empfohlen, dass das API vorzugsweise über Gradle heruntergeladen wird. Trotzdem ist der Setup-Prozess von Google APIs sehr kompliziert, vor allem wenn man keine Erfahrung mit Google APIs hat kann es sehr viel Zeit kosten.

1 Punkt

[20] https://github.com/PredictionIO/PredictionIO/
[21] https://svn.cms.waikato.ac.nz/svn/weka/

Prediction.io Auf der Homepage gibt es zwar eine Anleitung, jedoch funktioniert diese nicht wie eigentlich erhofft. Der große Nachteil beim Setup ist, dass dafür eine Linux- oder Mac OS-Maschine benötigt wird. Des Weiteren werden auch erweiterte Kenntnisse der Shell und Private-Public-Key Verschlüsselung benötigt. Das komplette Setup findet über die Kommandozeile statt und somit findet sich der Setup-Prozess von Prediction.io am anderen Ende des Spektrums als vergleichsweise BigML wieder.

0,5 Punkte

Weka Toolkit Beim Weka Toolkit handelt es sich um eine Java Applikation, die nebenbei auch ein Java API bietet. Die Applikation kann über das Internet heruntergeladen und installiert werden. Setup gibt es nicht wirklich, jedoch ist die Oberfläche mit sehr vielen Funktionen ausgestattet, wodurch die eigentliche Verwendung des Programms eine gewisse Einarbeitungszeit benötigt.

1,5 Punkte

7.4 API-Komplexität

Die Lernkurve von neuen APIs ist ein nicht zu unterschätzendes Kriterium. Ein API sollte im besten Fall selbsterklärende Methoden besitzen und der Workflow bestmöglich dokumentiert. Tools wie Gradle und Maven bieten Entwicklern die Möglichkeit externe Softwarebibliotheken sehr komfortabel in das eigene Programm einzubinden. Durch die Einfachheit fremden Code zu verwenden, anstatt sich die Mühe zu machen ihn selbst zu schreiben, muss natürlich auch die verwendete Bibliothek ein dementsprechend einfaches API bieten um auf die Funktionen zuzugreifen. In vielen Bereichen gibt es mehrere APIs für einen Anwendungsfall. Meistens werden die Bibliotheken mit dem einfachsten API verwendet. Im folgenden Absatz wird nur Bezug auf das jeweilige Java API genommen.

BigML BigML eine sehr gutes und robustes API. Es setzt vermehrt auf das Datenformat JSON, damit die Schnittstellen möglichst generisch gehalten werden. Das API weist jedoch eine schlechte Dokumentation auf. Es gibt lediglich eine Referenzimplementierung, was in den meisten Fällen auch ausreichend ist, aber sicherlich besser ausgeführt werden könnte.

1,5 Punkte

Google Prediction API Wie schon beim Setup gilt für die Komplexität dasselbe. Hat man einmal ein Google API verwendet, so findet man sich

sofort bei einem anderen API zurecht. Für das Prediction API ist wenig Dokumentation vorhanden. Der Code für die Authentifizierung kann aber direkt von dem Gmail API kopiert werden, welche auch besser dokumentiert ist. Nichtsdestotrotz ist das API für Neueinsteiger doch eher kompliziert.

1,5 Punkte

Prediction.io Im Grunde hat Prediction.io ein schlankes Java API, aber eine enorm lückenhafte Dokumentation, die eine rasche Implementierung beinahe unmöglich macht. Es ist nicht wirklich selbsterklärend und in Kombination mit der schlechten Dokumentation ergibt sich ein API mit deutlichem Verbesserungspotential.

0,5 Punkte

Weka Toolkit Ein solides API, bei dem die verwendeten Klassen sehr sprechend benannt sind. Mit einigen Hintergrundwissen zu Machine Learning und den dabei verwendeten Algorithmen kann man schnell in das gut strukturierte API einsteigen. Durch die vielen Klassen verliert man zwangsläufig den Überblick und die Komplexität des APIs ist nicht unerheblich. Dennoch ist die Architektur der Bibliothek sehr gut, die Komplexität ist vor allem dem Funktionsumfang geschuldet.

1,5 Punkte

7.5 Konfigurierbarkeit

Für Machine Learning gibt es eine Vielzahl von Algorithmen und diese müssen dementsprechend konfiguriert sein. Es gibt keinen Algorithmus, der bei jedem Anwendungsfall immer die besten Ergebnisse liefert. Hier sollte der Programmierer die Wahl zwischen verschiedenen Algorithmen haben und den besten für seinen Anwendungsfall verwenden. Somit ist Konfigurierbarkeit wohl eines der wichtigsten Kriterien, wenn es um API Anbieter geht.

BigML BigML ist nicht wirklich konfigurierbar, aber immerhin weiß der Entwickler, dass Entscheidungsbäume zum Einsatz kommen. Dadurch ist eine Eingrenzung der Anwendungsfälle gegeben. Außerdem ist ein Entscheidungsbaum eine Whitebox und dadurch für den Entwickler immer transparent. Zusätzlich kann man Ensembles kreieren und die Entscheidungsbäume als PMML-Dateien exportieren.

0,5 Punkt

Google Prediction API Das API ist im Grunde nicht konfigurierbar und auch der Entwickler hat keine Ahnung was das dahinterliegende System macht. Es handelt sich um eine klassische Blackbox. Es funktioniert, nur hat man keinerlei Information darüber wie es optimiert ist und wie man es konfigurieren könnte.

0 Punkt

Prediction.io Eigentlich sollte ein eigener Prediction-Server sehr gut zu konfigurieren sein. Jedoch stellt sich die Konfiguration von Prediction.io schwieriger dar als es sein sollte. Immerhin wird auf die Machine Learning Bibliothek *MLlib* zurückgegriffen. Das System ist in der verwendeten Version(0.9.5) nicht wirklich konfigurierbar, weitere Einstellungsmöglichkeiten sollen per Update nachgereicht werden

0,5 Punkt

Weka Toolkit Das Kriterium Konfigurierbarkeit gewinnt Weka um Längen. Kein anderer API Anbieter ist in solchem Maße konfigurierbar wie Weka. Hier kann wirklich jeder Algorithmus ausgewählt und nach den eigenen Wünschen angepasst werden. Nebenbei gäbe es auch die Möglichkeit eigene Klassifizierer selbst zu schreiben und diese wie andere Klassifizierer zu verwenden. Natürlich ist das nicht der primäre Fokus, aber es soll verdeutlichen wie flexibel und konfigurierbar das Weka Toolkit ist.

2 Punkte

7.6 Portierbarkeit

Mit Portierbarkeit ist hier der Wechsel auf diverse Plattformen/Programmiersprachen gemeint. Dabei handelt es sich nicht um die klassische Portierbarkeit von Source Code per se, sondern vielmehr auch die Einschränkungen von einer Plattform oder auch nur einer Pogrammiersprache.

BigML Immerhin handelt es sich bei BigML um ein Online-Tool, also ist es von praktisch jeder Plattform aus zugänglich, welche über HTTP kommunizieren kann. Es gibt außerdem ein großes Arsenal an APIs, womit BigML von Android, iOS, Windows, Linux und Mac OS X ohne weitere Probleme und auch in mehreren Programmiersprachen zugänglich ist.

2 Punkte

Google Prediction API Für das Google Prediction API gilt dasselbe wie für BigML. Als Entwickler kann man ebenfalls aus mehreren Programmiersprachen wählen. Google unterstützt neben Java, Objective-C und .NET vor allem sehr viele Web APIs, wie PHP, Javascript, Node.js und Ruby.

2 Punkte

Prediction.io Der Grund warum Prediction.io so vielversprechend klingt ist, dass er alle Eigenschaften von Googles und BigMLs Lösung bietet, nur transparent im eigenen Netzwerk gehostet ist und damit einhergehend auch besser konfiguriert werden kann. Die bekanntesten Programmiersprachen werden wie bei Google und BigML ebenfalls unterstützt. Es fehlt lediglich der grundlegende Windows Support, damit der Server auf Windows Hostrechnern installiert werden kann.

1,5 Punkte

Weka Toolkit Die Portierbarkeit von Weka hält sich als einzige in Grenzen, da es auf Java als Programmiersprache limitiert ist. Wobei Java sehr vielseitig ist und auf Windows, Linux, Mac OS und zusätzlich noch auf Android lauffähig ist. Sollte Weka aber auch unter iOS erreichbar sein, so würde immer noch die Möglichkeit bestehen, mit diversen Java-Technologien einen Server aufzusetzen und Weka am Server zu betreiben. Damit würde man die gleiche Funktionalität wie die oben genannten API Anbieter zur Verfügung stellen, jedoch mit dementsprechend mehr Aufwand.

1 Punkte

7.7 Laufzeiten

Die Punktevergabe erfolgt in diesem Punkt anders als in den vorangegangenen Punkten. Der beste Anbieter erhält die Höchstpunktzahl. Die Laufzeiten werden anhand der drei in Kapitel 5.3 beschriebenen Datensets bestimmt. Die durschnittliche Prädiktionsdauer wird aus 30 Prädiktionen errechnet. Aufgrund der Komplexität von Prediction.io liegen keine Ergebnisse vor.

BigML 1,5 Punkte

	Iris	Diabetes	Brustkrebs	Durschnitt
Trainingsdauer	13.879s	11.141s	10.037s	**11.686s**
Prädiktionsdauer	0ms	2ms	1ms	**1ms**

Tabelle 7.1: BigML Laufzeiten.

Google Prediction API 1 Punkte

	Iris	Diabetes	Brustkrebs	Durchschnitt
Trainingsdauer	28.577s	578.388s	29.393s	**212.120s**
Prädiktionsdauer	548ms	500ms	597ms	**548ms**

Tabelle 7.2: Google Laufzeiten.

Prediction.io 0 Punkte

	Iris	Diabetes	Brustkrebs	Durchschnitt
Trainingsdauer	- s	- s	- s	**- s**
Prädiktionsdauer	- ms	- ms	- ms	**- s**

Tabelle 7.3: Prediction.io Laufzeiten.

Weka Toolkit 2 Punkte

	Iris	Diabetes	Brustkrebs	Durchschnitt
Trainingsdauer	2.944s	3.538s	2.885s	**3.122s**
Prädiktionsdauer	0ms	0ms	0ms	**0ms**

Tabelle 7.4: Weka Laufzeiten.

7.8 Genauigkeit

Wie in Kapitel 7.7 beschrieben, wird auch bei der Genauigkeit die Punkte nach dem Notensystem vergeben. Die Genauigkeit bezieht sich auf die im ebenfalls im vorigen Kapitel beschriebenen Prädiktionen. Bei identer Genauigkeit wird die gleiche Punktanzahl vergeben. Aufgrund der Komplexität von Prediction.io liegen keine Ergebnisse vor.

BigML 2 Punkte

	Iris	Diabetes	Brustkrebs	Gesamt
Genauigkeit	93.3%	100%	66.7%	**86.7%**

Tabelle 7.5: BigML Genauigkeit.

Google Prediction API 1 Punkte

	Iris	Diabetes	Brustkrebs	Gesamt
Genauigkeit	70%	43.3%	50%	**54.4%**

Tabelle 7.6: Google Genauigkeit.

Prediction.io 0 Punkte

	Iris	Diabetes	Brustkrebs	Gesamt
Genauigkeit	- %	- %	- %	**- %**

Tabelle 7.7: Prediction.io Genauigkeit.

Weka Toolkit 1,5 Punkte

	Iris	Diabetes	Brustkrebs	Gesamt
Genauigkeit	86.7%	73.3%	73.3%	**77.8%**

Tabelle 7.8: Weka Genauigkeit.

7.9 Gesamtfazit

BigML BigML ist sicherlich die beste Wahl, wenn es um Klassifizierung geht und Entscheidungsbäumen verwendet werden. Die *Local Prediction Models* erlauben eine rasche Klassifizierung am Gerät. Zusätzlich ist es auch der einzige Anbieter, der PMML-Dateien exportieren kann.

+ Local Prediction Models
+ PMML Export
+ Kostenlose Datensets

- Eingeschränkter Umfang (nur Entscheidungsbäume)
- Kostenpflichtig
- API wenig dokumentiert

Google Prediction API Das Google Prediction API ist nur ein kleiner Teil der durchaus mächtigen Cloud Plattform. Es ist sehr simpel, aber durch eine schlechte Dokumentation ist die Verwendung durchaus umständlich. Außerdem gibt es keine Möglichkeit zur Konfiguration und es bietet vergleichsweise nur durschnittlich gute Prädiktionen.

+ Simples API
+ Einbindung in Google Cloud Plattform
+ Vordefinierte Models

- Sehr schlecht dokumentiertes API
- Kaum konfigurierbar (Blackbox)
- Ungenau im Vergleich zu BigML und Weka

Prediction.io Das Konzept eines eigenen Prediction Servers klingt sehr vielversprechend und Prediction.io bietet auch einen spannenden Ansatz mit der Separierung zwischen Event-Server und Engine-Sever. Jedoch ist die Umsetzung derart schlecht gelöst, dass eine Einbindung ein äußerst hohes Maß an Konfiguration und Quellcode benötigt. Das API ist wenig umfangreich und nur auf bereits verfügbare Engines (Models) abgestimmt. Prediction.io ist zu komplex, um eine einfache Einbindung in Programme zu gewährleisten, weswegen auch keine Testergebnisse dafür vorliegen.

+ Eigener Server
+ Kostenlose Engines
+ Open Source

- Kompliziertes Setup
- Kein Java-API für Modelltraining

- Einbindung von eigenen Engines nicht praktikabel

Weka Toolkit Das Toolkit bietet die beste Konfigurierbarkeit, benötigt aber dementschprechende Kenntnisse in Machine Learning um gute Resultate zu erzielen. Es handelt sich beim Weka Toolkit zwar um ein Open-Source-Produkt, ist aber nur auf die Programmiersprache Java begrenzt.

+ Hohe Konfigurierbarkeit
+ Vollständiges Programm
+ Open Source

- Komplexes, umfangreiches API
- Auf Java als Programmiersprache begrenzt
- Konfiguration benötigt Kenntnisse in Machine Learning

	BigML	Google Prediction API	Prediction.io	Weka Toolkit
Umgebung	Cloud / Lokal	Cloud	Cloud	Lokal
Kosten	1	0,5	2	2
Setup	2	1	0,5	1,5
API Komplexität	1,5	1,5	0,5	1,5
Konfigurierbarkeit	0,5	0	0,5	2
Portierbarkeit	2	2	1,5	1
Laufzeiten	1,5	1	0	2
Genauigkeit	2	1	0	1,5
Gesamt	**10,5**	**7**	**5**	**11,5**

Tabelle 7.9: Vergleich der verschiedenen API Anbieter.

7.10 Zusammenfassung und Aussicht

Die Umgebung beschreibt, ob es sich um eine Cloud-basierte oder eine lokale Lösung handelt. Die Umgebung hängt vom zugrundeliegenden Anwendungsfall ab und wird nicht in die Evaluierung miteinbezogen.

Aus den verwendeten API Anbietern stechen ganz klar BigML und das Weka Toolkit heraus. Das Google Prediction API weist nur durschnittliche Werte auf. Enttäuschende Werte lieferte hingegen der Prediction Server Prediction.io, der äußerst kompliziert und umständlich aufzusetzen ist. Hier wurde sicherlich mehr erwartet, da das Konzept eines eigenen Prediction Servers – so wie ihn BigML und Google anbieten – durchaus ein spannendes Potential mit sich bringt. Die Implementierung eigener Models (oder Engines

wie sie bei Prediction.io heißen) gestaltet sich als nicht trivial, im Vergleich zu den Implementierungen der anderen Anbieter. Durch diese Komplexität war es nicht möglich Laufzeit- und Genauigkeitsmessungen für Prediction.io durchzuführen. Im Mittelmaß findet sich Google mit deren API. Es bietet ein durschnittlich gutes API mit wenig Konfigurationsumfang. Dafür kann die Software relativ rasch in eine bereits bestehende Applikation eingebunden werden. Die Genauigkeit der Prädiktionen hinkt jedoch denen von BigML und Weka nach. Für Desktopapplikationen und Android-Apps, die mithilfe von Java entwickelt werden können, ist sicherlich das API von Weka eine exzellente Wahl. Es ist zwar wie erwähnt nur auf Java limitiert und der Programmierer benötigt auch einiges an Wissen im Bereich von Machine Learning. Die Konfigurierbarkeit von Weka ist mit den anderen Anbietern nicht zu vergleichen. Wird aber nur eine Klassifizierung benötigt, so kann auch BigML verwendet werden. Das ist der einzig signifikante Nachteil gegenüber Weka. Während Weka direkt am Gerät ausgeführt werden muss (sowohl Training als auch die Prädiktion) wird das Training von neuen Modellen bei BigML online ausgeführt. Die Prädiktionen können wahlweise am Gerät oder online erfolgen. BigML ist wie die Lösung von Google ab einem gewissen Nutzungsgrad kostenpflichtig. Außerdem sind die Models online synchronisiert und dadurch von diversen Geräten zugänglich. Bei den Laufzeiten gibt es keine wirklichen Schwankungen. Das Google Prediction API benötigt circa eine halbe Sekunde pro Prädiktion, während Weka und BigML (mit Local Prediction Models) zwischen 0-3 Millisekunden benötigen. Je nach Anwendungsfall empfiehlt sich die Verwendung von Weka oder BigML.

Machine Learning ist in jeder Hinsicht eine vielversprechendes Thema für die Zukunft. Die Big Player im Geschäft wie Google, Apple, Amazon und Co. setzen auf smarte Assistenten, die nun auch schon Einzug im Haus halten. Die kontextbasierte, intelligente Suche rückt immer mehr in den Vordergrund. Durch die Digitalisierung der Welt werden vermehrt Daten aufgezeichnet, die mittels Machine Learning in Zusammenhang gebracht werden können. Mithilfe der verschiedenen API Anbieter hat nun der Entwickler die Möglichkeit seine Applikationen mit Machine Learning Algorithmen zu verknüpfen.

Quellenverzeichnis

Literatur

[1] IA Basheer und M Hajmeer. „Artificial neural networks: fundamentals, computing, design, and application“. In: *Journal of microbiological methods* 43.1 (2000), S. 3–31 (siehe S. 20).

[2] Rick Cattell. „Scalable SQL and NoSQL data stores“. In: *ACM SIGMOD Record* 39.4 (2011), S. 12–27 (siehe S. 30).

[3] Fay Chang u. a. „Bigtable: A distributed storage system for structured data“. In: *ACM Transactions on Computer Systems (TOCS)* 26.2 (2008), S. 4 (siehe S. 31).

[4] Judith E Dayhoff und James M DeLeo. „Artificial neural networks“. In: *Cancer* 91.S8 (2001), S. 1615–1635 (siehe S. 20).

[5] Acharjya D.P., Dehuri Satchidananda und Sanyal Sugata. *Computational Intelligence for Big Data analytics*. Cham, SUI: Springer International, 2015 (siehe S. 5).

[6] Eibe Frank u. a. „Weka“. In: *Data Mining and Knowledge Discovery Handbook*. Springer, 2005, S. 1305–1314 (siehe S. 41).

[7] Neha Gupta. „Artificial neural network“. In: *Network and Complex Systems* 3.1 (2013), S. 24–28 (siehe S. 19–21).

[8] Witten Ian H. und Frank Eibe. *Data Mining*. San Francisco, USA: Elsevier, 2005 (siehe S. 4, 9, 14, 15, 22, 24, 29).

[9] Kelleher John D., Mac Namee Brian und D'Arcy Aoife. *Fundamentals of machine learning for predictive data analytics*. Cambridge, USA: The MIT Press, 2015 (siehe S. 4–8, 10, 13–19, 21).

[10] Ward Jonathan Stuart und Barker Adam. *Undefined By Data: A Survey of Big Data Definitions*. Techn. Ber. University of St. Andrews, UK, 2013 (siehe S. 4, 5).

[11] Leskovec Jure, Rajaraman Anand und Ullman Jeffrey D. „Mining of massive datasets“. In: Stanford University, 2014. Kap. Recommendation Systems (siehe S. 7).

[12] Kaelbling Leslie Pack, Littman Michael L. und Andrew W. Moore. *Reinforcement Learning: A Survey*. Techn. Ber. Journal Of Artificial Intelligence Research, 1996 (siehe S. 23).

[13] Steve Lohr. „The age of big data“. In: *New York Times* 11 (2012) (siehe S. 1).

[14] Sutton Richard S. und Barto Andrew G. *Reinforcement Learning: An Introduction*. Cambridge, USA: The MIT Press, 2005 (siehe S. 23).

[15] Dennis W Ruck, Steven K Rogers und Matthew Kabrisky. „Feature selection using a multilayer perceptron“. In: *Journal of Neural Network Computing* 2.2 (1990), S. 40–48 (siehe S. 20).

[16] Justin Sahs und Latifur Khan. „A machine learning approach to android malware detection“. In: *Intelligence and Security Informatics Conference (EISIC), 2012 European*. IEEE. 2012, S. 141–147 (siehe S. 2).

[17] Raza Saleha und Haider Sajjad. *Suspicious activity reporting using dynamic bayesian networks*. Techn. Ber. ScienceDirect, 2010 (siehe S. 7).

[18] Nitin Sawant und Himanshu Shah. *Big Data Introduction*. Springer, 2013 (siehe S. 4, 29, 31).

[19] Margaret A Shipp u. a. „Diffuse large B-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning“. In: *Nature medicine* 8.1 (2002), S. 68–74 (siehe S. 6).

[20] Galit Shmueli und Otto Koppius. „Predictive analytics in information systems research“. In: *Robert H. Smith School Research Paper No. RHS* (2010), S. 06–138 (siehe S. 3).

[21] Richard S Sutton. „Introduction: The challenge of reinforcement learning“. In: *Reinforcement Learning*. Springer, 1992, S. 1–3 (siehe S. 23).

[22] Ayodele Taiwo Oladipupo. *Machine Learning Overview*. Techn. Ber. University of Portsmouth, UK, 2010 (siehe S. 6, 22).

[23] Abu-Mostafa Yaser S., Magdon-Ismail Malik und Lin Hsuan-Tien. *Learning from data*. Pasadena, USA: AMLbook, 2012 (siehe S. 13, 31–33).

[24] Qing-Hai Ye u. a. „Predicting hepatitis B virus–positive metastatic hepatocellular carcinomas using gene expression profiling and supervised machine learning“. In: *Nature medicine* 9.4 (2003), S. 416–423 (siehe S. 1).

[25] Xiaojin Zhu, Zoubin Ghahramani, John Lafferty u. a. „Semi-supervised learning using gaussian fields and harmonic functions“. In: *ICML*. Bd. 3. 2003, S. 912–919 (siehe S. 24).

Online-Quellen

[26] BigML. *BigML API*. URL: https://bigml.com/developers (besucht am 20. 10. 2015) (siehe S. 25).

[27] BigML Blog. *BigML Machine Learning models*. URL: http://blog.bigml.com/2012/08/16/machine-learning-throwdown-part-3-models/ (besucht am 08. 12. 2015) (siehe S. 11, 25).

[28] Apache Software Foundation. *Apache Hadoop*. URL: https://hadoop.apache.org/ (besucht am 18. 01. 2016) (siehe S. 31).

[29] Dorard Louis. *Different Prediction APIs*. URL: https://www.quora.com/Who-are-the-main-competitors-to-the-Google-Prediction-API (besucht am 19. 10. 2015) (siehe S. 6).

[30] Prediction.io. *Prediction.io*. URL: https://prediction.io/ (besucht am 10. 12. 2015) (siehe S. 24, 26).

[31] Prediction.io. *Salesforce acquires Prediction.io*. URL: https://blog.prediction.io/salesforce-signs-a-definitive-agreement-to-acquire-predictionio/#.Vyy5-PmLTIU (besucht am 06. 05. 2016) (siehe S. 59).

[32] Der Standard. *Amazon bestätigt Preisanpassungen je nach Kundeninteresse*. URL: http://derstandard.at/2000024879192/Amazon-bestaetigt-Preisanpassungen-je-nach-Kundeninteresse (besucht am 13. 11. 2015) (siehe S. 6).

[33] Irvine University Of California. *UCI Machine Learning Repository*. URL: https://archive.ics.uci.edu/ml/datasets.html (besucht am 16. 11. 2015) (siehe S. 8).

[34] University of Waikato. *Weka Toolkit*. URL: http://www.cs.waikato.ac.nz/ml/weka/ (besucht am 14. 11. 2015) (siehe S. 9, 24, 60).

[35] Matt Weinberger. *Google cloud could make more money than ads*. URL: http://www.businessinsider.de/urs-holze-talks-google-cloud-beat-search-2015-11?r=US&IR=T (besucht am 08. 12. 2015) (siehe S. 24).

[36] wired.com. *Google just Open Source TensorFlow*. URL: http://www.wired.com/2015/11/google-open-sources-its-artificial-intelligence-engine/ (besucht am 09. 12. 2015) (siehe S. 11).

Glossar

API Application Programming Interface, Programmierschnittstelle. 1, 24, 27, 36, 37, 40, 60–62

API Anbieter Eine Firma oder eine Software-Bibliothek, die es ermöglicht auf bereits implementierte Machine Learning Algorithmen über ein API zuzugreifen. 2, 6, 11, 24, 28, 34–36, 40, 41, 45, 46, 58, 62–64, 68, 69

ARFF Attribute-Relation File Format, ein Dateiformat, welches Weka für Trainingsdaten verwendet. 9

Blackbox Nicht transparentes System, außer den Eingangs- und Ausgangsgrößen ist nichts bekannt.. 11, 63

Cloud Verbund mehrerer Rechner, die über das Internet erreichbar sind.. 68

Data Prediction API Analysis Tool Java-Applikation für den Vergleich von Geschwindigkeit und Genauigkeit von Machine Learning APIs. 36, 38, 43

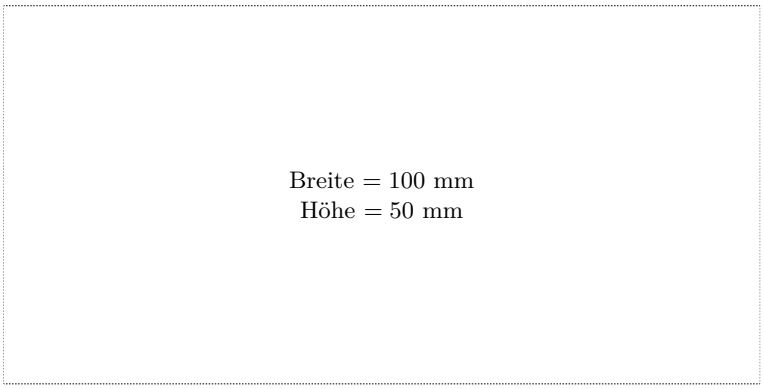
Machine Learning Künstliches Anlernen von Intelligenz, aufgrund von bereits vorliegenden Wissen in Form von Daten.. 43

PMML Predictive Model Markup Language, ein auf xml basierender Standard für Prediction Models. 4, 25, 49, 62

Whitebox Transparentes System, die Funktion des Systems liegt offen.. 11, 25, 62

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —