Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Daniel Müller

Evaluation des „Apache Spark“-Frameworks für den Anwendungsfall Social Media Monitoring

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Daniel Müller

Evaluation des „Apache Spark“-Frameworks für den Anwendungsfall Social Media Monitoring

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Olaf Zukunft
Zweitgutachter: Prof. Dr. Axel Schmolitzky

Eingereicht am: 11. November 2016

Daniel Müller

Thema der Arbeit
Evaluation des „Apache Spark“-Frameworks für den Anwendungsfall Social Media Monitoring

Stichworte
Social Media Monitoring, Apache Spark, maschinelles Lernen, Softwareentwurf, Twitter, Filme, Box Office Analyse

Kurzzusammenfassung
Diese Arbeit befasst sich mit dem Framework zur Clusterberechnung „Apache Spark“. Speziell wird untersucht, ob sich das Framework für das Social Media Monitoring eignet. Um eine Evaluation vornehmen zu können, wird aufbauend auf dem Framework ein exemplarisches System entworfen und entwickelt. Dieses System wird Twitter und IMDb beobachten und anhand der dort gesammelten Daten Vorhersagen zum Erfolg von Filmen treffen. Anhand der während der Entwicklung gewonnenen Erfahrungen wird das Framework zum Abschluss evaluiert.
Daniel Müller

Title of the paper
Evaluation of the "Apache Spark" framework for the use case Social Media Monitoring

Keywords
social media monitoring, Apache Spark, machine learning, software engineering, twitter, movies, box office analysis

Abstract
This thesis is about the cluster-computing framework "Apache Spark". It will be examined, whether the framework is capable of creating a powerful social media monitoring application. In order to properly evaluate the framework, a exemplary software-system will be designed and developed. This system will monitor Twitter and IMDb and will be able to predict the opening-weekend success of a movie based on the data collected from those sources. At the end, the framework will be evaluated based on the experience gained during the development process.

Inhaltsverzeichnis

1 Einleitung **1**

2 Analyse **2**
2.1 Social Media Monitoring . . . 2
2.2 Maschinelles Lernen(ML) . . . 4
2.2.1 Verwendete Algorithmen . . . 6
2.3 Apache Spark Framework . . . 8
2.4 Fallstudie . . . 11

3 Anforderungen **12**
3.1 Funktionale Anforderungen . . . 12
3.2 Nicht-funktionale Anforderungen . . . 13

4 Konzept **15**
4.1 Datenmodell . . . 15
4.1.1 Technologieauswahl . . . 16
4.1.2 Datenmodell auf MongoDB angewendet . . . 17
4.2 Kontextsicht . . . 19
4.3 Bausteinsicht . . . 20
4.4 Komponenten im Detail . . . 21
4.4.1 DataAcquisitions . . . 21
4.4.2 DataAccess . . . 23
4.4.3 DataAnalysis . . . 24
4.4.4 Controller . . . 30
4.4.5 Laufzeitsicht . . . 32

5 Realisierung und Tests **36**
5.1 Realisierung . . . 36
5.2 Tests . . . 37
5.2.1 Testziel . . . 37
5.2.2 Testumgebung . . . 38
5.2.3 Testvorgehen . . . 38
5.2.4 Testdaten . . . 39
5.3 Erkenntnisse . . . 40

6 Experiment und Evaluation **42**
6.1 Experiment . . . 42
6.1.1 Erste Phase . . . 42
6.1.2 Zweite Phase . . . 43
6.1.3 Dritte Phase . . . 44
6.1.4 Vierte Phase . . . 45
6.1.5 Fazit . . . 45
6.2 Evaluation . . . 46
6.2.1 Lernkurve . . . 46
6.2.2 Entwicklungs-Performance . . . 48
6.2.3 Dependency Management . . . 48
6.2.4 Möglichkeit zum Performance- und Sicherheitstuning . . . 49
6.2.5 Funktionalität . . . 49
6.2.6 Skalierbarkeit . . . 51
6.2.7 Robustheit . . . 51
6.2.8 Wartbarkeit . . . 52
6.2.9 Testbarkeit . . . 52

7 Fazit und Ausblick **53**
7.1 Fazit . . . 53
7.2 Ausblick . . . 53

Abbildungsverzeichnis

2.1 Beispielhafter Decision Tree 8
2.2 Der „Spark Stack“ 9
2.3 Übersicht über den Großteil der in der MLlib zur Verfügung stehenden Algorithmen 10

4.1 Das Datenmodell 16
4.2 Kontextsicht 19
4.3 Bausteinsicht 20
4.4 Datenstruktur Tweet 21
4.5 Datenstruktur Movie 21
4.6 Sentiment Enumeration 21
4.7 Die DataAcquisitions Komponente 22
4.8 Die DataAccess Komponente 23
4.9 DataAnalysis Komponente: Übersicht 24
4.10 DataAnalysis Komponente: Sentiment Analyse 25
4.11 Vergleich von Sentiment Analysis APIs 26
4.12 DataAnalysis Komponente: Übersicht 29
4.13 Der Ablauf des GetComingSoonMovies-Jobs 32
4.14 Der Ablauf des UpdateMovies-Jobs 33
4.15 Der Ablauf des CreateStream-Jobs 34
4.16 Der Ablauf des PredictMovies-Jobs 35

6.1 Ergebnisse der zweiten Phase des Experiments 42
6.2 Ergebnisse der ersten Phase des Experiments 44
6.3 Ergebnisse der vierten Phase des Experiments 45

Listings

4.1 Tweets Collection JSON Aufbau 17
4.2 Movies Collection JSON Aufbau 18
4.3 Aufbau des Sentiment140 JSONs 28

1 Einleitung

Social Media Monitoring ist ein modernes Werkzeug, das uns erlaubt Meinungen und Anregungen der Bevölkerung weltweit wahrzunehmen und zu quantifizieren. Doch wie setzt man Social Media Monitoring um? In dieser Arbeit wird das Framework „Apache Spark“ angewandt, um ein Prognosesystem bezüglich der Filmindustrie zu erstellen. Das System wird soziale Medien überwachen, die gesammelten Daten analysieren und mithilfe dieser Daten durch maschinelles Lernen eine Prognose über den finanziellen Erfolg von Filmen abgeben. Anhand dieses Anwendungsfalls wird evaluiert, ob das „Apache Spark“- Framework ausreicht, um ein solches System zu entwickeln und wie gut sich „Apache Sparks“ Bestandteile für verschiedene Aufgaben eignen.

Zunächst wird in der Arbeit grundlegendes Wissen über die relevanten Themenbereiche zusammengetragen. Es werden die Bereiche Social Media Monitoring und maschinelles Lernen erläutert und das Apache Spark Framework und seine Bestandteile vorgestellt. Aufbauend auf diesem Wissen wird dann das oben genannte System detailliert entworfen, die Realisierung und aufgetretene Probleme beschrieben und getestet. Als nächstes wird das System über einen Zeitraum von 3 Monaten angewandt und die Ergebnisse ausgewertet. Dabei wird sich die Qualität der Ergebnisse mit steigender Datenmenge im Laufe des Experiments verändern. Abschließend wird eine Evaluation des „Apache Spark“- Frameworks im Allgemeinen, sowie in Bezug auf den Anwendungsfall durchgeführt. Dabei steht im Vordergrund herauszufinden, wie gut das Framework zu erlernen ist, wie schnell es anwendbar ist und in wie weit die Funktionalitäten vorhanden, nützlich und ausreichend sind.

2 Analyse

In diesem Kapitel wird grundlegendes Wissen zu den für die Arbeit relevanten Themengebieten zusammengetragen und das Apache Spark Framework mit all seinen Bestandteilen vorgestellt.

2.1 Social Media Monitoring

Der moderne Mensch kommuniziert gern und viel im Internet. Weltweit verzeichnen wir ca. 3,42 Mrd. Internetnutzer. Davon äußern ca. 2,31 Mrd. Menschen sekündlich ihre Meinung zu den diversesten Themen in sozialen Netzwerken, in Deutschland verbringt der Durchschnitt mehr als eine Stunde pro Tag mit Sozialen Medien, in den USA knapp 2 Stunden pro Tag [We Are Social (2016)]. Dabei wurden bei diesen Statistiken Internetnutzer von 16 - 64 Jahren in Betracht gezogen. Passt man das Alter nach unten hin an, sollte diese Zahl deutlich größer werden, da vor allem junge Menschen das Internet und speziell soziale Netzwerke nutzen [ARD-ZDF (2015)]. Dazu stehen ihnen diverse Möglichkeiten zu Verfügung: Web Blogs, Foren, Bewertungen zu Produkten, Kommentarfunktionen, etc. Vor allem beliebt sind die Sozialen Netzwerke Twitter, Facebook, Google+, YouTube, Reddit, aber auch themenspezifische Netzwerkseiten wie Yelp und IMDb. Mit diesen Diensten hat man bereits eine Menge an heterogenen Datenquellen, die alle große Mengen an Daten produzieren (Stichwort Big Data). Beispielsweise werden auf Facebook alle 20 Minuten 1 Million Links geteilt und 3 Millionen Nachrichten gesendet [Statistic Brain Research Institute (2015a)], während auf Twitter im Durchschnitt 58 Millionen Tweets pro Tag gesendet werden [Statistic Brain Research Institute (2015b)].

Dort geführte Diskussionen und Beiträge bieten sich dementsprechend gut für Marktforschungszwecke an. Ein Beispiel sind Produktbewertungen und Erfahrungsberichte auf Shoppingseiten wie Amazon. Für viele Nutzer sind die Meinungen anderer Personen kaufentscheidend. [Lange (2011); Sen (2015)]

Kurz gesagt: Social Media Monitoring ist das Verfolgen und Verstehen von Gesprächen im Internet. Dabei ist wichtig zu beachten, dass nur User Generated Content betrachtet wird. [Lange (2011)]

Man kann verschiedene Methoden des Social Media Monitoring unterscheiden [Sen (2015)]:

Privates Monitoring Das Verfolgen von Events, Bewertungen, Tweets etc. in privatem Interesse, wie zum Beispiel das vorn genannte Betrachten von Userreviews um zur Kaufentscheidung beizutragen.

Professionelles Monitoring Dabei kann man wiederum unterscheiden:

Automatisches Monitoring Mit Hilfe von Software werden relevante Inhalte identifiziert und semantisch analysiert.

Halbautomatisches Monitoring Die relevanten Quellen werden mit Hilfe einer Software zur Verfügung gestellt und zusammengefasst. Rechercheure wenden Suchanfragen auf die Quellen an, um relevante Ergebnisse zu erhalten. Diese werden dann von Experten im untersuchten Feld ausgewertet. Dabei ist zu anzumerken, dass die Software je nach Anforderung die ausgegebenen Daten verschieden stark Filtern kann, je nachdem wie intelligent das System ist.

Manuelles Monitoring Soziale Netzwerke werden von „Web-Spezialisten“ identifiziert und dann ungefiltert von Experten ausgewertet.

In dieser Arbeit wird ein System entworfen und realisiert, dass die sozialen Netzwerke Twitter und IMDb automatisch untersucht und die dort gefundenen Beiträge semantisch analysiert. Es handelt sich hierbei also um ein System zum automatischen Monitoring.

Mögliche relevante Daten, die gesammelt werden können sind [Lange (2011)]:

- die sich äußernden Personen
- die Stimmung der Inhalte und einzelnen Beiträge
- die Quellen, auf denen Kommunikation über ein Thema stattfinden
- die Menge der Beiträge pro Webpräsenz
- die Entwicklung eines Themas über einen Zeitraum

Diese Daten können auf verschiedenste Fragen hin analysiert werden. Hier ein paar Beispiele [Lange (2011)]:

- Wie ist das Image einer Marke im Internet?
- Wie ist die Meinung zu einem Produkt?
- Welche Anforderungen haben Kunden?

- Wie reagiert die Community auf verschiedene Inhalte und wie verändert sich in dem Zusammenhang die Wahrnehmung auf Inhalt X?
- Werden Markenrechte von einem Unternehmen verletzt?
- u.v.m.

Wichtig zur Analyse und damit zur Beantwortung der gestellten Frage ist es, sich klarzumachen, welche Daten und welche Äußerungen dafür relevant sind und die gesammelten Daten danach zu filtern [Sterne (2011)].

Zuletzt ist sehr wichtig anzumerken, dass diese Art der Datenakquisition nicht ohne rechtliche und ethische Probleme daher kommt. Zwar wurden die Daten von den Usern frei ins Internet gestellt, doch sind viele dieser Daten und Aussagen dennoch sehr privat und die Speicherung und Analyse dieser Daten ohne direkte Zustimmung der unbewusst Befragten steht im klaren Konflikt mit dem Bundesdatenschutzgesetz [Lange (2011)]. Darüber sollte sich jede Privatperson und jedes Unternehmen, das dieses Verfahren anwendet, im Klaren sein.

2.2 Maschinelles Lernen(ML)

Es ist nun Zeit ein Forschungsfeld vorzustellen, mit dessen Erkenntnissen es möglich ist ein automatisches Social Media Monitoring-System umzusetzen. Maschinelles Lernen ist ein Teil der künstlichen Intelligenz. [Wikipedia (2016b)] Es beschreibt die automatische Generierung von Wissen aus Erfahrung [Wikipedia (2016c)]. Generierung von Wissen bedeutet für uns, dass auch Vorhersagen getroffen werden können. Daher soll nicht nur bei Wiederholung einer Aufgabe ein gleiches oder besseres Ergebnis erzielt werden, sondern aus der Erfahrung so verallgemeinert werden, dass diese Verbesserung des Ergebnisses auch bei ähnlichen Aufgaben auftritt. Weil die meisten Domänen, die untersucht werden können, sehr umfangreich sind, untersucht man mit einem Lernsystem nur kleine Teile einer Domäne. Allerdings muss das System in der Lage sein aus dieser kleinen Menge an Erfahrungen auf bisher für das System unbekannte Aufgaben der Domäne zu schließen. Daher müssen lernende Systeme mit Hilfe von Heuristiken auswählen, welche der gesammelten Erfahrungen in Zukunft potentiell von Wert sein können. Dies nennt man induktive Kalibrierung(engl. inductive biases). Bestimmte Erfahrungen können das Lernergebnis negativ beeinflussen, z.B. wenn die gesammelten Daten falsch oder nur für einen extremen Sonderfall gedacht waren. Man muss daher besonders darauf Acht geben, wie man Erfahrungen als wertvoll klassifiziert. Diese Erfahrungen haben zur Folge, dass Änderungen im lernenden System vorgenommen werden [Luger (2002b)].

An dieser Stelle kann man drei Arten von Ansätzen nennen, nach denen Änderungen repräsentiert werden.

Symbolische Ansätze Das System wird durch explizit beschriebenes Domänenwissen beeinflusst. Es werden also Daten gesammelt und explizit für das lernende System vorbereitet, um so einen schnellen Lernerfolg zu verzeichnen. Ebenso speichert das System neu generiertes Domänenwissen in dieser exakten Weise. Meistens werden die Erfahrungen in eine formale Sprache wie z.B. Logik übertragen und repräsentiert. [Luger (2002b)]

Neuronale Netze Neuronale Netze sind vom Gehirn abgeleitet und sind daher Systeme aus miteinander verbundenen, künstlichen Neuronen. Neuronale Netze lernen nicht durch das Aneignen von Wissen in symbolischer Form. Das Wissen steckt in dem Aufbau und der Interaktion der Neuronen. Sie lernen nicht, indem man Wissen in einer bestimmten Repräsentation hinzufügt, sondern ändern ihre Gesamtstruktur. Wissen wird hier vor allem durch numerische Vektoren und Matrizenmultiplikation dargestellt. [Luger (2002b)]

Evolutionäres Lernen Angelehnt an die Theorie Charles Darwins handelt es sich hier um Lernen durch Anpassung. Dieses Lernmodell, gefunden in menschlichen und tierischen Systemen, gilt als eines der Stärksten. Menschliche und tierische Systeme müssen sich entwickeln und anpassen, um in der Welt bestehen zu können. Dieser Ansatz des Lernens durch Anpassung spiegelt sich durch die genetischen Algorithmen, genetische Programmierung und Forschung im Bereich künstliches Leben wider [Luger (2002b)]. In dieser Arbeit wird auf diesen Aspekt des maschinellen Lernens nicht weiter eingegangen.

Des Weiteren lassen sich Aufgaben des maschinellen Lernens in 3 Kategorien einteilen:

Überwachtes Lernen Das System erhält anfangs Trainingsdaten, die von einem „Lehrer“ klassifiziert wurden. Dem System wird so mitgeteilt, ob es sich bei einem Fall um ein positives oder negatives Beispiel handelt. [Luger (2002b)]

Unüberwachtes Lernen Im Gegensatz zum überwachten Lernen, wird das System hier nicht mit klassifizierten Beispielen trainiert. Das System muss hier selbst Kategorien bilden. Dies nennt man konzeptuelles Clustering und kann auch das eigentliche Ziel des Systems sein, wenn man zum Beispiel versteckte Zusammenhänge in Daten finden möchte. [Wikipedia (2016b)]

Verstärkendes Lernen Auch Lernen mit Bewertungsfunktion genannt. Hier wird ein so genannter Agent einer Umgebung ausgesetzt, in der er ein bestimmtes Ziel erreichen

soll. Er erhält in dem Kontext je nach ausgeführter Aktion eine Rückmeldung, ob die Aktion gut oder schlecht einzuordnen ist und lernt so eine optimale Strategie zur Lösung eines Problems.

Welche dieser Kategorien zum Einsatz kommt hängt von der jeweiligen Aufgabe und der zur Verfügung stehenden Daten ab. In dieser Arbeit wird überwachtes und unüberwachtes Lernen zum Einsatz kommen.

2.2.1 Verwendete Algorithmen

Im erstellten Programm werden zwei ML Algorithmen angewandt. Beide fallen in die Kategorie des überwachten Lernens. Es wird ein Klassifikations- und ein Regressionsalgorithmus verwendet. Der Klassifikationsalgorithmus wird Datensätze in vorgegebene Klassen, anhand von Erfahrungswerten, einordnen. Der Regressionsalgorithmus erschafft selbstständig, anhand von Erfahrungswerten, Klassen und ordnet die Datensätze dann diesen Klassen zu. Dadurch ist eine genauere Schätzung des Ergebnisses möglich.

Klassifikation - Naive Bayes Mit Hilfe des Naive Bayes Ansatzes kann eine Klassifikation mit mehr als zwei Klassen durchgeführt werden. In diesem Anwendungsfall werden die Klassen durch die Opening Weekend-Einnahmen der Filme vorgegeben. Um ein möglichst genaues Ergebnis zu erhalten muss daher eine relativ große Anzahl von Klassen verwendet werden. Außerdem ist Naive Bayes ein Algorithmus, der erfahrungsgemäß gut mit kleinen Datenmengen umgehen kann [Wikipedia (2016e)]. Die benötigten Daten für diesen Anwendungsfall liegen nicht vor und müssen eigenständig generiert werden, daher wird zu Beginn nur eine sehr kleine Anzahl an Datensätzen vorhanden sein. Dies musste in der Auswahl des Algorithmus bedacht werden. Daher wurde Naive Bayes gewählt, obwohl es eher ein Algorithmus zur Dokumenten-Analyse ist und mit Word-Bags gut arbeiten kann. Naive Bayes arbeitet mit der Annahme, dass die eingegebenen Daten unabhängig voneinander sind. D.h. dass kein Zusammenhang zwischen 3D Film und „viele Tweets“ vorhanden ist. Dies ist für unsere Fallstudie zuträglich, da diese Zusammenhänge nur in extremen Fällen bestehen, wie er 2009 bei „Avatar“ 3D Hype gegeben war. Würde ein solcher Zusammenhang dennoch bestehen, würde er gefunden werden, da Bayes' Theorem angewandt wird.

Bayes' Theorem Bayes' Theorem findet in der Wahrscheinlichkeitstheorie und Statistik Anwendung und beschreibt die Wahrscheinlichkeit des Eintretens eines Ereignisses, unter

Annahme von äußeren Umständen, die mit diesem Ereignis in Verbindung stehen könnten. Mathematisch ausgedrückt lautet das Theorem:

$$P(H_i|E) = \frac{P(E|H_i) \times P(H_i)}{\sum_{k=1}^{n} P(E|H_k) \times P(H_k)}$$

Dabei ist $P(H_i|E)$ die Wahrscheinlichkeit, dass Ereignis H_i eintritt, wenn Ereignis E gegeben ist. $P(E|H_i)$ ist dementsprechend die Wahrscheinlichkeit, dass Ereignis E eintritt, wenn Ereignis H_i eingetreten ist. $P(H_i)$ ist die Wahrscheinlichkeiten des Eintretens des EreignissesH_i. n ist die Anzahl möglicher Ereignisse. [Luger (2002a); Wikipedia (2016a)]

Der von Spark standartmäßig verwendete Ansatz ist der multinomiale naive Bayes. Dieser wendet die Multinomialverteilung lose auf das Bayessche Theorem an. Dabei werden die Wahrscheinlichkeiten des Eintretens der Ereignisse pro Klasse in einem Multinomial zusammengefasst und mit der tatsächlich gezählten Anzahl von eingetretenen Ereignissen verrechnet. [Apache Software Foundation (2016a); Wikipedia (2016d,e)]

Regression - Decision Tree Zur Regression wird das Decision Tree Modell verwendet. Decision Trees, oder Entscheidungsbäume, sind eine beliebte Methode zur Klassifikation und Regression. Ein Decision Tree hat die Eigenschaft gelerntes Wissen sehr gut zu repräsentieren, da durch die Baumdarstellung die Übersicht der gefundenen Zusammenhänge schnell erfolgen kann. Dabei ist jeder Knoten im Baum ein Test einer Eigenschaft. Jeder mögliche Wert ist eine Verzweigung von dem Knoten ausgehend. Die Blattknoten sind dann die zu errechnenden Klassen. Die Größe und damit Komplexität des Baumes hängt von der Reihenfolge ab, in der die Eigenschaften überprüft werden. In der Spark Dokumentation wird nicht explizit genannt, welcher Algorithmus zum Erstellen des Baumes genutzt wird. Es wird allerdings von einem Greedy-Algorithmus gesprochen. Das heißt es wird immer das beste Ergebnis für jeden Teilschritt gewählt. Es wird so vorgegangen, dass der Algorithmus eine Menge an Beispieldaten erhält. Je Größer die Menge, desto genauer das Ergebnis. Diese Menge wird pro Eigenschaft in Teilmengen aufgeteilt, die jeweils einen möglichen Wert für diese Eigenschaft beinhalten. Die überprüfte Eigenschaft ist der Ausgangsknoten. Der gemeinsame Wert ist die Kante, die nun zunächst zu der Teilmenge führt. Die neuentstandene Teilmenge wird jetzt auf eine noch nicht untersuchte Eigenschaft überprüft und wiederum aufgeteilt. Dies geschieht rekursiv, bis alle in der Teilmenge vorhandenen Einträge derselben Klasse angehören. Diese Klasse ist der Blattknoten und das Endergebnis. Entstanden ist ein Baum, der wie in Abbildung 2.1 Eigenschaften als Knoten und mögliche Werte der Eigenschaft als Kanten besitzt.

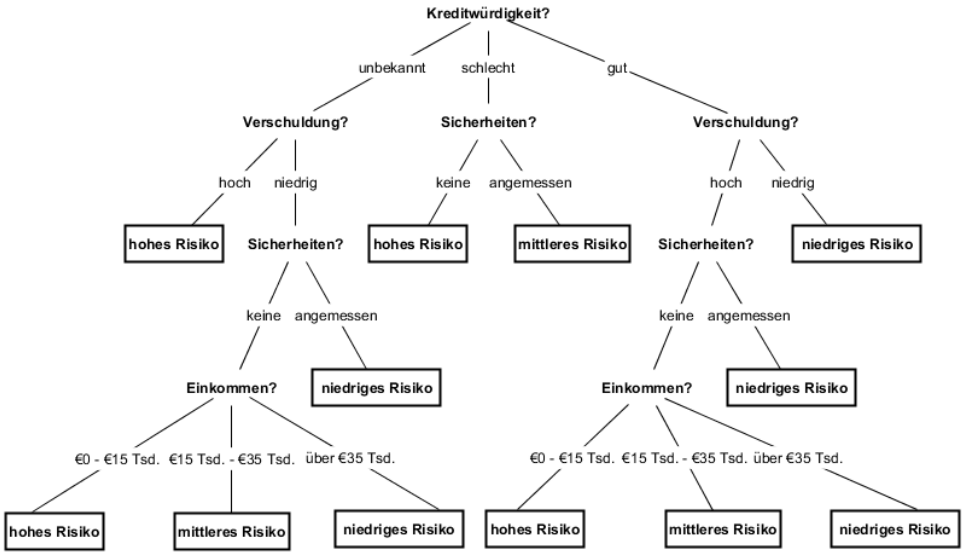


Abbildung 2.1: Beispielhafter Decision Tree

2.3 Apache Spark Framework

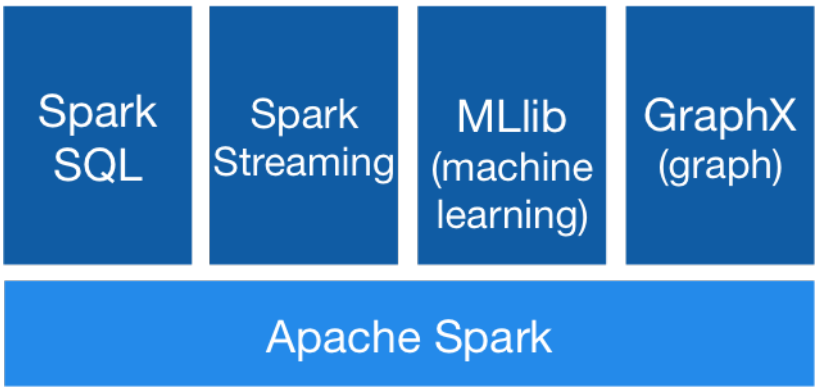
Apache Spark ist ein open-source Framework zur Datenanalyse, das auf Clusterberechnungen ausgelegt ist. Es wurde von der University of California, Berkeley als Verbesserung von Apache's Hadoop entwickelt. Diese beiden Frameworks ermöglichen es Berechnungen auf ein Cluster von Rechnern zu verteilen und setzen das Map-Reduce Modell um. Dabei ist Spark allerdings nicht an Map-Reduce gebunden, sondern kann vielfältiger eingesetzt werden. Dabei schafft es Spark laut eigenen Angaben bis zu 100x schneller zu sein [Apache Software Foundation (2016c)]. Diese Perfomancesteigerung kann deshalb erreicht werden, weil Spark im Gegensatz zu Hadoop nicht auf azyklischem Datenfluss basiert. D.h. im Fall von Hadoop muss jede Anwendung einzelne Jobs starten, die jeweils erst Daten aus z.B. einem verteilten Datensystem (HDFS) lesen und am Ende ihrer Ausführung die Daten dahin zurückschreiben. Dies verursacht hohe Latenzzeiten, vor allem da verteilte Dateisysteme redundant replizierte Datenspeicher sind.[Matei Zaharia (2010)]

Spark löst dieses Problem durch die sogenannten Resilient Distributed Datasets (RDDs). RDDs können zwischen Anfragen als Working Sets im verteilen Arbeitsspeicher gehalten werden, ohne repliziert werden zu müssen. Sie sind immutable objects, was bedeutet, dass jede Änderungsoperation auf einem RDD ein neues RDD erzeugt. Trotzdem sind sie fehlertolerant, da sie sich merken, wie sie aus welchen Datensätzen erstellt wurden und können sich daher bei

Fehlern selbst wieder rekonstruieren. Durch die RDDs ist es Spark im Gegensatz zu Hadoop möglich, iterative Algorithmen, interaktives Data Mining und Streaming Applikationen effizient umzusetzen. Damit soll es sich besser für komplexere Analysen eignen, die maschinelles Lernen (iterative Algorithmen), Graph-Algorithmen oder aggregierte Zustände benötigen.

Spark ist darauf ausgelegt mit einem verteilten Dateisystem (DFS), also im Cluster zu arbeiten. Dabei ist Spark so konzipiert, dass man es auf viele verschiedene dieser DFSs aufsetzen kann. Damit ist es flexibel einsetzbar, da auch schon bestehende Datenbanksysteme als Datenspeicher genutzt werden können. Zusätzlich bietet Spark für Entwickler und zum Testen einen pseudo-verteilten Modus an, der kein DFS benötigt, dann allerdings viele der Vorteile nicht ausnutzen kann. In diesem Modus simuliert jeder CPU-Kern einen Worker, also Rechner im Cluster. In dieser Arbeit wird dieser Modus verwendet. Es wird daher untersucht, wie das Framework anzuwenden ist, nicht jedoch beurteilt wie schwer es ist ein Cluster aufzusetzen oder die Performance im Cluster.

Spark bietet von Haus aus 4 Komponenten an: Spark SQL, Spark Streaming, Spark MLlib für maschinelles Lernen und Spark GraphX zum Arbeiten auf Graphen. Diese Komponenten setzen alle auf dem Spark Core auf und bilden den so genannten „Spark Stack“.



Quelle: spark.apache.org

Abbildung 2.2: Der „Spark Stack“

Apache Spark Core Der Spark Core ist zuständig für Management Funktionen, wie das Task Scheduling im Cluster. Er implementiert die RDDs. [Scott (2015)] Jede Spark Anwendung muss den Spark Core einbinden.

Spark SQL Spark SQL ist das Modul, das zum Arbeiten auf strukturierten Daten verwendet wird und implementiert dafür eine weitere Abstraktion namens „DataFrames“. Es bietet

SQL-esque Anfragemöglichkeiten, genau wie Apache's „Hive", um damit die DataFrames zu manipulieren. Außerdem unterstützt es JDBC und ODBC und damit auch indirekt alle damit kompatiblen Datenbanksysteme. [Scott (2015)]

Spark Streaming Spark Streaming ermöglicht skalierbares und fehlertolerante Streamverarbeitung und hat eingebaute Unterstützung für Streams von HDFS, Flume, Kafka, Twitter und ZeroMQ. Man hat allerdings auch die Möglichkeit eigene Datenquellen zu definieren. Es nutzt die RDDs und kann somit die Daten des Streams im Arbeitsspeicher vorbehalten. Die RDDs unterscheiden nicht zwischen Streaming- und Batch-Anwendungen. Dadurch ist es möglich in Spark geschriebene Batch-Analysen mit wenigen Änderungen auch für Streaming Analysen zu nutzen. [Scott (2015); Apache Software Foundation (2016e)]

Spark MLlib Die MLlib ist eine skalierbare Bibliothek für maschinelles Lernen. Sie ist darauf ausgelegt ML-Algorithmen auch verteilt und damit effizient auf großen Datenmengen auszuführen. Sie implementiert eine Anzahl von oft genutzten ML- und Statistik-Algorithmen. Darunter das Finden von Zusammenhängen, statistische Tests, Klassifikation, Cluster Analyse und Optimierungsalgorithmen.

Typ	Algorithmus
Basic Statistics	Summary Statistics, Correlations, Stratified Sampling, Hypothesis Testing, Random Data Generation
Classification and Regression	Support Vector Machines (SVM), Logistic Regression, Linear Regression, Naive Bayes, Decision Trees, Random Forests, Gradient-Boosted Trees
Collaborative Filtering	Alternating Least Squares (ALS).
Clustering	k-Means
Dimensionality Reduction	Singular Value Decomposition (SVD), Principal Component Analysis (PCA)
Feature Extraction and Transformation	Term Frequency-Inverse Document Frequency (TF-IDF), Word2Vec, StandardScaler, Normalizer

Quelle: www.contexagon.com

Abbildung 2.3: Übersicht über den Großteil der in der MLlib zur Verfügung stehenden Algorithmen

Zur MLlib gehören indirekt noch zwei weitere Teile: die MLI und der ML-Optimizer. MLI ist eine API, die es ermöglicht, eigene ML-Algorithmen und Features zu entwickeln und gleichzeitig Vorteile von Spark auszunutzen. Der ML-Optimizer soll ML-Probleme vereinfachen, indem er bei der Modellauswahl hilft und automatisiert. [Lorenz (2016)]

Spark GraphX GraphX war ursprünglich ein eigenes Projekt der University of Berkeley, das mittlerweile eine Erweiterung von Spark ist, die es ermöglicht Berechnungen auf und Analysen von Graphen durchzuführen. Dies ist gerade unter Verwendung von Maschinellem Lernen oder Data Mining sehr hilfreich.

All diese Module können miteinander verwendet und kombiniert werden und bieten so auf den ersten Blick alles, was man zum Social Media Monitoring und der Analyse benötigt.

2.4 Fallstudie

Um dieses Framework nun zu testen, benötigt man ein Beispielprojekt. Die hier erstellte Software soll Vorhersagen treffen können, ob ein Film am Opening Weekend ein Erfolg sein wird oder nicht. Dabei wird der Erfolg konkret vorhergesagt. Es werden also von der Software Vorhersagen zum tatsächlich eingespielten Box Office Ergebnis getätigt. Die hierfür benötigten Daten, also Ausstrahlungsdatum, diverse technische Informationen, Userbewertungen und tatsächliche Box Office-Einnahmen, werden per „HTML-Scraping“ auf der Internetseite „IMDb“ gesammelt und in einer Datenbank gespeichert.

Die Vorhersagen werden anhand von Twitter-Aussagen getroffen. Angesehen wird sich hierbei, wie viele Meldungen zu den Haupt- „Hashtags“ des Films, sowie zum Namen des Films eingegangen sind. Die Hashtags sind maschinell nur schwer zu ermitteln, daher kann der Benutzer diese zusätzlich eigenhändig in das System einpflegen. Ebenso wird er zusätzliche Schlagworte für Filme hinzufügen können. Diese so ermittelten Meldungen werden dann per „Sentiment-Analyse“ auf die Neigung untersucht, ob die Aussage eher positiv oder negativ behaftet ist. Die gesammelte Anzahl an positiven und negativen Aussagen wird ebenfalls zur Analyse verwendet. Meldungen werden in einem Zeitraum von einem Monat vor Filmstart bis zum Filmstart gesammelt, um ein aussagekräftiges Meinungsbild zur „Vorfreude“ zu bekommen. Weiterhin wird betrachtet, ob der Film eine Fortsetzung ist, ob er in 3D läuft oder ob es sich um ein Remake handelt.

Das fertige System wird mindestens zwei Monat laufen, um Tweets zu sammeln, zu analysieren, Vorhersagen zu treffen und das Ergebnis mit Realwerten vergleichen zu können. Bereits vor Fertigstellung des Systems werden Tweets zu allen im Zeitraum von April 2016 bis zur Fertigstellung des Systems erscheinenden Filmen gesammelt, damit diese später zur Analyse und als Trainingsdaten genutzt werden können.

3 Anforderungen

In diesem Kapitel werden die funktionalen und nicht-funktionalen Anforderungen an das zu erstellende System ermittelt und erläutert.

3.1 Funktionale Anforderungen

Aus der Fallstudie lassen sich nun einige funktionale Anforderungen ableiten:

F1. Sammeln von Tweets über einen nach Schlagworten zu Filmen gefilterten Twitter-Stream

F2. Text des Tweets per Sentiment Analyse auf Stimmung untersuchen

F3. Protokollierung der Stimmung zum jeweiligen Film

F4. Vorhersage über Erfolg der Filme anhand der Stimmung und Anzahl der zugehörigen Tweets

F5. Sammeln von Daten (Erscheinungsdatum, Box-Office-Einnahmen) zu demnächst erscheinenden Filmen aus der IMDb und Speicherung dieser Daten in einer Datenbank

Erläuterungen

F1 Teil der in der Fallstudie beschriebenen Aufgabe ist es, Tweets zu bestimmten Filmen zu sammeln. Das Verwenden eines Streams führt dazu, dass man Live-Daten erhält. Durch diese sollte es auch möglich sein Live-Vorhersagen zu treffen. Außerdem kann der Twitter-Stream 24/7 laufen und ständig neue Daten liefern. Dahingegen wird z.B. eine periodische Suche nach Tweets erstens durch Twitter beschränkt und zweitens würden viele redundante Daten generiert. Ein ungefilterter Stream würde viele irrelevante Daten liefern. Daher ist es wichtig den Stream nach Schlagworten zu filtern.

F2 Ebenfalls in der Fallstudie beschrieben wird die Aufgabe, Stimmungen der Tweets zu untersuchen, um diese zur Vorhersage zu verwenden.

F3 Die durch F2 erhaltene Stimmung sollte nicht verloren gehen, da sie zur späteren Vorhersage verwendet werden soll. Daher muss die Stimmung protokolliert bzw. gespeichert werden.

F4 Das Ziel der Software laut Fallstudie.

F5 Für die genauere Vorhersage benötigt man zusätzliche Daten über die zu untersuchenden Filme. Diese Daten sollten automatisch akquiriert werden.

3.2 Nicht-funktionale Anforderungen

Es werden nun die gängigen nicht-funktionalen Anforderungen auf ihre Wichtigkeit untersucht.

Zuverlässigkeit : Die Software soll fehlertolerant in Bezug auf häufig auftretende Fehlertypen sein, wie z.B. der Nicht-Verfügbarkeit des Twitter-Streams. Wiederherstellbarkeit der Daten des Streams im Fehlerfall ist nicht wichtig, da die in der Downtime verlorengegangenen Tweets im Vergleich zur Gesamtmenge keinen großen Anteil haben.

Benutzbarkeit Ist nicht wichtig, da in diesem System die Funktionalität im Vordergrund steht. Die Software muss gestartet und Suchbegriffe angepasst werden können.

Leistung und Effizienz Die Software muss so effizient sein, dass der Stream verarbeitet werden kann und ein Ergebnis in endlicher Zeit geliefert wird. Der Stream erhält ca. 10 Tweets pro Sekunde. Diese müssen verarbeitet werden können. Abseits davon ist die Effizienz nicht wichtig, da es sich hier lediglich um ein Forschungsprojekt handelt und die Ressourcen daher begrenzt sind.

Wartbarkeit Soll möglichst hoch sein. Aufgrund der fehlenden Erfahrung im Umgang mit dem gewählten Framework und ML im Allgemeinen, sollten Änderungen am System einfach vorgenommen und Fehlerquellen schnell identifiziert werden können. Ebenfalls soll das System erweiterbar sein, da neue Datenquellen in Zukunft von Bedeutung sein können.

Funktionalität Die Korrektheit der Ergebnisse kann nicht beurteilt werden. Es handelt sich um ein System, das Vorhersagen anhand von Daten trifft. Die Natur von Vorhersagen ist es, dass sie nicht zu 100% korrekt sein können. Aufgrund der Nutzung von maschinellem Lernen sollte die Korrektheit allerdings mit zunehmender Laufzeit des Systems steigen.

Sicherheit Ist im Rahmen der Bachelorarbeit nicht wichtig. Das System wird nicht von außen ansprechbar werden.

Verfügbarkeit Das System soll ohne Unterbrechung laufen, um keine Tweets zu verpassen. Pro Tag ist eine dennoch Unterbrechung von ca. 10 Sekunden vorgesehen, um den Stream mit neuen Schlagworten neu zu starten.

4 Konzept

In dieser Arbeit geht es darum Spark zu testen. Daher wird dieses Framework, wenn möglich verwendet. Obwohl Spark in Scala geschrieben wurde, wird hier Java genutzt werden, da sich die beiden Implementationen in der Funktionalität im Endeffekt nicht unterscheiden. Für Java gibt es eine große Anzahl an Frameworks, APIs und Bibliotheken, die sich als hilfreich erweisen können. Da diese im Allgemeinen auch in Scala genutzt werden können, sind die beiden Sprachen als gleichwertig anzusehen und können je nach Präferenz genutzt werden. Ebenfalls ist es möglich für Spark in Python zu entwickeln. Allerdings müssen dabei in einigen Fällen und Funktionalitäten Abstriche gemacht werden. Daher wurde Python als Programmiersprache nicht weiter in Betracht gezogen.

Für die Datenakquise wird ein konstanter Stream an Daten bzw. Tweets benötigt. Twitter bietet dafür den Twitter Public Stream an, in dem alle öffentlich geposteten Tweets erscheinen. Per Twitter-API kann sich man sich an diesen Stream anmelden. Über bestimmte HTTP-Requests können die Tweets, die man von dem Stream erhalten möchte, spezifiziert werden. [Twitter, Inc. (2016)] Spark bietet zum Verbinden mit dem Twitter Stream im Spark Streaming-Kontext eine Twitter-API an. Diese API basiert auf der Twitter4j API. Sie bringt allerdings zusätzlich den Stream direkt in ein für Spark verständliches und verteilbares Format. Da diese API angeboten wird, wird davon ausgegangen, dass dies die beste Möglichkeit ist, Spark im Zusammenspiel mit Twitter zu nutzen. Da Spark getestet werden soll, wurden keine alternativen APIs in Betracht gezogen.

Um die Daten der IMDb zu erhalten, wird ein HTML-Parser zum „Scrapen“ der Seite genutzt, da IMDb keine offizielle API anbietet. Die zugrundeliegende Datenbank wird eine MongoDB sein, da ich die Flexibilität und Einfachheit einer NoSQL-Datenbank schätze. MongoDB ist der erwählte Kandidat, da es einen offiziellen Connector gibt, der es ermöglicht Daten aus der MongoDB direkt in RDDs für Spark umzuwandeln und andersherum. [MongoDB, Inc. (2016)]

4.1 Datenmodell

Zur Erfüllung der Anforderungen F1, F3 und F5 wird eine Datenbank benötigt, da Daten gesammelt werden müssen. Das allgemeine Datenmodell umfasst die drei Entitäten User,

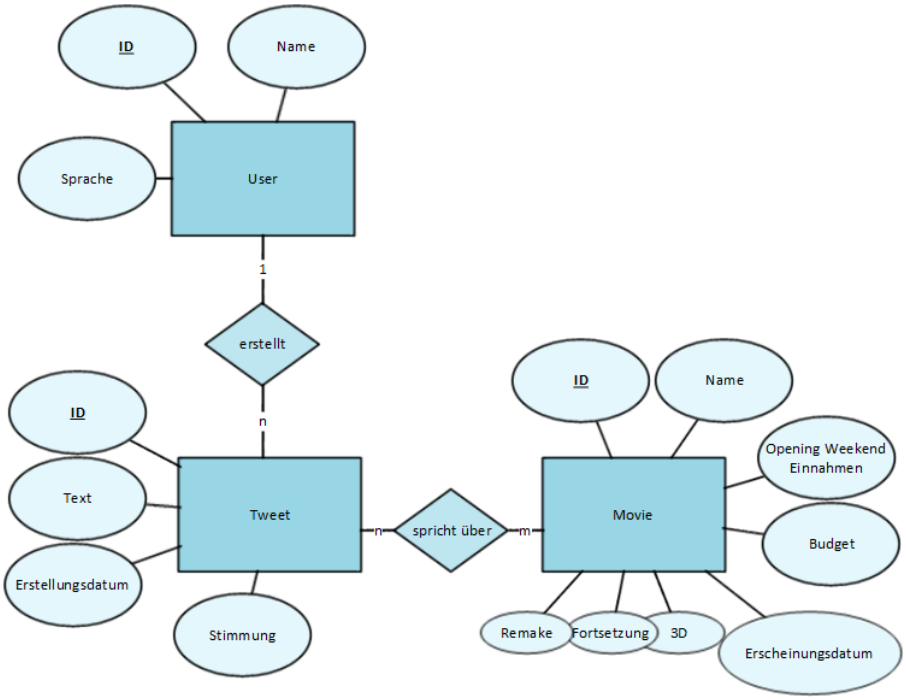


Abbildung 4.1: Das Datenmodell

Tweet und Movie. Ein User kann beliebig viele Tweets erstellen und jeder dieser Tweets kann über einen oder mehrere Filme sprechen. Jeder Tweet kann eine Stimmung ausdrücken, die in der Masse der Tweets einen Hinweis darauf gibt, wie sehr ein Film gemocht wird oder nicht. Von der anderen Seite aus betrachtet kann es zu jedem Film beliebig viele Tweets geben. Die Anzahl, sowie die Stimmung dieser Tweets macht es möglich Aussagen zu der Beliebtheit eines Films zu treffen.

4.1.1 Technologieauswahl

Spark SQL unterstützt jede Datenbank die JDBC unterstützt. Damit würden sich SQL Datenbanken für ein Spark Projekt eignen. Ebenso bietet Spark eine direkte Anbindung an die spaltenorientierten NoSQL Datenbanksysteme Hbase und Cassandra. Aufgrund der Datenakquisition, die während des Erstellungsprozesses der Software betrieben wird, ist eine NoSQL Datenbank zu empfehlen. Es werden tausende Tweets pro Tag gesammelt und abgespeichert. Damit ist die Geschwindigkeit der NoSQL Datenbanksysteme ein wichtiger Aspekt. Ebenso ist die Flexibilität der NoSQL Systeme von Vorteil, da in Zukunft neue Arten von Daten relevant werden können. Durch ein NoSQL-DBS wäre eine Erweiterung der Datenbank ohne Downtime

des Systems möglich. Hier würden sich Hbase und Cassandra demnach gut eignen. Die zu akquirierenden Tweets liegen im JSON Format vor. Daher wurde sich in dieser Umsetzung für ein dokumentenorientiertes NoSQL-DBS entschieden. In dokumentenorientierten DBS werden die Daten meist in JSON-Format abgespeichert, was zur Folge hat, dass keine Konvertierung der Datenformate vorgenommen werden muss.

Da die Datenbank stabil laufen soll, wurde sich bei der Recherche auf die großen Namen beschränkt. MongoDB und CouchDB wurden in Betracht gezogen. Da MongoDB im Gegensatz zu CouchDB einen Connector für Spark bereitstellt, wurde MongoDB gewählt. Dieser Connector ermöglicht es Daten aus der MongoDB direkt in RDDs für Spark umzuwandeln und andersherum [MongoDB, Inc. (2016)]. Durch diese Datenbankauswahl ist die Implementierung außerdem hochflexibel für eventuelle Änderungen in der Datenstruktur.

4.1.2 Datenmodell auf MongoDB angewendet

Das Datenmodell in JSON-Format für die MongoDB sieht wie folgt aus. Es wird 2 Collections geben:

1. Tweets Diese Collection enthält die abgespeicherten Tweets.

2. Movies In dieser Collection werden alle Informationen zu den Filmen gesammelt, die untersucht werden.

Listing 4.1: Tweets Collection JSON Aufbau

```
{
        _id: ObjectID,
        tweet_id: Long,
        film: Liste von Strings,
        text: String,
        created_at: String,
        user_id: Long,
        username: String
        user_language: String
}
```

_id ist die von MongoDB vergebene ID.

tweet_id ist die von Twitter vergebene ID des Tweets und kann später verwendet werden, um doppelte Tweets zu löschen oder zu ignorieren.

film ist eine Liste, der im Tweet angesprochenen Filme, um später die Tweets den Filmen zuzuordnen. Dabei werden hier die Titel der Filme genannt, die auch in der Movie-Collection angegeben werden.

text ist der eigentliche Text des Tweets, der später analysiert wird.

created_at ist das Datum, an dem der Tweet gesendet wurde und kann für verschiedene Zwecke verwendet werden.

user_id ist die von Twitter vergebene ID des Users. Mit Hilfe der User ID ist es möglich zum Beispiel nur Tweets anzusehen, die von verschiedenen Usern kommen und somit ein bereinigtes Meinungsbild zu schaffen.

user_language gibt die Sprache an, mit der der User für gewöhnlich tweetet.

Während im ER-Diagramm der User als eigene Entität angegeben wurde, werden die User-Eigenschaften im MongoDB Datenmodell mit in die Tweet-Dokumente gezogen. Dadurch können die Tweets nach direkt nach User gefiltert werden.

Listing 4.2: Movies Collection JSON Aufbau

```
{
        _id: ObjectID,
        imdb_id: String,
        title: String,
        release_date: String,
        tweetcount: Integer,
        pos_tweets: Integer,
        neg_tweets: Integer,
        estimated_success: String,
        opening_we_gross: double
}
```

Im Gegensatz zum ER-Modell werden Tweets direkt im Movie-Dokument gezählt. Wenn ein Tweet ankommt, wird er auf die Stimmung untersucht und das zu der Stimmung gehörende Attribut im Film inkrementiert. Dies erspart eine collectionübergreifende Abfrage.
Die Movie-Dokumente werden mit der Zeit gefüllt, sobald die jeweiligen Daten vorhanden sind. Es steht frei, wie viele Arten von Stimmungen genutzt werden, daher wäre es möglich weitere „sentiment_tweets“ Attribute hinzuzufügen. Ebenfalls ist es sinnvoll mehrere „estimated_success“ Attribute einzuführen, je nachdem welcher Algorithmus zur Berechnung genutzt wurde. Durch die Nutzung eines NoSQL-DBS sind diese Änderungen und Erweiterungen jederzeit möglich.

4.2 Kontextsicht

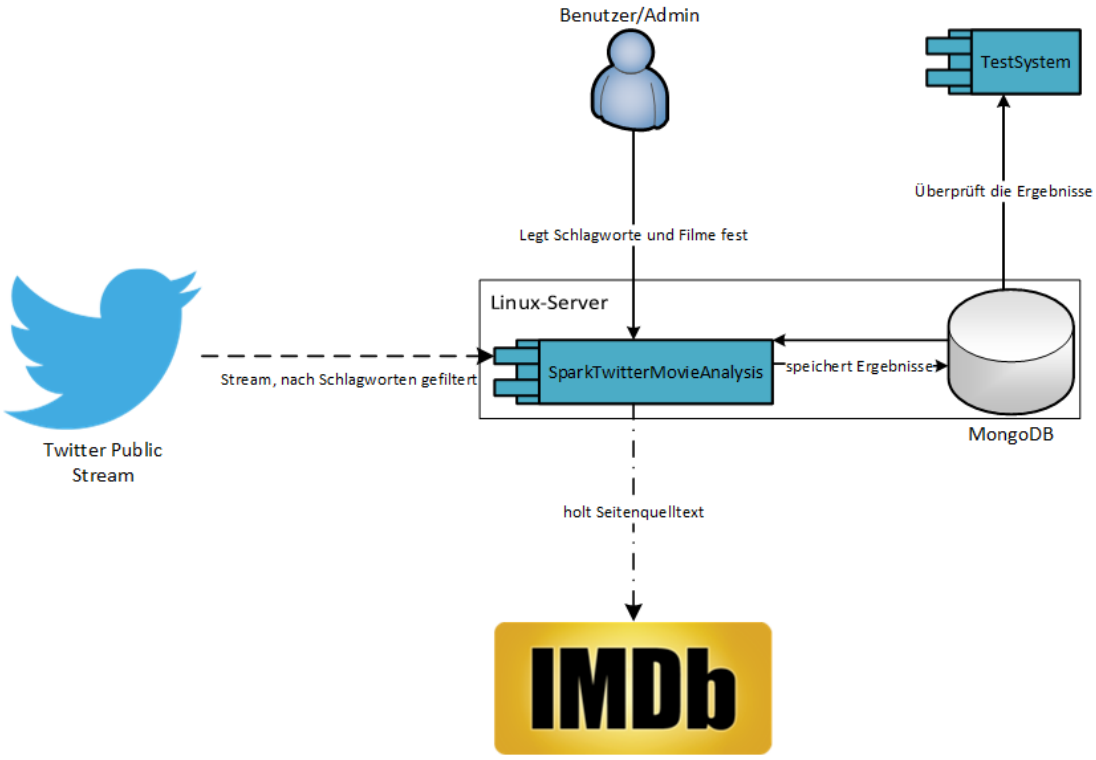


Abbildung 4.2: Kontextsicht

Das zu erstellende System, sowie die verwendete Datenbank werden in dieser Implementation auf einem Rechner laufen. Das System soll eigenständig die „Coming Soon“-Filme von IMDb auslesen und in der Datenbank speichern (Anforderung F5), sowie eine Datei erstellen, in der diese Filme aufgelistet sind, mitsamt der Schlagworte, die pro Film genutzt werden (Anforderung F1) - die Watchlist. Da das System allerdings nur bedingt automatisch diese Schlagworte erstellen kann, ist es dem Benutzer/Admin möglich diese Datei per Hand zu bearbeiten und so die Schlagworte und Filme selbst festzulegen. IMDb stellt keine offizielle API zur Verfügung, daher wird eine HTML-Parser API verwendet, um die gesuchten Seiten auszulesen. Das System wird sich mit dem Twitter Public Stream verbinden, der bereits über die in der Watchlist aufgeführten Schlagworte gefiltert ist. Dieser Stream wird mit Hilfe des Spark Frameworks gemapped, auf die Sentiment Analyse vorbereitet und letztendlich analysiert und ausgewertet (Anforderung F2). Die durch die Analyse erhaltenen Ergebnisse werden ebenfalls in der Datenbank gespeichert (Anforderung F3). Täglich werden dann die gesammelten

Daten ausgewertet und per ML Vorhersagen zu den Opening Weekend Einnahmen gemacht und abgespeichert (Anforderung F4). Liegen Ergebnisse in der Datenbank vor, werden sie durch ein Testsystem auf ihre Korrektheit überprüft. Dies soll außerdem einen Vergleich von verschiedenen ML-Algorithmen möglich machen.

4.3 Bausteinsicht

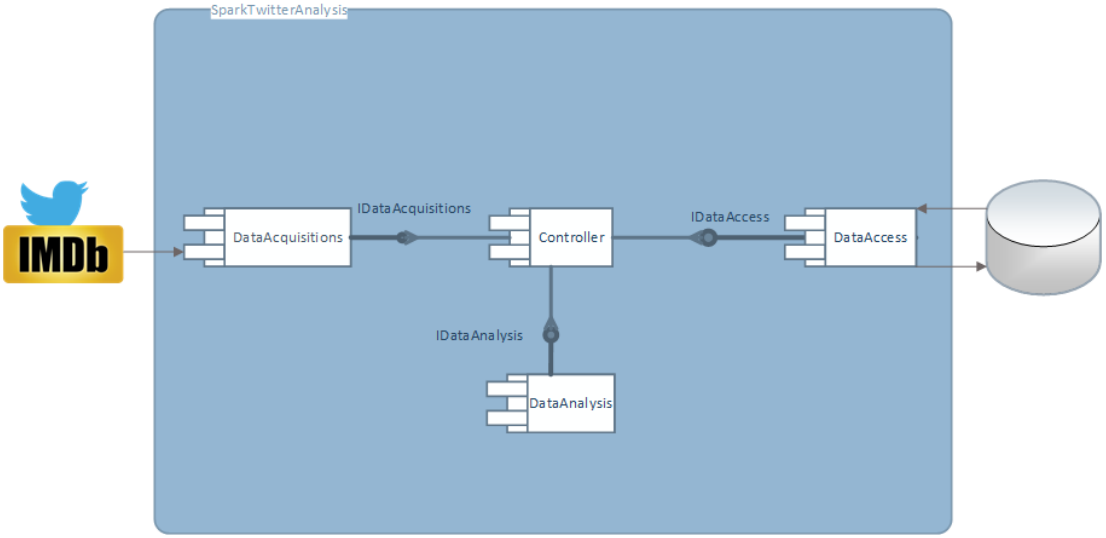


Abbildung 4.3: Bausteinsicht

Das System besteht aus 4 Komponenten.
„DataAcquisitions“ hat die Aufgabe, Daten zu akquirieren und bereitzustellen. Dabei werden die Daten so aufbereitet, dass sie von den anderen Komponenten verwendet werden können. Die Komponente greift über das Internet auf den Streaming Service von Twitter und die Webpräsenz von IMDb zu. „DataAnalysis“ ist die Komponente, die die akquirierten Daten weiterverarbeitet. In der Komponente werden alle Aufgaben des maschinellen Lernens zusammengefasst. Über die Schnittstelle kann auf Funktionalitäten zugegriffen werden, die die Sentiment-Analyse und Vorhersagen durchführen. „DataAccess“ ist die Anbindung an die Datenbank. Hier werden alle notwendigen Mittel bereitgestellt, um auf die Datenbank zuzugreifen und um Daten aus der Datenbank im gewünschten Format zu erhalten. „Controller“ koordiniert die Zusammenarbeit der Komponenten per Scheduling, erstellt die Watchlist und sorgt dafür, die übergeordneten Aufgaben zu erfüllen.

4.4 Komponenten im Detail

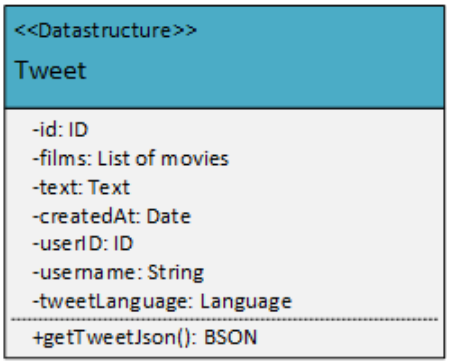


Abbildung 4.4: Datenstruktur Tweet

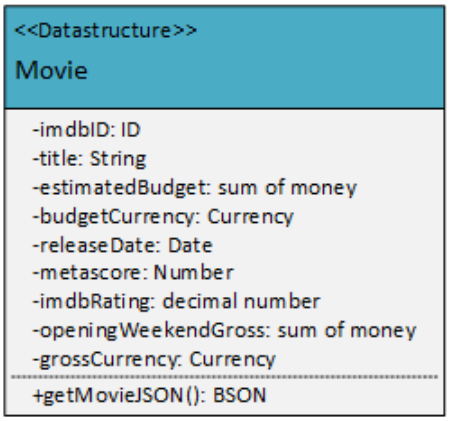


Abbildung 4.5: Datenstruktur Movie

Diese Datenstrukturen enthalten jeweils die akquirierten, relevanten Daten. Sie stellen eine Funktionalität bereit, die die Daten in ein für die MongoDB verständliches Format formatieren, sodass eine Speicherung einfach möglich ist. Ebenso ist es möglich per entsprechendem Konstruktor aus einem JSON Datensatz eins der jeweiligen Objekte zu erzeugen.

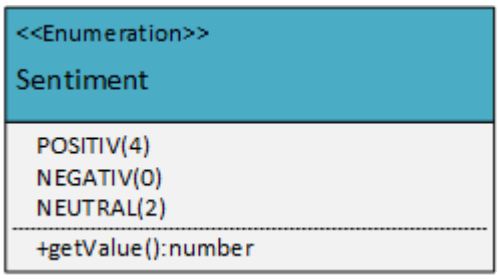


Abbildung 4.6: Sentiment Enumeration

Um effizient mit Stimmungen arbeiten zu können und in Zukunft potentiell mehr Stimmungsarten hinzufügen zu können, nutzen wir ein Sentiment-Enum.

4.4.1 DataAcquisitions

Diese Komponente hat die Aufgabe, benötigte Daten aus dem Internet zu akquirieren. Der zu untersuchende Twitter-Stream wird nach Schlagworten und auf relevante Daten gefiltert. Die von IMDb benötigten Daten werden akquiriert, in dem die relevanten Websites geladen und

mit Hilfe eines HTML-Parsers auf die gewünschten Daten gescannt werden. Die so erhaltenen Daten werden in der Movie-Datenstruktur zusammengefasst.

Der Twitter-Stream soll nach dem Start dauerhaft laufen. Die IMDb-Datenakquise muss im besten Fall nur ein Mal pro Film ausgeführt werden. Die Herangehensweisen der beiden Arten der Datenakquisition sind demnach stark unterschiedlich. Sie werden daher und aufgrund der Übersicht und Wartbarkeit jeweils als eigene Klasse implementiert.

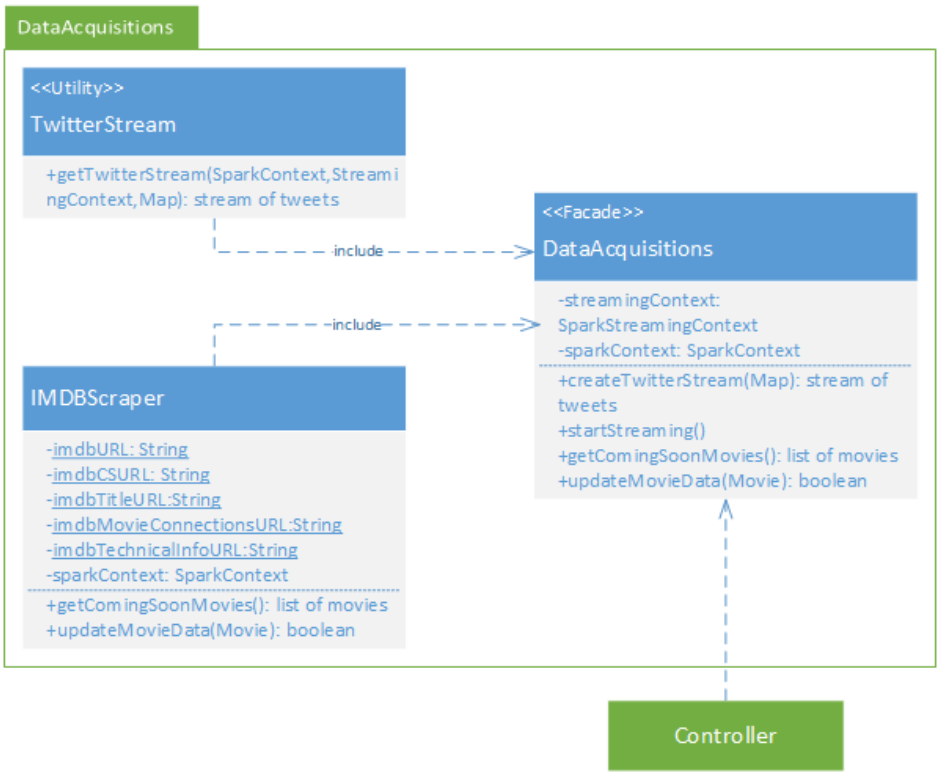


Abbildung 4.7: Die DataAcquisitions Komponente

Die Klasse DataAcquisitons ist die Schnittstelle, die der Controller nutzt. Sie gibt an, welche Arten und Methoden der Datenakquisition der Controller ausführen kann. Sie nutzt dafür die Klassen TwitterStream und IMDBScraper, die jeweils die Funktionalitäten zum Zugriff auf den Twitter-Stream bzw. IMDb kapseln. Durch diese Fassaden-Struktur ist es möglich, in Zukunft weitere Arten von Daten/-akquisition hinzuzufügen, ohne das System komplett überarbeiten zu müssen. Die Klasse TwitterStream gibt mit der Methode „getTwitterStream“ einen für Spark nutzbaren Stream von Tweets zurück. Diese Tweets sind Objekte des Typs Tweet. Diese Methode wird in „createTwitterStream“ aufgerufen und bekommt als Parameter eine Map übergeben, in der alle Filme und die dazugehörigen Suchworte aufgelistet sind. IMDBScraper

bietet drei Methoden an. Zum einen können alle im nächsten Monat erscheinenden Filme als Liste oder direkt als Spark-RDD beschafft werden. Zum anderen können die Daten eines Films aktualisiert werden. Dies ist nötig, falls sie bei der Beschaffung nicht vollständig waren oder um sie auf Änderungen zu prüfen. Geupdatet werden nur die Daten, die im Datentyp Movie gesammelt werden.

4.4.2 DataAccess

DataAccess bietet Funktionalitäten zum Zugriff auf die zugrundeliegende (MongoDB-) Datenbank. Dafür gibt es eine Klasse „MongoAccess“, die die DataAccess-Schnittstelle für eine MongoDB-Datenbank implementiert. Die Schnittstelle beinhaltet alle notwendigen Zugriffs-

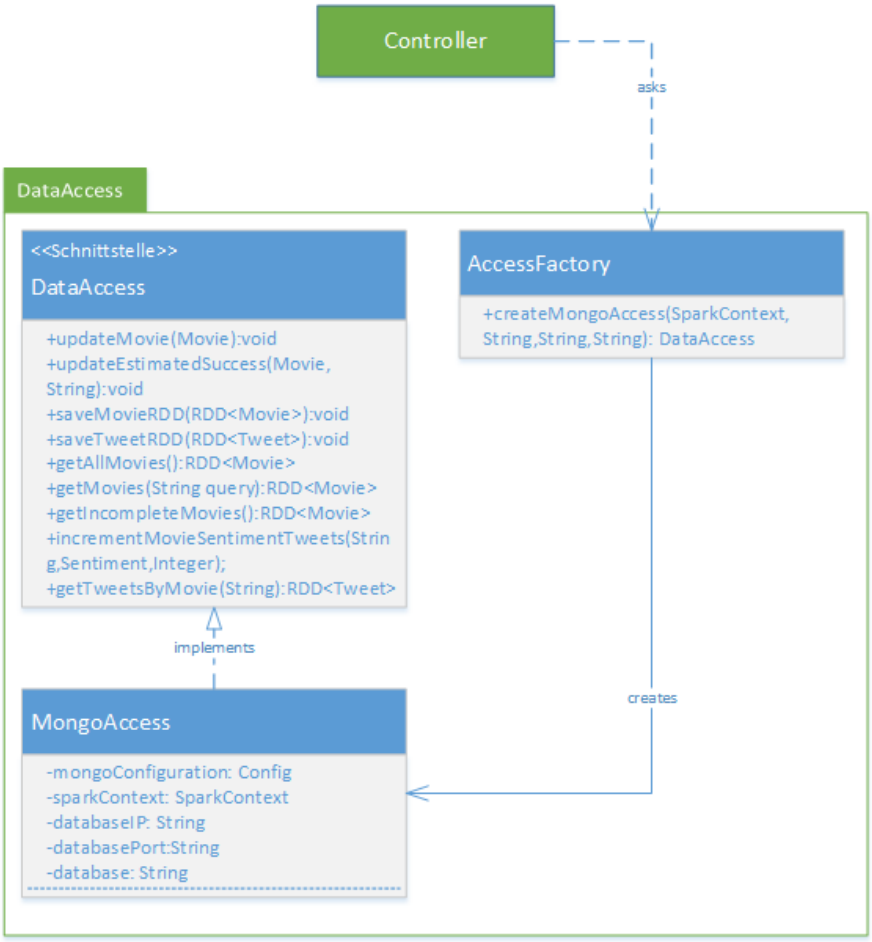


Abbildung 4.8: Die DataAccess Komponente

funktionen. Notwendig sind Funktionen, die exakt die Daten liefern und abspeichern, die später bei der Verarbeitung, Aktualisierung und Analyse benötigt werden. Zusätzlich zu den Funktionalitäten muss jede implementierende Klasse ihre Member-Variablen so definieren, dass ein Zugriff auf die jeweilige Datenbank möglich wird. Die Erstellung des vom Controller gewünschten DataAccess Typs geschieht über eine Factory. Durch das verwendete Factory-Pattern ist es in Zukunft möglich, die DataAccess-Komponente für viele verschiedene Datenbanken kompatibel zu machen, ohne Code im Controller ändern zu müssen.

4.4.3 DataAnalysis

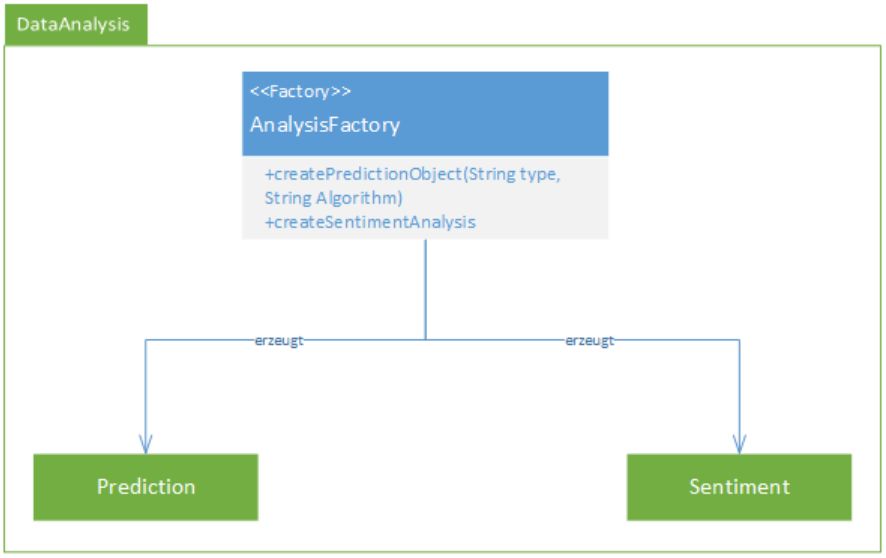


Abbildung 4.9: DataAnalysis Komponente: Übersicht

Alle in dieser Komponente enthaltenen Klassen müssen serialisierbar sein. Die hier verwendeten Klassen und ihre Funktionen werden innerhalb eines Spark-Kontexts, also potentiell verteilt im Cluster ausgeführt. Um dies tun zu können, sendet Spark eine Kopie der Klasse oder des Objekts an die Worker und stellt vor Start der Aufgabe sicher, dass alle verwendeten Klassen serialisierbar sind.

Sentiment Analyse

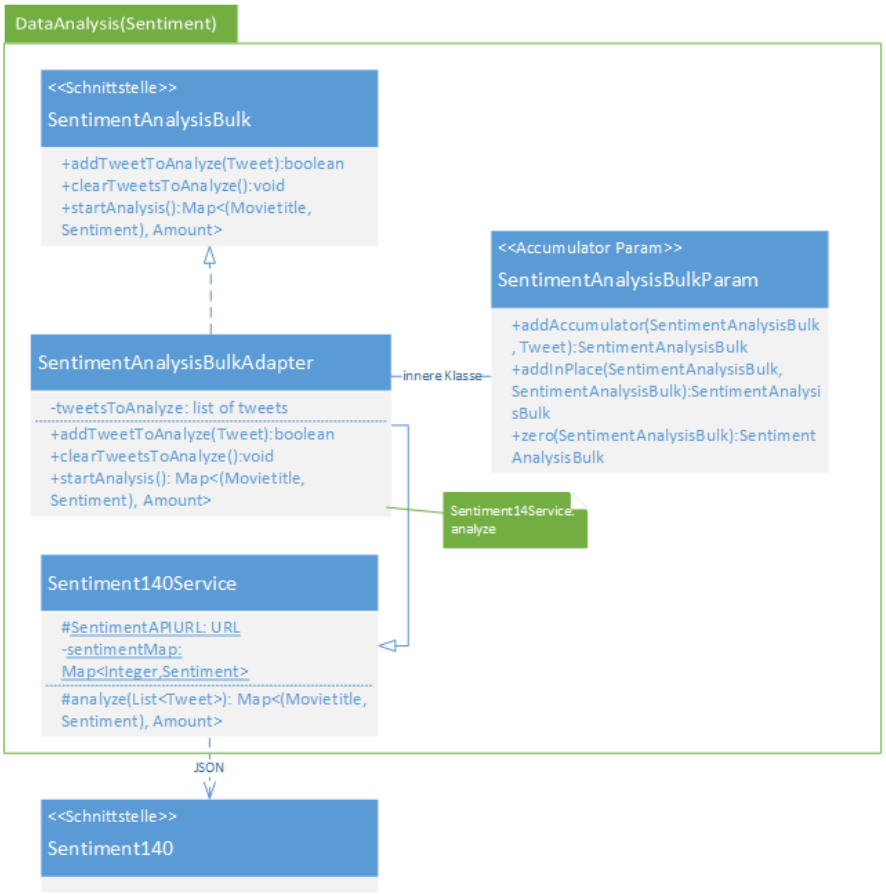


Abbildung 4.10: DataAnalysis Komponente: Sentiment Analyse

Ein wichtiger Bestandteil des Systems ist die Analyse der Stimmung, die in den gesammelten Tweets übertragen wird. Diese Auswertung macht einen wichtigen Teil der Daten aus, die in die Vorhersage genutzt werden. Da eine Sentiment Analyse von Tweets komplex ist und vom Zeitaufwand eine eigene Arbeit rechtfertigen würde - vor allem, wenn sie qualitativ mit der Konkurrenz mithalten soll - wird hier eine API genutzt. Um einen potentiellen Austausch der verwendeten APIs/Bibliotheken zu erleichtern wird hier ein Adapter Pattern angewendet.

Technologieauswahl Es existieren einige APIs und Bibliotheken, die Sentiment Analyse implementieren. Um eine Auswahl zu treffen wurden vier APIs und eine eigene Implementation miteinander verglichen. Eine eigene Implementation würde mit Hilfe der Spark MLlib umgesetzt werden. Um allerdings eine zuverlässige Sentiment Analyse zu ermöglichen, würden große

Mengen an Daten benötigt werden, sowie viele Faktoren berücksichtigt werden müssen. Diese Faktoren umfassen unter anderem die Gewichtung bestimmter Worte in bestimmten Fällen. Beispielsweise würde der Begriff „Civil War“ wahrscheinlich negativ behaftet sein, was in den meisten Fällen zutrifft. Der jüngst erschienene Film „Captain America: Civil War“ allerdings wurde von der Allgemeinheit sehr positiv aufgenommen, würde aber durch die negative Behaftung des Begriffs überwiegend negativ bewertete Tweets erhalten. Solche und ähnliche Faktoren müssen bei einer Implementation berücksichtigt werden. Da dies zu einem hohen Zeitaufwand führt, kann hier nur eine Schätzung zu den verschiedenen betrachteten Kategorien gemacht werden. Aufgrund dieser Ungewissheit wird die eigene Implementation auch nicht weiter in Betracht gezogen und der Vergleich zwischen den ausgewählten APIs in den Fokus gesetzt.

	Eigene Impl.	Sentiment140	StanfordCoreNLP	Semantria	AlchemyAPI
Preis	Kostenlos	Kostenlos	Kostenlos	Free-Trial | ab 999€/Monat	Kostenlos | 0,007 $ USD/Call
# API Calls	unbegrenzt	kein konkretes Limit	unbegrenzt	15.000 einmalig | ab 100.000/Monat	1000/Tag | unbegrenzt
Twitteroptimiert	Ja	Ja	Nein	Ja	Nein
Korrektheit	niedrig-hoch	15/30	15/30	16/30	18/30
Zeitaufwand	hoch- sehr hoch	gering	mittel	mittel	mittel
Sprachen	theoretisch alle	Eng/Es	Eng	alle relevanten Sprachen	Eng,De,Fr,Ru,Es,It,Ar,Por

Abbildung 4.11: Vergleich von Sentiment Analysis APIs

Getestet und recherchiert wurden die APIs „Sentiment140“, „StanfordCoreNLP“, „Semantria“ und „AlchemyAPI“. Zunächst gehen wir auf die Key Performance Indikatoren ein. Diese sind für diesen Anwendungsfall die Korrektheit der Analyse, die unterstützten Sprachen, sowie die spezifische Optimierung für eine Nutzung mit Twitter. Der letzte Punkt ist besonders wichtig, aufgrund Begrenzung eines Tweets auf 140 Zeichen. Diese Begrenzung führt zu Kurzschreibweisen und Emoticon-Nutzung, sowie falschem Satzbau und Zeichensetzung. Diese Dinge sollten besonders behandelt werden.

Für Twitter optimierte Implementierungen sind von den ausgewählten APIs „Sentiment140“ und „Semantria“. „StanfordCoreNLP“ und „AlchemyAPI“ sind für die Analyse von geschriebener bzw. natürlicher Sprache entwickelt. Dies spiegelt sich direkt im Korrektheitstest wieder. Es wurden 30 Tweets untersucht. Einige davon sind in korrekter Sprache mit ganzen Sätzen geschrieben, andere enthalten Abkürzungen, Hashtags und Nutzernamen. Wie in Abbildung 4.11 zu sehen unterschieden sich die Ergebnisse der APIs nicht besonders. Alle haben eine Trefferquote von 50-60%. Hierbei fällt allerdings auf, dass die AlchemyAPI besonders präzise bei natürlich geschriebenen Tweets ist und diese mit 90% Sicherheit richtig einordnet. Sobald die Tweets twitterspezifische Konstrukte und Abkürzungen enthalten fällt die Präzision allerdings rapide ab. Das gleiche gilt für StanfordCoreNLP, mit dem Unterschied, dass diese API

von Beginn an nicht ganz so präzise ist. Entgegen der Erwartung ist auch Semantria trotz Twitteroptimierung nicht sonderlich präzise.

Die meisten Sprachen unterstützen Semantria und AlchemyAPI. Beide APIs unterstützen alle weit verbreiteten Sprachen, wobei Semantria hier noch einige Sprachen wie Koreanisch und Japanisch anbietet. Sentiment140 unterstützt Englisch und Spanisch, die zusammen ca. 50% aller Tweets ausmachen [Statista.com (2013)]. StanfordCoreNLP unterstützt lediglich die englische Sprache.

Da sich keine der APIs in Korrektheit absetzt, ist ausschlaggebend welche Sprachen und damit wie viele Tweets analysiert werden können. Aufgrund diese Tatsache würde sich Semantria oder AlchemyAPI am besten eignen. Diese beiden APIs bringen allerdings hohe Kosten mit sich. Für Semantria ist eine kostenlose Trial Version verfügbar, die aber nur insgesamt 15.000 Anfragen zulässt. Auf Dauer ist Semantria damit keine Option. Alchemy bietet in der kostenlosen Variante monatlich 1000 Anfragen an. Trotz möglicher Bulk-Anfragen ist diese Anzahl zu gering um für dieses Projekt zweckdienlich zu sein.

Letztendlich wurde Sentiment140 ausgewählt, da diese API zwei Sprachen unterstützt und kostenlos unbegrenzt nutzbar ist. Mit Sentiment140 ist es möglich bis zu 5000 Tweets pro Anfrage im Bulk zu analysieren. Sentiment140 weist außerdem darauf hin, dass Einzelanfragen in hohen Ausmaßen zu einer Sperrung des Zugangs führen. Daher muss die Implementierung eine Bulk-Analyse vorsehen. [AlchemyAPI (2016); Lexalytics (2016); Christopher D. Manning (2014); Sentiment140 (2016)]

SentimentAnalysisBulkAdapter Diese Klasse ist der Adapter für die zu verwendenden Technologien.

Da hierbei eine Bulk-Analyse von Tweets ermöglicht werden soll, verwaltet der Adapter eine Liste von Tweets. Diese Liste wird von allen im Spark-Kontext verwendeten Workern/Threads gefüllt. Um einen - auch über Rechner hinweg - gemeinsamen Datenzugriff zu ermöglichen, stellt Spark das „Accumulator“-Konstrukt zur Verfügung. Alle Worker können auf gemeinsam auf einen solchen Akkumulator zugreifen und so verteilt auf einem Objekt arbeiten.

Damit Spark weiß, was in dem Akkumulator gespeichert oder hochgezählt werden soll, muss ein „AccumableParam“ definiert werden. Dieser Parameter gibt an, von welchem Typ der Akkumulator ist und von welchem Typ die Objekte sind, die man in diesem speichert. [Apache Software Foundation (2016d)]

Dieser AccumableParam ist hier „SentimentAnalysisBulkParam“ und wird als innere Klasse des SentimentAnalysisBulkAdapters definiert. Sind alle zur Verfügung stehenden Tweets in der Liste, so kann die Anfrage mit startAnalysis() gestartet werden. Diese Methode leitet die

Analyse an die verwendete Technologie weiter, die dann die Analyse ausführt. Das Ergebnis der Methode startAnalysis() ist eine Map deren Key jeweils ein Tupel aus Film und Sentiment und deren Value die Gesamtanzahl der dementsprechenden Bewertungen ist.

Sentiment140Service Die Klasse ist die Implementierung, die nötig ist, um die API von Sentiment140 zu nutzen. Sie kann eine Anzahl von Tweets an die API senden und so Bewertung des Sentiments vornehmen. Vom SentimentAnalysisBulkAdapter bekommt diese Klasse eine Liste an Tweets, die analysiert werden sollen. Die API ist in der Lage höchstens 5000 Tweets pro Anfrage zu verarbeiten, daher muss innerhalb der Klasse dafür gesorgt werden, dass nur diese begrenzte Anzahl an Tweets pro Anfrage verwendet werden. Die Tweets werden dann per JSON an die API geschickt. Der Aufbau des JSONs, das die API verarbeiten kann ist wie folgt:

Listing 4.3: Aufbau des Sentiment140 JSONs

```
{"data":[
{"text":text of tweet,
 "film":list of movies,
 "query":movies,
 "topic":"movies"}
 ,...]}
```

Von diesen Attributen werden lediglich "text", "queryünd "topic"von der API ausgewertet. Alle zusätzlich mitgeschickten Attribute werden von der API unverändert zurückgeschickt [Sentiment140 (2016)]. Das Attribut "filmïst dazu da, das Ergebnis auf den jeweiligen Film zurückverfolgen zu können. Als Antwort erhalten wir ein JSON, das so aufgebaut ist, wie jenes, das wir geschickt haben. Das Ergebnis JSON enthält zusätzlich das Attribut "polarity", das angibt, welche Stimmung der Tweet hat. Dabei steht 4 für positiv, 2 für neutral und 0 für negativ. Um eine Abbildung dieser Werte auf ein Sentiment effizient zu ermöglichen, besitzt diese Klasse eine Map, die diese Abbildung vornimmt. Zuletzt werden die Ergebnisse für Film und die dazugehörige Stimmung zusammengefasst, aufaddiert und als Ergebnis zurückgegeben. Das Ergebnis ist demnach eine Map, die Film und Stimmung auf eine Anzahl der dazugehörigen Bewertungen abbildet.

Prediction

Der Prediction Teil der DataAnalysis Komponente ist beliebig komplex, da sehr viele verschiedene Algorithmen implementiert werden können. Es gibt zwei Arten der Vorhersage. Zum einen gibt es die Klassifikation, die Daten in bestimmte Klassen einteilt. Die Klassen

werden vom Enum „MulticlassClassification“ vorgegeben. In diesem System wird die Multiclass Classification genutzt, weil eine binäre Klassifizierung zur Vorhersage der Einspielergebnisse eines Films zu unpräzise wäre und keinen Mehrwert bringt.

Zum anderen gibt es die Regression, die versucht einen konkreten Wert zu ermitteln. Diese Art der Vorhersage ist bei einer kleinen Datenmenge, wie sie in diesem Projekt leider gegeben ist, sehr unpräzise. Dennoch wird sie implementiert, um sie in Zukunft effektiv nutzen zu können.

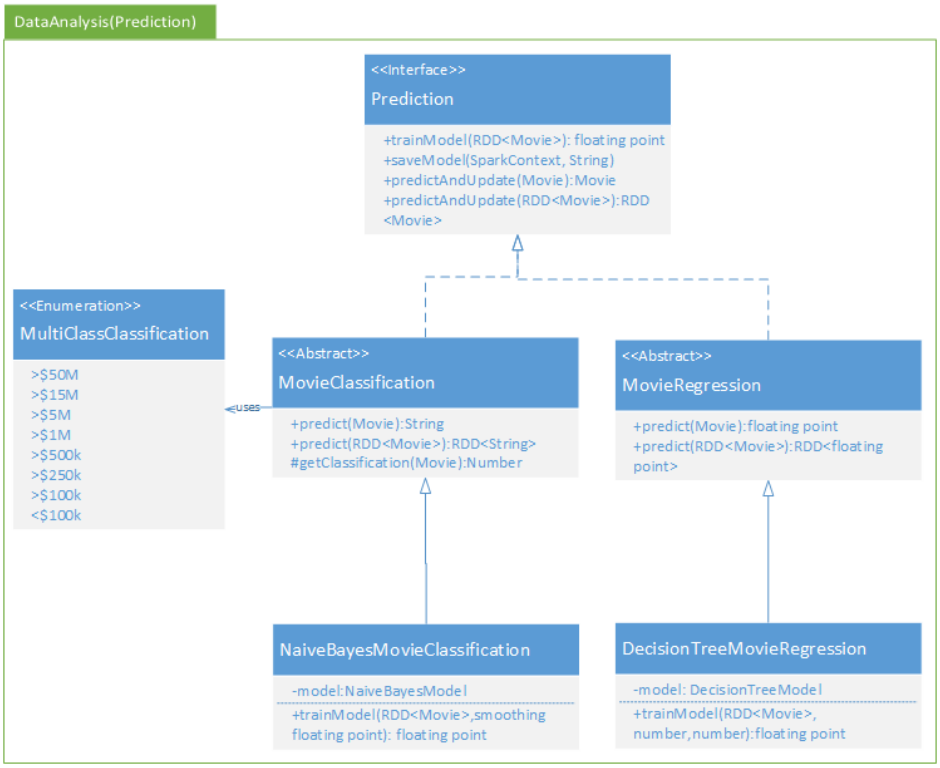


Abbildung 4.12: DataAnalysis Komponente: Übersicht

Die Prediction Schnittstelle legt die allgemeinen Methoden fest, die alle Algorithmen implementieren müssen. Diese Methoden sind das Speichern und Laden von vorhandenen und gebildeten Modellen, sowie das Trainieren und Anwenden eines Modells auf ein oder mehrere Filme. Beim Training eines Modells erhält der Nutzer als Ergebnis die Genauigkeit des generierten Modells. Der Trainingsmethode wird ein RDD aus Filmen als Parameter überreicht. Es empfiehlt sich hier alle Filme zu nutzen, zu denen bereits Daten zu den Opening Weekend-Einnahmen zur Verfügung stehen. Es ist aber auch möglich nur eine Teilmenge zu nutzen. Das erste trainierte Modell, wurde aus den Daten erzeugt, die vorab gesammelt wurden. Da

diese Datenmenge sehr klein ist, empfiehlt es sich regelmäßig neue Modelle zu trainieren. „PredictAndUpdate“ ist die Methode, die das jeweilige Modell auf einen Film anwendet. Nach Anwendung werden die dazugehörigen Attribute im Movie Objekt auf den errechneten Wert geändert. Die abstrakten Klassen MovieClassification und MovieRegression geben spezifischere Methoden für die Anwendung eines Algorithmus der jeweiligen Kategorie vor. Für erbende Klassen stellt MovieClassification eine Methode „getClassification“ zur Verfügung, die die Klassifizierung eines Films vornimmt und die für Berechnungen notwenige dazugehörige Zahl zurückgibt. Die Klassifizierung bezieht sich hierbei nur auf Filme, die bereits Daten über die Opening Weekend-Einnahmen besitzen. Diese Methode ist demnach für die Erstellung eines Modells notwendig. Des Weiteren geben die abstrakten Klassen weitere „predict“ Methoden vor. Diese Methoden geben ein direktes Ergebnis zurück. Bei der Klassifikation handelt es sich dabei um eine Zeichenkette, die die Klasse benennt. Im Falle der Regression ist dies die tatsächlich berechnete Summe der Einnahmen. Die Klassen, die die Algorithmen umsetzen und von den abstrakten Klassen erben, bieten zusätzlich eine weitere Trainingsmethode an. Diese Methode soll es ermöglichen die Trainingsalgorithmen zu verfeinern, indem der Nutzer je nach Algorithmus unterschiedliche Parameter übergibt. Um eine bessere Vorhersage treffen zu können, wird für jede Berechnung die Anzahl der Tweets (tweetcount), sowie die Anzahl an positiven und negativen Tweets, durch die Anzahl der Tage geteilt, die seit Beginn der Beobachtung eines Films vergangen sind. Dadurch bekommen wir eine durchschnittliche Menge an Tweets pro Tag.

4.4.4 Controller

Die Controller-Komponente sorgt dafür, dass alle Komponenten zusammenarbeiten. Sie definiert zeitlich geschaltete Jobs, die in unterschiedlichen Abständen gestartet werden. Wie diese Jobs im genauen ablaufen, wird in Kapitel 4.4.5 Laufzeitsicht beschrieben. Die Controller Komponente enthält die Main-Methode und startet die Ausführung des Programms.

Technologieauswahl

Um eine automatisch gesteuerte Zeitschaltung zu implementieren, wird eine Scheduling-Library verwendet. Hier wurden vier Implementierungen untersucht: Cron-Jobs, Java Timer, Java ScheduledExecutorService und Quartz.

Cron Cron ist die zeitliche Steuerung von Abläufen auf einem Linux-System. Da unser Programm auf einem Linux-System laufen wird, wäre dies eine Möglichkeit das Scheduling zu implementieren. Der Nachteil ist, dass unser System in verschiedene Teil-Jars aufgeteilt werden

müsste. Für jeden Job würde eine Jar erstellt werden müssen. Dies ist für diese Architektur nicht praktikabel und erhöht den Wartungsaufwand. Die Bindung an ein Linux-System ist ebenfalls potentiell problematisch, da sie den Vorteil von Java aushebelt, plattformunabhängig laufen zu können. Ebenso problematisch ist die Tatsache, dass der Cron-Job auf jedem Rechner per Hand erstellt werden muss.

Java Timer/ScheduledExecutorService Die java-interne Scheduling Implementation „Timer“ ermöglicht es, die Ausführung einer Aufgabe bis zu einem festgelegten Zeitpunkt zu verzögern oder eine Aufgabe periodisch auszuführen. Nachteilig ist, dass Timer anfällig für Änderungen an der Systemuhr ist. Dies kann zu unerwünschtem Verhalten des Ablaufs führen. Außerdem hat ein Timer nur einen Thread zur Ausführung, was bei langlaufenden Aufgaben zu Verzögerung von anderen Aufgaben führen kann. Tritt ein Fehler in einer Aufgabe des Timers auf, so beendet sich der gesamte Timer und damit alle zukünftigen Aufgaben.

ScheduledExecutorService ist eine neue Implementierung von Timer, die seit Java 5 existiert. Sie behebt einige der Probleme von Java Timer und macht es möglich eine nahezu unbegrenzte Anzahl von Jobs zu schedulen. ScheduledExecutorService arbeitet mit relativer Zeit und ist daher nicht anfällig auf Änderungen in der Systemuhr. Des Weiteren ist es möglich Fehler, genauer Runtime Exceptions, zu behandeln. Ein Fehler beendet nur die Ausführung der verursachenden Aufgabe, alle weiteren Aufgaben werden weiterhin ausgeführt. Es ist nicht möglich genaue Zeitpunkte der Ausführung anzugeben oder eine Ausführung auf einen bestimmten Wochentag oder ähnliches zu legen. Dies muss per Rechnung mit relativer Zeit händisch durchgeführt werden.

Quartz Quartz ist eine Scheduling-Library für Java. Mit Hilfe der Library ist es möglich, genaue Daten, Wochentage, Zeitabstände, usw. anzugeben, zu denen ein vorher spezifizierter Job ausgeführt werden kann. Ebenso hat man mit Quartz die Möglichkeit, auf Fehler im Job-Ablauf zu reagieren und im Fehlerfall den Job neu zu starten. Quartz ist sehr mächtig und bietet viele nützliche Funktionen. Problematisch ist der große Overhead, der durch die Mächtigkeit entsteht. Es ist schwierig Übersicht über alle Funktionalitäten zu gewinnen. Außerdem ist Quartz relativ aufwändig zu konfigurieren. Es wird hierbei zur Konfiguration mit aufwändigem XML gearbeitet. Ebenfalls ist es möglich die Konfiguration in einer Datenbank abzuspeichern und daraus zu laden. Diese Funktionalitäten sind gut für große Unternehmen und langlaufende Projekte, da dies ein unternehmensweit festgelegtes Scheduling ermöglicht. Für diese Arbeit ist der Aufwand sehr hoch und die Funktionalität, obwohl sie praktisch ist, nicht notwendig.

Es wurde sich für den java-internen ScheduledExecutorService entschieden, da dieser simpel einzusetzen ist und alle notwendigen Funktionalitäten, wenn auch über Umwege, anbietet. Durch die low-levelige Implementierung ist er außerdem sehr flexibel. Ebenfalls ist dadurch sichergestellt, dass keine Dependency Probleme entstehen, die durch ein großes Framework wie Spark, in Verbindung mit einem weiteren Framework wie Quartz auftreten können.

4.4.5 Laufzeitsicht

Zur Laufzeit wird es ein paar Abläufe geben, die im Folgenden näher erläutert werden. Alle hier gezeigten Abläufe werden automatisch durch ein Scheduling-System gesteuert.

GetComingSoonMovies

GetComingSoonMovies ist ein Ablauf, der bei Systemstart und dann wöchentlich einmal ausgeführt wird. Dabei wird die IMDb-Webseite auf Filme durchsucht, die im nächsten Monat erscheinen werden.

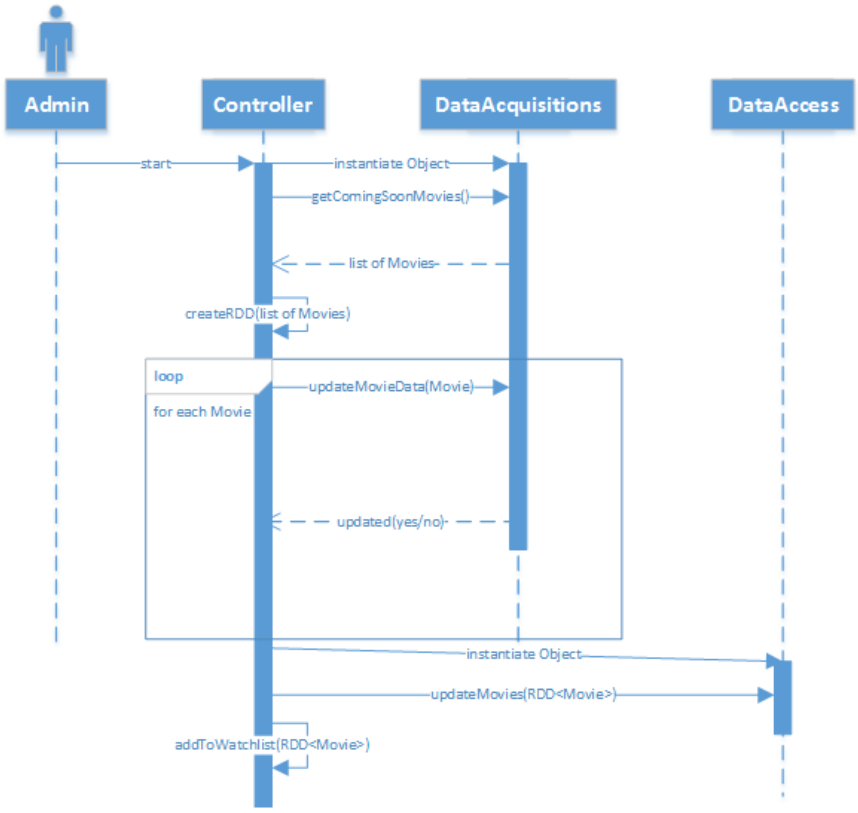


Abbildung 4.13: Der Ablauf des GetComingSoonMovies-Jobs

Nachdem das System vom Admin gestartet wurde, wird die DataAcqusitions-Komponente instanziiert und die demnächst erscheinenden Filme durch den IMDb-Scraper abgefragt. Die dadurch erhaltenen Filme werden an die Controller Komponente zurückgegeben. Dort werden sie in das RDD Format übertragen, um eine effizientere Verarbeitung durch Spark- Routinen zu ermöglichen. Da die durch getComingSoonMovies() erhaltenen Movie-Objekte noch unvollständig sind und nur Kerndaten (IMDb-ID, Releasedate, Title) enthalten, wird daraufhin für jedes der im RDD vorhandenen Movie-Objekte ein Update angefragt, das wiederum per IMDb-Scraper ermittelt wird. Für jeden Film gibt es jeweils eine Rückmeldung, ob das Objekt nun neue Daten enthält. Sind alle aktuell vorhandenen Daten ermittelt, werden die Movie-Objekte durch die DataAccess-Komponente in der Datenbank gespeichert. Dazu muss ein Objekt des jeweiligen DataAccess-Typs (in diesem Fall MongoAccess) instanziiert werden. Die erzeugten Objekte bleiben aktiv, um für spätere Jobs verwendet werden zu können.
Zuletzt werden alle durch diesen Ablauf gesammelten Filme in der Watchlist gespeichert, um die Filme, die durch den Stream überwacht werden, festzulegen.

UpdateMovies

Dieser Job wird täglich ein Mal durchgeführt. Er nimmt sich alle Filme und überprüft sie auf geänderte Daten und aktualisiert sie, wenn nötig.

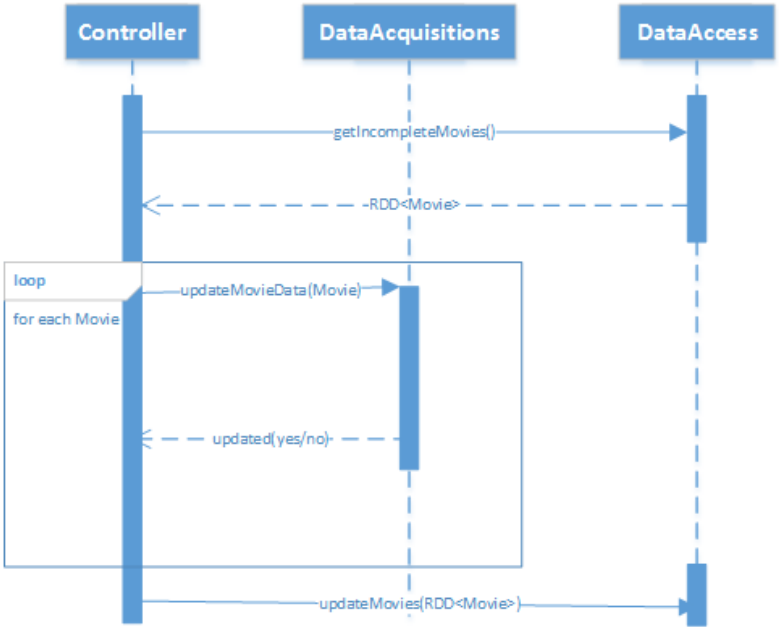


Abbildung 4.14: Der Ablauf des UpdateMovies-Jobs

Über die DataAccess-Komponente holt sich der Controller die relevanten Daten. Dabei wird die Methode „getIncompleteMovies()“ genutzt, die nur die Filme zurückgibt, die noch keine Daten über die tatsächlichen Opening Weekend-Einnahmen enthalten und daher noch aktiv verfolgt werden müssen. Andere Filme sind nicht relevant, da sie bereits vollständige Daten enthalten. Diese Filme werden dann durch die DataAcquisition-Komponente wie gewohnt aktualisiert. Die aktuellen Daten werden dann in der Datenbank abgespeichert bzw. aktualisiert.

CreateStream

Der CreateStream-Job wird täglich einmal ausgeführt. Dabei wird überprüft, welche Filme mit welchen Suchworten aktuell beobachtet werden und ein neuer dementsprechender Twitter-Stream wird gestartet. Zunächst holt sich die Controller-Komponente die aktuelle Watchlist.

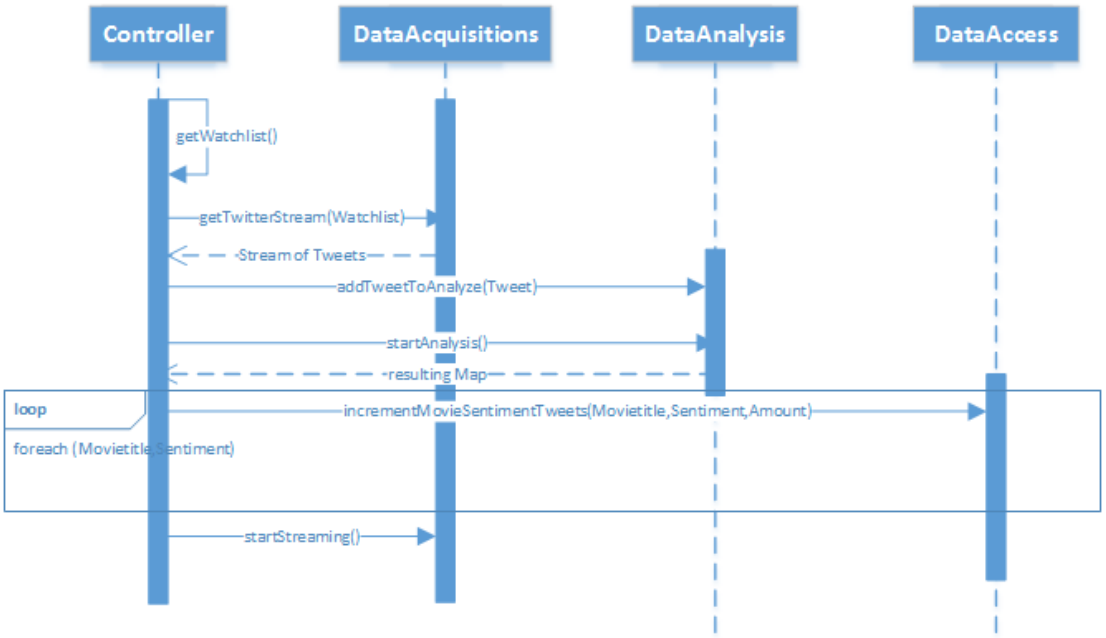


Abbildung 4.15: Der Ablauf des CreateStream-Jobs

Daraus wird eine Map erstellt. Es werden die Schlagworte auf die zugehörigen Filme gemapped. Diese Map wird an die DataAcquisitions-Komponente übergeben, die daraus den Stream erstellt. Das darauffolgende Vorgehen scheint unlogisch, ist aber bedingt durch die Art und Weise, wie Spark funktioniert. Als nächstes wird nicht der Stream gestartet, sondern definiert, wie mit dem Stream umgegangen wird. Es wird nun definiert, was mit den im Stream gesammelten Daten passieren soll. Er nachdem diese Definition abgeschlossen ist, kann der Stream gestartet

werden. Der Grund dafür ist, dass der Stream verteilt bearbeitet wird und das Vorgehen an alle Worker übermittelt werden muss. Es muss sozusagen von vornherein festgelegt werden, wer was zu tun bekommt.

Nach Erstellung des Streams wird also festgelegt, dass alle im Stream ankommenden Tweets an die DataAnalysis-Komponente gesendet werden. Sobald genug Tweets gesammelt wurden oder nach einer festgelegten Zeit, wird die Analyse vom Spark Master gestartet und ein Resultat als Map zurückgegeben. Der Inhalt dieser Map wird dann an die DataAccess-Komponente übergeben und die Daten in der Datenbank werden aktualisiert. Nachdem dieser Ablauf beschrieben wurde, wird der Stream gestartet.

PredictMovies

Der PredictMovies-Job ist der Job, der das Resultat des Systems erzielt. Er wird täglich einmal ausgeführt und aktualisiert die Vorhersage für jeden Film, der sich in der Watchlist befindet und über den ausreichende Daten zur Verfügung stehen.

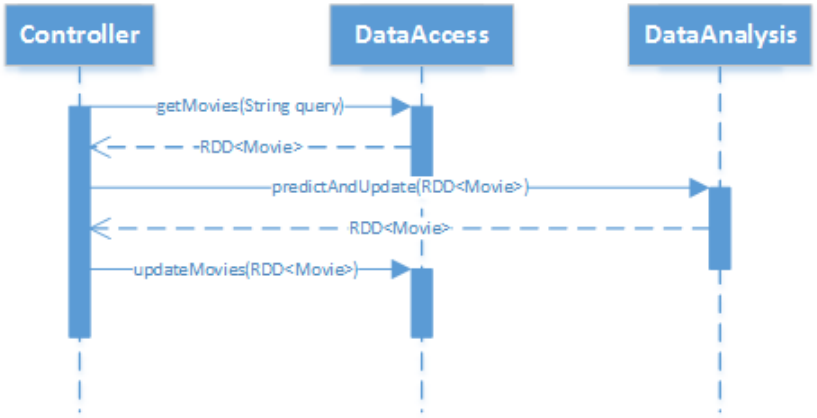


Abbildung 4.16: Der Ablauf des PredictMovies-Jobs

Zunächst holt sich der Controller per DataAccess alle Filme aus der Datenbank, die in der Watchlist aufgeführt sind. Diese Filme werden dann durch die DataAnalysis-Komponente untersucht. Das Ergebnis ist ein RDD von Filmen, die eine neue Vorhersage durch den jeweils gewählten Algorithmus erhalten haben. Diese Filme werden dann durch die DataAccess-Komponente in der Datenbank aktualisiert. Dieser Job ist der allgemeine Fall. Im laufenden Betrieb wird die predictAndUpdate-Methode für jeden gewünschten Algorithmus aufgerufen.

5 Realisierung und Tests

In diesem Kapitel werden die Realisierung und vor allem die aufgetretenen Probleme während der Realisierung beschrieben. Danach wird erklärt, wie beim Testen vorgegangen wurde.

5.1 Realisierung

Aufgrund der „versteckten“ Komplexität des Spark-Frameworks und der Unerfahrenheit in der Nutzung des Map-Reduce Patterns, ergaben sich im Laufe der Realisierung des Öfteren Fragen. Die gefundenen Antworten führten immer wieder zu kleineren Änderungen im Entwurf. Versteckte Komplexität bedeutet hierbei, dass die Nutzung des Frameworks auf Papier recht simpel erscheint, jedoch in der Praxis einige Probleme aufwirft, die meist aufgrund der potentiellen Verteilung der Ausführung von Aufgaben auftreten. Ein Beispiel ist der in der Dokumentation nicht angegebene Fakt, dass alle im Spark-Kontext zu nutzenden Datentypen und Objekte serialisierbar sein müssen. Der Grund dafür ist recht offensichtlich. Soll etwas verteilt im Cluster berechnet werden, so müssen Daten von einem Rechner zum Nächsten verteilt werden. Da die Nutzung im Cluster nicht vorgesehen war, wurde dieser Fakt nicht in Betracht gezogen und führte dazu, dass viele nicht-serialisierbare Klassenobjekte im Entwurf genutzt wurden. Spark prüft jedoch vor jeder Ausführung, ob alle potentiell verteilt genutzten Daten serialisierbar sind. Dies bedeutete eine Umstrukturierung einiger Klassen. Ebenso führt dieser Fakt dazu, dass RDDs nur vom Master-Prozess erzeugt werden können, da dies die Aufgabe der Spark-Kontext verwaltenden Einheit ist. Da RDDs nicht erweiterbar sind, bedeutet dies auch, dass große Datenmenge potentiell erst auf dem Master gesammelt werden müssen, bevor sie dann verteilt abgearbeitet werden können.

Eine Änderung im Vergleich zum Entwurf ist auch im Controller bzw. konkret im CreateStream-Job nötig gewesen. Der Grund hierfür war der ScheduledExecutorService. Dieser ermöglich zwar des Scheduling von periodischen Aufgaben, allerdings ist es hierbei wichtig, dass die Aufgaben innerhalb des gesetzten Zeitraums beendet werden, da es sonst entweder zu Verzögerungen oder zur doppelten Ausführung einer Aufgabe kommt. Ein Problem bereitete hier der endlos-laufende Stream der Spark-Streaming API. Da dieser nicht terminiert, bis er explizit

drauf hingewiesen wird, musste ein Wrapper-Job erstellt werden, der den Streaming Job startet und nach einiger festgelegten Zeit terminiert, bevor der Neustart des Streams ausgeführt werden kann. Ebenfalls wurde die Analyse der gesammelten Tweets in einen periodisch ausgeführten Job ausgelagert. Der Grund dafür ist, dass sonst die Ausführung entweder nur einmal pro Streamstart geschieht, oder permanent auf die Anzahl der im Akkumulator vorhandenen Tweets geprüft werden muss. Eine Auslagerung auf einen Thread hat außerdem den Vorteil, dass es keine sonstigen Verzögerungen im Programmablauf kommt.

5.2 Tests

Nach, während und vorbereitend zur Realisierung wurde das Programm fortlaufend getestet. Das Vorgehen wird im Folgenden beschrieben.

5.2.1 Testziel

Ziel der Tests ist es, Fehler zu finden, die die Ergebnisse verfälschen. Richtige Ergebnisse sind in diesem Fall pro Komponente zu unterscheiden.

DataAcquisition Die akquirierten Daten sind gleichförmig, enthalten immer alle relevanten und derzeit verfügbaren Teildaten und sind vollständig. Dies bedeutet, dass beispielsweise kein Film ausgelassen wird.

DataAccess Daten werden wie von der jeweiligen Methode vorgesehen zurückgegeben bzw. abgespeichert. Es gehen keine Daten verloren. Daten in nicht vorgesehenen Formaten müssen konvertiert werden.

DataAnalysis Die Daten-Analyse muss zu allen korrekt eingegebenen Daten ein Ergebnis liefern. Die Korrektheit dieses Ergebnisses ist dabei kein Teil des Funktionstests. Die Korrektheit der Ergebnisse kann erst nach Inbetriebnahme des Programms und mindestens einem Monat Laufzeit überprüft werden. Die Korrektheit der Sentiment-Analyse wurde bereits in der dazugehörigen Technologieauswahl untersucht. Inkorrekt eingegebene Daten sollen verworfen werden, bzw. sollten nicht vorkommen können.

Controller Die Komponenten liefern ihre Ergebnisse auch im Scheduling korrekt und das Scheduling wird regelmäßig ausgeführt.

5.2.2 Testumgebung

Eine dedizierte Testumgebung wurde nicht geschaffen. Getestet wird direkt auf dem für die Endnutzung vorgesehenen System. Dies hat den Vorteil, dass beim Deployment auftretende Fehler durch Versionierung oder Konfigurationsunterschiede der Umgebungen vermieden werden. Nachteilig ist der erhöhte Aufwand durch jeweils einen Upload auf den Server und die dortige Ausführung des Tests. Dies erfordert erhöhte Sorgfalt bei der Auswahl und Erstellung der Tests. Nicht von Spark abhängige Teil-Programme können direkt auf dem Programmierungsrechner getestet werden.

5.2.3 Testvorgehen

Im Laufe der Arbeit wurden immer wieder Tests durchgeführt, um die richtige Funktionsweise der Komponenten und Frameworks zu garantieren und Fehler frühzeitig zu finden.

Pre-Entwurfsphase

Bevor mit der Entwicklung des Entwurfs begonnen wurde, wurde das Spark Framework getestet und inspiziert. Es wurden offizielle Tutorials befolgt und so die Funktionsweise der Komponenten des Frameworks einzeln und im Zusammenspiel getestet. Es war nötig direkt zu Beginn des Projekts einen Prototyp zu erstellen, der Tweets zu festgelegten Suchworten sammeln und abspeichern kann. Dadurch war es möglich auf einen kleinen Datenbestand zurückgreifen zu können, der später zum Trainieren der ML-Modelle benötigt wurde. Mit diesem Prototyp konnten außerdem Anforderungen an die DataAccess und DataAcquisitions Komponente festgestellt werden. Zusätzlich hat die Entwicklung des Prototyps gezeigt, auf welche Fehler geachtet werden muss und wie man diese verhindern kann. So ist beispielsweise die Untersuchung des Tweet Textes von besonderer Wichtigkeit, da Sonderzeichen bei String Manipulationen zu Problemen führen können.

Entwurfsphase

Während der Entwurfsphase wurden die einzelnen Komponenten unabhängig voneinander getestet, indem Teilanwendungen gebaut wurden. J-Unit Testing ist mit Spark zwar durchführbar, allerdings hat es in diesem speziellen Fall keinen Mehrwert. Da die Spark-Umgebung auf einem separaten Server eingerichtet wurde und für Tests jeweils eine kompilierte Version auf diesen geladen werden musste, war es ebenso sinnvoll eine Testanwendung zu entwickeln, die alle Funktionen einmal durchläuft und ein Ergebnis liefert und ausgibt. Diese Testanwendung wurde mehrfach ausgeführt, um potentielle Fehler in den Live-Daten des Streams oder den

verschiedengearteten Seiten der IMDb zu finden. Durch das Fehlen eines GUI sind Nutzereingaben keine Fehlerquelle. Durch den sehr durchstrukturierten Ablauf des Systems ist, abgesehen von der Verfügbarkeit von Twitter und IMDb vorhersehbar, was passieren kann. Lediglich Eingabeunstimmigkeiten der IMDb Community können zu unerwarteten Fehlern führen. Dies wurde in dieser Phase jedoch bemerkt und Vorkehrungen wurden dahingehend getroffen.

Pre-Deploymentphase

Nachdem alle Komponenten einzeln erfolgreich getestet und Fehler behoben wurden, wurde eine Version, die alle Komponenten miteinander verbindet auf dem Server ausgeführt. Diese Version hatte alle Funktionalitäten der finalen Version, führt die Jobs aber viel häufiger aus, so dass potentiell entstehende Probleme schnell gefunden werden. Mit Hilfe von Debugging-Ausgaben wurden letzte Fehler und Unstimmigkeiten gefunden, die durch das Scheduling aufgetreten sind.

5.2.4 Testdaten

Testdaten für ein Live-System zu beschaffen ist problematisch. Die Komponenten DataAcquisitions und zum Teil DataAccess können ohne Echtzeit-Daten getestet werden. Die DataAcquisitions-Komponente wird getestet, indem für den IMDB-Scraper Filme ausgewählt wurden, die zum einen alle notwendigen Daten enthalten und zum anderen verschieden unvollständig waren. Außerdem wurden die Daten aus dem „Coming-Soon“ Bereich der Website beschafft. So kann wahrscheinlicher getestet werden, ob alle möglichen Konstellationen an Datenvollständigkeit verarbeitet werden können. Diese so beschafften Filmdaten wurden zum Testen der Erstellung des Twitter-Stream verwendet und ebenso zum Testen der DataAccess-Komponente.

Um den Sentiment-Analyseteil der DataAccess Komponente zu testen wurden 30 Tweets von Hand ausgewählt und auf das Ergebnis überprüft. Diese Tweets enthalten Hashtags, Usernames, Sonderzeichen und sind zu allen möglichen Sentiments zuzuordnen. Im späteren Projektverlauf wurde die Sentiment-Analyse mit Live-Daten aus dem Stream getestet.

Testdaten für die Modellerstellung des maschinellen Lernens zu generieren ist zeitaufwendig. Mit Hilfe eines sehr frühen Prototyps wurden Tweets gesammelt, die auf Filme zuordenbar sind. Diese Tweets wurden später auf ihr Sentiment analysiert und gezählt, sodass für eine Anzahl von 25 Filmen Daten vorhanden sind. Diese Daten wurden verwendet, um ein Modell zu erstellen, das für das maschinelle Lernen verwendet wird. Im Laufe der Zeit wird die Anzahl der vollständigen Daten anwachsen, sodass das Modell genauer wird. Der erste vollständige Test des ML-Teils kann erst frühestens einen Monat nach Start des Programms durchgeführt

werden. Erst zu diesem Zeitpunkt ist überprüfbar, ob die Analyse der Daten ein der Realität entsprechendes Ergebnis liefert.

5.3 Erkenntnisse

Im Umgang mit der Programmierung von Spark ist mir aufgefallen, dass Spark eine Menge an Fehlern versteckt und eigenhändig behandelt. Von Spark selbst werden innerhalb des Programmablaufs Warnungen und Fehler angezeigt, die allerdings innerhalb des Codes nicht behandelbar sind. Die meisten dieser Fehler führen allerdings nicht zum Programmstopp und so ist es undurchsichtig zu erkennen, was welcher Fehler bewirkt, wie gefährlich er für das Ergebnis ist und wo der Ursprung dieses Fehlers liegt. Sollte der eigene Code der Fehlerherd sein, so bricht das Programm ab. Hierbei ist zu beachten, dass daher Fehler nicht nach oben weitergereicht werden sollten, sondern direkt behandelt werden sollten.

Positiv hervorzuheben ist die MLlib, die es sehr einfach macht maschinelles Lernen zu implementieren. Ein solcher Mechanismus ist in 5 Zeilen Code implementiert. SparkStreaming ist dafür umso uneleganter. Streams können einfach gestartet werden und auch mehrere Streams innerhalb eines Kontexts sind möglich. Allerdings ist es nicht möglich diese Stream unabhängig voneinander zu stoppen. Will man einen Stream beenden, so muss man den gesamten Streaming Kontext beenden. Will man einen Stream neu starten, muss man alle Streams neu starten. Da Streams im Normalfall sehr lange laufen sollen, ist dies nicht unbedingt ein Argument gegen die Nutzung von SparkStreaming. Läuft ein Stream, so wird dieser permanent und zuverlässig abgearbeitet. Für dieses Projekt entstand dadurch allerdings ein Problem, da das beenden und neustarten eines Streaming Kontextes ein paar Sekunden in Anspruch nimmt und sich nicht gut „anfühlt“.

Die Eigenschaft von Spark Fehler eigenständig zu behandeln führt dazu, dass viele Fehler oder Probleme erst beim x-ten Testlauf auftreten. Zu Fehlern die in dieser Art auftreten gehören vor allem Java-Heapspace und Speicherprobleme. Es kann passieren, dass der Garbage Collector nicht mit der Erzeugung der Objekte seitens des Streams mithalten kann. Da dies aber nur sehr unregelmäßig auftritt, war es schwer dieses Problem zu identifizieren.

Ebenso abraten muss man von der Nutzung äußerer APIs, vor allem im Streaming Kontext. Aufgrund der Sentiment140 API wurde der Spark Master gezwungenermaßen zum Flaschenhals, da alle erhaltenen Tweets auf diesem zwischengespeichert werden mussten, um die Tweets an den Service zu senden. Der Spark Master muss daher und für solche und ähnliche Fälle besonders viel Speicherkapazität mitbringen. Eine eigene Implementierung einer Sentiment Analyse wäre für den Ablauf des Streams natürlicher und effizienter, da die einzelnen

Tweets direkt analysiert werden können, sobald sie eintreffen. Dadurch wäre keine weitere Zwischenspeicherung der Tweets notwendig gewesen und keine Verzögerung im Ablauf würde auftreten.

Allgemein ist aufgefallen, dass die Programmierung ohne anständige Testumgebung langwierig ist und vermieden werden sollte. Da eine Spark Installation auf Windows etwas umständlicher ist, empfand ich es nicht für nötig dies auf dem zur Programmierung genutzten Rechner zu tun. Im Nachhinein muss festgestellt werden, dass auch aufgrund der etwas versteckteren Fehlerquellen eine ordentliche Testumgebung für Spark ein Muss ist und sehr viel Zeit spart.

Die Nutzung des java-internen Schedulers „ScheduledExecutorService“ stellte sich als problematisch heraus. Gerade im Zusammenspiel mit dem dauerhaft laufenden Stream wurde das Scheduling sehr unpräzise und verzögerte sich um viele Stunden. Genaue Gründe dafür wurden bisher nicht gefunden. Allerdings scheint dieses Problem bekannt zu sein, da in der Dokumentation angegeben ist, dass dies vorkommen kann.

Abseits der Spezifika des Spark Frameworks ist aufgefallen, dass die Erstellung und Nutzung des Entwurfs gut funktioniert hat. Durch das Erlernen von Feinheiten und Fehlerquellen des Frameworks wurde der Entwurf iterativ erweitert und verbessert und bietet stets einen guten Überblick über das System. Besondere Schwierigkeiten bereitet das Finden von klar definierten nicht-funktionalen Anforderungen.

Die Arbeit mit Maven ist äußerst komfortabel, kann aber auch schnell zu Dependency Problemen führen.

6 Experiment und Evaluation

In diesem Kapitel werden das Experiment und dessen Ergebnisse erläutert und betrachtet. Auf der Grundlage des Experiments wird dann eine Evaluation des Apache Spark Frameworks vorgenommen.

6.1 Experiment

Das Experiment nutzt das erstellte Programm, um den Erfolg der ab dem 1. August 2016 erscheinenden Filme vorherzusagen. Zunächst werden dazu für die Prediction-Modelle die bis zu diesem Zeitpunkt gesammelten und vollständigen Daten verwendet. Die Anzahl der vorhandenen Datensätze, liegt dabei bei 35. Aufgrund der sehr kleinen Datenmenge ist die Vermutung, dass die Vorhersage eher schlecht funktionieren wird. Die dargestellten Ergebnisse sind die zuletzt getroffenen Vorhersagen. Die Vorhersagen können sich im Laufe der Zeit verändert haben.

6.1.1 Erste Phase

Die erste Überprüfung der Ergebnisse wurde am 18. August 2016 durchgeführt und umfasst ein Ergebnis für 11 Filme.

Name	Suicide Squad	Nine Lives	Little Men	Five Nights i	Front Cover	Elliot, der Drac
Tweets	3.800.520	50.394	14.339	806	22.547	58.774
Positiv	448.349	10.203	2.154	90	3.898	18.335
Negativ	187.571	1.342	450	4	1.095	1.002
Vorhersage(Kl.)	>$50M	>30$M	>$100.000	<$100.000	>$50M	>$30M
Vorhersage(Reg)	179.139.142	24.047.441	5.784.558	231.868	24.047.441	103.261.464
Tatsächlich	133.682.248	6.249.915	30.368	9.059	6.907	21.514.095

Florence Foster J	Sausage Party	Hell or High \	Anthropoid	Ghost Team
37.924	124.078	17.386	26.593	1.151
5.129	28.836	1.771	7.776	389
316	3.195	156	81	5
>$15M	>$30M	>$100.000	>$15M	>$15M
36.835.125	24.047.441	5.784.558	5.784.558	231.868
6.601.313	34.263.534	621.329	1.233.519	8.069

Abbildung 6.1: Ergebnisse der zweiten Phase des Experiments

Zunächst fällt bei der Betrachtung der Ergebnisse auf, dass diese meist nicht treffend sind. Ebenfalls fällt auf, dass die Vorhersagen zwischen Klassifikation und Regression häufig stark unterschiedlich sind. Die Klassifikation kommt lediglich in 4 Fällen zu einem ansprechenden Ergebnis. Dabei ist zu bemerken, dass „>$50M“ und „<$100.000“ die jeweils größte bzw. kleinste Klasse darstellt und Ergebnisse darüber und darunter demnach als Treffer zu werten sind. Die Trefferquote liegt damit bei ca. 36%.

Die Regression kommt in 5 Fällen zu passablen Ergebnissen, überschätzt allerdings in den meisten Fällen. Hier ist zu beachten, dass „231.868“ die kleinste Klasse darstellt, die die Regression gefunden hat.

Vollkommen aus dem Rahmen fällt der Film „Front Cover“, der sowohl von der Regression, als auch von der Klassifikation deutlich über seinem eigentlichen Einspielergebnis eingeordnet wurde. Eine plausible Erklärung lässt sich dafür nicht finden, da Filme mit ähnlichen Tweet-Anzahlen geringer eingeschätzt wurden. Es fällt zu diesem Zeitpunkt auf, dass die kleine Datenmenge dazu führt, dass Ergebnisse in den „mittleren Bereichen“ nicht korrekt eingeordnet werden können. Dies hat zum Grund, dass diese Bereiche im Verhältnis deutlich kleiner sind, als die Bereiche der Blockbuster (>$50M - Open End) und der unbekannten Indie-Filme (<$100.000 - $0). Um eine genauere Vorhersage in diesem Bereich treffen zu können, muss eine größere Datenmenge angesammelt werden.

Nach der Aufzeichnung dieser Ergebnisse wurde das Modell um die nun hinzugekommenen Filme ergänzt. Außerdem sind in dieser Phase einige Fehler im Programm aufgefallen, wie die fehlende Vervollständigung von anfangs nicht vorhandenen, aber benötigten Daten. Dies hat das Ergebnis ebenfalls verfälscht. Diese Fehler wurden behoben.

6.1.2 Zweite Phase

Mit den nun hinzugekommenen Werten umfasst die Datenmenge 46 Datensätze und ist somit ca. 31% größer, als die zunächst verwendete Datenmenge. Das Ergebnis sollte daher geringfügig besser sein, als in der ersten Phase.

Auch anhand dieser Ergebnisse ist zu sehen, dass die Vorhersagen der Klassifikation und Regression recht weit auseinander liegen können. Die Klassifikation liefert in 5 Fällen gute Ergebnisse und überschätzt dabei nicht, wenn man von der Schätzung „<$100.000“ absieht, da dies die kleinstmögliche Schätzung ist. Die Trefferquote liegt dabei allerdings immer noch nur bei 33%, was deutlich zu gering ist, um einen Mehrwert zu erhalten. Die Regression versagt in dieser Phase vollkommen und liefert in nur zwei Fällen passable Ergebnisse. Die Filme „Morris...“ und „Der Staat gegen...“ sagen jeweils die zweitkleinste ermittelte Klasse vorher und können somit eventuell auch als „passabel“ eingestuft werden. Zum Film „Kingsglaive“, der von

Name	Kubo: Der Tap	Ben-Hur	War Dogs	Morris aus Ar	Kingsglaive	Der Staat ge
Tweets	22.892	9.999	42.810	3.940	38.688	34
Positiv	2.820	4.071	9.993	389	4.037	6
Negativ	121	146	651	3	513	0
Vorhersage(Kl.)	>$30M	>$30M	>$30M	<$100.000	>$30M	<$100.000
Vorhersage(Reg)	5.784.558,25	5.784.558,25	36.835.125	231.868,38	36.835.125	231.868
Tatsächlich	12.608.372	11.203.815	14.685.305	15.673	119.801	33.781

The Mechanic	Don't Breathe	Complete Unl	My first Lady	Hands of Sto	Mia madre
48.156	94.319	3.509	24.782	88.633	43.442
5.634	8.752	201	10.265	42.893	884
1.480	5.829	21	154	1023	244
>$30M	>$15M	>$250.000	>$1M	>$1M	>$100.000
22.383.146	6.022	249.504	103.261.464	27.836.935	36.835.125
7.456.525	26.411.706	14.149	2.868.956	1.751.388	34.098

The Hollars	Einer nach de	The Light bet
7.354	156	17.840
1.551	2	1.405
64	1	366
>$100.000	>$250.000	>$100.000
5.784.558	249.504	249.504
38.210	5.208	4.984.000

Abbildung 6.2: Ergebnisse der ersten Phase des Experiments

beiden Modellen stark überschätzt wurde, kann gesagt werden, dass es sich dabei, um einen Film handelt, der zur Marketing Kampagne des Videospiels „Final Fantasy XV“ gehört. Dies erklärt die hohe Anzahl an Tweets. Dieser Film wurde jedoch nur in stark vereinzelten Kinos ausgestrahlt, was das geringe Einspielergebnis erklärt. Zu diesem Zeitpunkt des Experiments liefern beide Modelle leider nicht viel mehr, als zufällige Treffer. Lediglich bei sehr hohen und sehr geringen Einspielergebnissen ist die Tendenz generell richtig. Die hier dargestellten Ergebnisse werden ebenfalls in das Modell der nächsten Phase übernommen.

6.1.3 Dritte Phase

Die Datenmenge umfasst nun 61 Datensätze. Dies ist eine Vergrößerung um ca. 33In dieser Phase, die die Filmreleases vom 01.09.2016 bis zum 19.09.2016 abdeckt, erschienen hauptsächlich kleine Filme, die nur in ausgewählten Kinos erschienen. Daraus folgte, dass ein Teil der Ergebnisse nicht korrekt vorhergesagt werden konnte. Von 12 Filmen waren 4 sogenannte „Limited Releases“, wurden also nur in wenigen (4-20) Kinos gezeigt. Die Vorhersagen dieser Filme lagen sehr stark daneben. Von den restlichen 8 Filmen wurden lediglich 3 zufriedenstellend vorhergesagt. Die Klassifikation erweist sich mittlerweile als sehr unzuverlässig.

6.1.4 Vierte Phase

Um die 12 neuen Einträge ergänzt, ist die Datenmenge nun auf 73 Einträge erweitert. Diese Phase deckt die Filmreleases vom 19.09.2016 bis zum 03.10.2016 ab.

Name	Die Gloreichen	Störche	Queen of Kat	Goat	The Lovers	Die Insel der
Tweets	60.520	40.327	70.491	75.558	992	41.077
Positiv	7.085	12.531	25.152	18.902	56	2.940
Negativ	619	84	284	3311	3	234
Vorhersage(K	>$5M	<$100.000	>$1M	>$15M	>$100.000	>$5M
Vorhersage(F	29.175.512	29.175.512	1.697.336	450.126	131.234	29.175.512
Tatsächlich	34.703.397	21.311.407	300.000	23.000	14.000	28.500.000

Deepwater Ho	Masterminds	American Honey
122.951	19.783	10.078
14.410	5.663	2.145
1379	389	138
>$15M	>$30M	>$250.000
16.973.363	4.560.798	4.560.798
20.600.000	6.600.000	71.203

Abbildung 6.3: Ergebnisse der vierten Phase des Experiments

Von den hier untersuchten Filmen sind „American Honey“, „Goat“ und „The Lovers and The Despot“ jeweils limited Releases. Wenn man diese Filme ausklammert, erreicht die Regression in diesem Fall bei 5 von 6 Filmen ein gutes bis sehr gutes Ergebnis. Die Klassifikation hingegen ist sehr schwach und erreicht lediglich ein gutes Ergebnis. Es zeichnet sich also bei dieser Datenmenge bereits ab, dass der Naive Bayes für Probleme dieser Art nicht zuverlässig funktioniert. Die Regression hingegen erzielt bei „fairen“ Umständen gute Ergebnisse.

6.1.5 Fazit

Nach einem zweimonatigen Test des Programms ist vorrangig auffällig, dass die Klassifikation mithilfe des Naive Bayes auch mit steigendem Datenbestand nicht an Präzision gewinnt. Die Regression durch das Decision Tree Model hingegen zeigt eine positive Tendenz. Besonders in diesem Fall ist zu erwarten, dass mit erhöhtem Datenbestand genauere Ergebnisse erzielt werden. Jedoch ist anzumerken, dass beide Verfahren mit der in diesem Experiment gesammelten Datenmenge noch sehr ungenau sind. Für eine zuverlässige Vorhersage ist das System daher nicht bereit. Zur weiteren Erhöhung der Präzision wäre es von Vorteil, nur Filme zu untersuchen, die einen großen Release erhalten. Die Limited Releases werden zwar teilweise auf Twitter gut besprochen, können jedoch nur von wenigen Menschen im Kino gesehen werden. Ebenso ist das Problem der Zuteilung von Tweet zu Film noch nicht optimal gelöst. Filme wie „Mother’s Day“ oder „Goat“ haben sehr generische Titel. Tweets, die diese Worte enthalten

werden zum Großteil nicht zum Film zugehörig sein, werden jedoch als Solches gewertet. Dies führt unweigerlich zu einer ungenauen Vorhersage für Filme dieser Art. Außerdem beeinflussen diese Filme ebenso die zukünftigen Vorhersagen wahrscheinlich negativ. Abgesehen von diesen Problemen läuft das Programm zuverlässig und - nach ein paar Fehlerbehebungen nach der ersten Phase - fehlerfrei. Systemabstürze kamen über den gesamten Zeitraum des Tests nicht vor.

6.2 Evaluation

Im Folgenden wird das Apache Spark Framework für den Anwendungsfall Social Media Monitoring und im speziellen für das hier aufgezeigte Fallbeispiel evaluiert. Es wird pro Kategorie eine Bewertung von schlecht, mittel und gut geben.

Bei der Evaluation eines Frameworks ist es wichtig zu erarbeiten, wie gut dieses anwendbar ist. Um dies zu tun, wird das Framework auf die Kategorien Lernkurve, Entwicklungsperformance und Dependency Management untersucht. Diese Kategorien beschäftigen sich weniger mit der Leistung der Frameworks und mehr mit der Nutzung. Die Leistung wird in den Kategorien Funktionalität, Skalierbarkeit, Robustheit, Wartbarkeit und Testbarkeit untersucht. Das Framework wird auf Skalierbarkeit untersucht, da diese von großer Wichtigkeit ist, wenn mit großen Datenmenge gearbeitet wird, wie es beim Social Media Monitoring der Fall ist. Robustheit ist wichtig, da beim Social Media Monitoring möglichst keine Daten verloren gehen sollen und das Programm daher stabil laufen muss. Wartbarkeit ist eine Qualität, die in großen Projekten gegeben sein muss. Es müssen neue Programmteile einfach hinzuzufügen sein, vor allem wenn neue Datenquellen zum Monitoring hinzugefügt werden sollen. Testbarkeit als Kategorie ist eine Kombination auf subjektiver Nutzbarkeit und objektiver Qualität. Testbarkeit ist für jedes Programm von Wichtigkeit, vor allem wenn es eine lange Zeit laufen soll.

6.2.1 Lernkurve

In dieser Kategorie wird untersucht den Schwierigkeitsgrad, mit dem das Framework zu erlernen ist.

Apache Spark bietet auf der Website einen so genannten „Programming Guide“ an. Zum einen gibt es dort allgemeine Informationen zum Setup des Frameworks, zum anderen gibt es erste „Hello World“-Tutorials und tiefergehende Informationen zu jedem Bestandteil des Frameworks. Des Weiteren werden Codebeispiele für verschiedene Funktionalitäten in zumeist allen drei unterstützten Programmiersprachen gegeben. Der Einstieg in das Framework fällt daher nicht schwer. Es ist dabei anzuraten, die Programming Guides sorgfältig zu lesen, da dort

auf Stolpersteine hingewiesen wird und auch Negativbeispiele gezeigt werden mit Begründung, warum eine Programmierung in der Art Fehler und Probleme verursachen kann. Es fällt daher nicht schwer zu behaupten, dass Spark einen der besten Einsteiger Guides bietet, den man sich wünschen kann.

Dennoch werden einige Kleinigkeiten nicht explizit genannt und so kann es im Laufe des Entwurfs dazu kommen, dass überlegte Lösungen in eine falsche Richtung geführt werden. Als Beispiel ist hier die Notwendigkeit der möglichen Serialisierung zu nennen, die für alle Objekte notwendig ist, die innerhalb einer Spark-Funktion, wie map oder reduce, verwendet werden sollen. Zwar wird genannt, dass Spark RDDs als serialisierte Objekte verwaltet, jedoch wird nicht genannt, dass der Spark Kontext selbst, nicht serialisierbar ist oder was in welchem Zusammenhang hier serialisiert wird.

Schön ist wiederum, dass Dinge wie Performanceoptimierung und diverse Setup Möglichkeiten in der Dokumentation besprochen und erklärt werden. Die Lernkurve ist damit anfangs sehr steil, wird aber zunehmend flacher wenn es in die Tiefe des Frameworks geht. Die einzelnen inneren Bestandteile werden nicht weiter erklärt und so ist es schwierig das Framework durch eigenen Code sinnvoll zu erweitern. Ein Beispiel dafür ist das Scheduling. Intern nutzt Spark einen Scheduling Mechanismus, der es Spark ermöglicht die aufgetragenen Aufgaben zu verteilen und die Bearbeitung zu regeln. Es würde daher Nahe liegen, dieses Scheduling System für weitere selbsterstelle Aufgaben zu nutzen. Dies wäre allein wegen des Dependency Managements sinnvoll, da man keine unterschiedlichen Scheduling Frameworks brauchen sollte für ein System. Es ist allerdings relativ schwierig herauszufinden, wie genau Spark dies regelt und welches Framework genutzt wird. Da dies nirgends erwähnt wird, wurde in dem in dieser Arbeit erstellten Programm ein eigenes Scheduling System verwendet, was eventuell nicht notwendig wäre.

Ebenso bietet Spark sehr viele kleine Stellschrauben, an denen die Performance optimiert werden oder im schlechten Fall verschlechtert werden kann. Es handelt sich bei diesem Framework um einen klassischen Fall von „einfach zu erlernen, aber schwierig zu meistern“. Aufgrund der sehr umfangreichen Dokumentation ist es trotz des großen Funktionsumfangs möglich, schnell einen Überblick zu erhalten und effizienten Code zu erstellen. Innerhalb eines Arbeitstages sollte es auch möglich sein, die Funktionsweise der einzelnen Bestandteile detaillierter zu verstehen. Aufgrund dieses Umstandes erhält das Spark Framework in dieser Kategorie die Bewertung **gut**.

6.2.2 Entwicklungs-Performance

Es wird nun untersucht, in welcher Geschwindigkeit einsatzfähiger Code produziert werden kann. Um ein sehr einfaches Beispiel oder sehr einfache Aufgaben, wie simple Map-Reduce Aufgaben zu erstellen ist von der Entwicklungsseite aus nicht viel Zeit und Aufwand notwendig. Wenn mit Java gearbeitet wird, sollte die seit Java 8 bestehende Lambda-Notation verwendet werden. Diese ermöglicht ein sehr sauberes und effizientes Schreiben von Jobs für Spark. Innerhalb weniger Zeilen und daher wenigen Minuten kann so bereits eine starke Verarbeitung von Daten stattfinden. Ein größeres Problem bereitet das Aufsetzen der Spark Umgebung. Es wird von Seiten des Entwicklers empfohlen das Framework eigens für eine gewählte Hadoop Version, die ebenfalls auf dem Rechner installiert sein sollte, zu bauen. Allerdings gibt es auch Pre-Build Versionen für verschiedenste Hadoop Versionen, die verwendet werden können. Es gibt auf der Spark Homepage Anleitungen, die beim Bau und aufsetzen der Spark-Umgebung helfen. Wer keine Erfahrungen mit Maven oder ähnlichem hat, kann dort dennoch kaum Hilfe erhalten.

Nach dem Aufsetzen der Umgebung ist ein Starten der Applikationen sehr einfach möglich. Spark regelt selbstständig die Verteilung von Ressourcen. Somit kann sich in der Erzeugung des Codes komplett auf die Aufgabe konzentriert werden und schon in wenigen Minuten ein zufriedenstellendes Ergebnis erzielt werden. Die Entwicklungsperformance ist damit ebenfalls mit **gut** zu bewerten.

6.2.3 Dependency Management

Die von Spark geforderten Dependencies können über Maven gemanagt werden. Dabei muss jeder Bestandteil des Frameworks (Also Core, MLlib, Streaming, usw.) einzeln eingebunden werden. Dies hat den Vorteil, dass keine unnötigen Dependencies geladen werden müssen. Es können einzeln die Bestandteile geladen werden, die für das Projekt benötigt werden.

Aufgefallen ist bei der Nutzung von Maven, dass stark auf die Versionierung der Bestandteile geachtet werden muss. Es sollte jeweils für jeden Bestandteil die gleiche Versionsnummer gewählt werden, da diese mit dem jeweiligen Spark Core zusammenhängt. Ebenfalls aufgefallen ist, dass es durch die Einbindung der Bestandteile über Maven zu Dependency-Konflikten kommen kann. In diesem Projekt gab es Probleme zwischen MLlib und dem Rest des verwendeten Frameworks. Dieses Problem wurde durch die eigenhändige Einbindung des MLlib JARs umgangen.

Allerdings müssen die Spark-Dependencies, sowie die Spark Bestandteile selbst nicht im kompilierten Programm vorhanden sein, da diese bereits in der Spark Installation auf dem

ausführenden Rechner vorhanden sind. Es müssen jegliche spark-fremde Dependencies vorhanden sein. Dazu gehört auch „TwitterStream“, welches von Spark angeboten wird, um Spark Streaming im Verbund mit Twitter nutzen zu können. Dies ist nicht intuitiv.

Während der Programmierung wird bzw. kann Maven genutzt werden. Auf dem ausführenden Rechner wird eine Installation von Spark vorgenommen. Es muss stark darauf geachtet werden, dass identische Versionen verwendet werden, da es sonst schnell zu Konflikten kommen kann.

Aufgrund all dieser genannten Punkte ist ersichtlich, dass die Einbindung über Maven generell komfortabel ist, jedoch auch einige Probleme entstehen können. Daher wird das Framework in diesem Fall als **mittelmäßig** mit Tendenz zu gut bewertet.

6.2.4 Möglichkeit zum Performance- und Sicherheitstuning

Spark bietet diverse Möglichkeiten zum Performancetuning. Es kann eingestellt werden, wie Ressourcen verteilt werden, die viele Kerne genutzt werden sollen, wie geschedult wird, wie viel Speicher verwendet werden darf, etc. Es gibt eine Vielzahl von Tuning-Stellschrauben. Eine Option, die eventuell fehlt ist die Möglichkeit bestimmte Worker zu präferieren, da die Zuteilung komplett automatisch erfolgt.

Die Möglichkeiten zum Sicherheitstuning sind ebenso zahlreich. Es gibt Optionen, um nur bestimmte Klassennamen zuzulassen, die bearbeitet werden. Man kann eine Liste von Administratoren, zugelassenen Workern und Passwörtern festlegen. Ebenso ist die verschlüsselte Übertragung von Daten zwischen den Workern und dem Driver möglich. [Apache Software Foundation (2016b)]

Das Tuning erfolgt durch Konfigurationsdateien, die von Spark vor jedem Start gelesen werden. Diese Funktionalität wurde allerdings nicht getestet, da in diesem Projekt kein Cluster verwendet wurde und Spark keinen Port nach außen öffnet. Dennoch wird dieser Abschnitt vorsichtig mit **gut** bewertet, da die Möglichkeiten vielzählig sind.

6.2.5 Funktionalität

In diesem Abschnitt wird untersucht, ob Spark die Funktionalität bietet, die zum Social Media Monitoring benötigt werden. Social Media Monitoring findet statt, indem Daten aus sozialen Netzwerken -möglichst in Echtzeit- extrahiert und analysiert werden. Hierfür kann Spark Streaming genutzt werden. Allerdings bietet Spark von Haus aus nur eine Streaming Implementation für das Social Network „Twitter“. Über die Integration von Apache Kafka oder Flume

ist es dennoch möglich eigene Streams zu erstellen. So sollte es möglich sein diverse soziale Netzwerke, sofern sie über APIs verfügen, zu verfolgen.

Die Echtzeitfähigkeit von Spark Streaming ist dabei jedoch ein Kritikpunkt. Spark Streaming tauscht Echtzeit gegen Verarbeitungsgeschwindigkeit aus. Durch den Stream erhaltene Daten werden in Batches gesammelt und erst nach einer definierten Zeit gemeinsam abgearbeitet. Da die Zeit zum Sammeln eines Batchs vom Nutzer definiert werden kann ist dies kein Ausschlusskriterium, sollte allerdings erwähnt werden. Spark Streaming bietet kein Echtzeit-Processing, sondern ein auf Streams angewandtes Batch-Processing. Dies hat zur Folge, dass die Verarbeitung der Daten schnell durchgeführt wird. So muss für jedes Projekt abgewägt werden, ob Echtzeit kritisch ist. In den meisten Fällen des SMM ist die Anwendung nicht exakt zeitkritisch und daher bietet Spark hier dennoch eine gute Lösung. Das SMM kann ebenfalls durch einfache Map-Reduce Aufgaben und daher vom Spark Core durchgeführt werden. In dem hier gezeigten Anwendungsfall wurde die Seite IMDb überwacht und Filme regelmäßig auf neue Daten überprüft. Dies wurde mit Hilfe des Spark Cores parallelisiert und so effizient durchgeführt.

Sollten Beziehungen und tatsächliche Netzwerke interessant sein, kann Sparks GraphX verwendet werden um diese als Graphen darzustellen und zu untersuchen. Dies ist in der Analyse von sozialen Netzwerken sehr hilfreich, da diese stark über Beziehungen zwischen Nutzern laufen und durch die Nutzung von GraphX vor allem im Zusammenspiel mit der MLlib zusammenhänge gefunden werden können.

Durch die MLlib ist es schnell möglich maschinelles Lernen in die Anwendung zu integrieren. Damit können Vorhersagen getroffen und Zusammenhänge gefunden werden. Auch dies ist eine Funktionalität, die im SMM gegeben sein muss.

Eine Funktionalität, die im Zusammenhang mit Social Media Monitoring wünschenswert ist, ist die native Möglichkeit Aufgaben für bestimmte Zeitpunkte und Termine zu schedulen. Da laut Spark nur ein Spark Kontext pro JVM bestehen soll, ist dies eine Funktionalität die praktisch wäre. Läuft eine Spark Aufgabe, die einen dauerhaften Stream untersucht, so läuft diese Aufgabe unendlich lang. Will man eine andere Aufgabe, beispielsweise die regelmäßige Untersuchung einer Website durchführen, so muss dies im gleichen Spark Kontext geschehen. Diese Aufgabe kann demnach nicht von außerhalb per Cron-Job oder ähnlichen Ansätzen gestartet werden, da hier ein neuer Kontext für diese Aufgabe erstellt werden würde. Es muss daher ein Super-Programm geschaffen werden, dass alle potentiellen Aufgaben managt, wie es in dem hier erstellten Programm der Fall ist. Dies ist unvorteilhaft und hebt erneut den Batch-Verabeitungsansatz von Spark heraus. Da dieses Problem lösbar ist und die Funktionalitäten abseits davon sehr vielseitig sind, ist Spark auch in diesem Bereich als **gut** einzuordnen.

6.2.6 Skalierbarkeit

Dieser Abschnitt befasst sich damit, wie gut Spark mit immer wachsenden Datenmengen umgehen kann. Spark ist dafür konzipiert, riesige Datenmengen zu verarbeiten. Spark ermöglicht es, Daten auf unterschiedlichen Rechnern verteilt zu verarbeiten und dennoch auf einem gemeinsamen Datenspeicher zu arbeiten. Es können beliebig viele Worker an einen Driver angebunden werden und dies auch während der Ausführung eines Programms. Damit können immer weiter wachsende Datenmengen verarbeitet werden. Diese Funktionalität ist natürlich nur dann gegeben, wenn Spark in einem Cluster konfiguriert wird. Dies war in diesem Projekt nicht der Fall, weshalb ein Großteil dieser Theorie nicht getestet wurde. Abzusehen ist aber auch in der lokalen Installation, dass der Entwickler nicht wissen muss wie viele Worker zur Verfügung stehen. Spark reguliert und scheduled eigenständig die Nutzung der verfügbaren Worker im Cluster.

Je nach Konzeption einer Aufgabe muss beachtet werden, dass der Driver-Rechner schnell zum Flaschenhals werden kann, da für einige Aufgaben eine Sammlung der Daten beim Driver nötig ist. So zum Beispiel bei der Erstellung eines initialen RDDs. Dahingehend muss verstärkt aufgepasst werden. Dies wurde ebenfalls mithilfe in der lokalen Installation während der Implementierung des hier entworfenen Programms festgestellt. Die Skalierbarkeit ist eine der großen Stärken von Apache Spark und daher mit **gut** zu bewerten.

6.2.7 Robustheit

Fehlertoleranz ist eine Eigenschaft, die Spark besonders auf ihrer Website hervorhebt und wird automatisch, ohne eigenen Code schreiben zu müssen erreicht. Die Daten werden verteilt in RDDs gespeichert und führen einen Log, der angibt, wie sie entstanden sind. So ist es möglich auch bei Ausfall eines Workers verlorene Daten wiederherzustellen. Allerdings ist ein RDD nur dann vollkommen fehlertolerant, wenn die ursprüngliche Datenquelle ebenfalls fehlertolerant ist. Ist die Datenquelle beispielsweise eine repliziert abgespeicherte Datenbank, ist die Wiederherstellung kein Problem und bietet in dem Fall eine exactly-once Verarbeitungsgarantie.

Wenn es um Streaming von Internetdaten geht, ist dies nicht der Fall. Trotzdem garantiert Spark hier eine hohe Fehlertoleranz. Diese hängt davon ab, ob für den Stream ein verlässlicher Empfänger vorhanden ist, dieser also eine Bestätigung an den Stream schickt, nachdem Daten erhalten wurden. Ist dies der Fall, so werden im Falle eines Ausfalls die nicht verarbeiteten Daten erneut versendet. Ist dies nicht der Fall, können Daten verloren gehen. Erhaltene Daten werden von Spark Streaming auf mehrere Worker repliziert, um einen Datenverlust zu vermeiden.

Fällt der Driver selbst aus, so verliert er jegliche Verbindungen zu den Workern und damit sind alle Daten verloren. Dies zeigt erneut, dass der Driver-Rechner der Flaschenhals des Systems ist. Seit Spark 1.2 gibt es die write-ahead Logs, die diesen Datenverlust verhindern. Jegliche erhaltenen Daten werden in dem Fall in einem permanent gespeicherten write-ahead Log geschrieben. Solange der Speicher Fehlertolerant ist, ist es in diesem Fall auch die Verarbeitung von Streamingdaten, wenn verlässliche Empfänger verwendet wurden. Die Verarbeitungsgarantie in dem Fall ist eine at least-once Garantie. Die write-ahead Logs bringen die Kosten mit sich, dass der Datendurchsatz verringert wird, da bei jedem Datenempfang in das Log geschrieben werden muss. Bis auf die write-ahead Logs, die manuell aktiviert werden müssen, geschieht diese Absicherung automatisch im Hintergrund.
Über die gesamte Länge des Experiments hinweg ist außerdem kein Ausfall aufgetreten. Die Robustheit von Spark erhält damit ebenfalls eine **gute** Bewertung.

6.2.8 Wartbarkeit

Spark bietet zur Wartung einen Webservice, der die Überwachung der Prozesse ermöglicht. Dieser Service bietet Informationen über Speichernutzung, Prozessanzahl, Prozessscheduling und ermöglicht die Einsicht von Logs. Dies ermöglicht es, zu jeder Zeit den Status des Programms zu überprüfen. Spark wird fortlaufend weiterentwickelt. Durch die Updatefrequenz ist ein regelmäßiges Upgrade der Software möglich und zu empfehlen. Dies erhöht jedoch den Wartungsaufwand, da bestimmte Codefragmente schnell „depricated“ und daher nicht mehr zu Nutzung empfohlen sind. Zu diesem Zeitpunkt der Entwicklung von Spark ist die Wartbarkeit damit eher **mittelmäßig** zu beurteilen.

6.2.9 Testbarkeit

Die Testbarkeit der mit dem Spark Framework erstellten Software hängt stark von der Entwicklungsumgebung ab. In jedem Fall muss eine Spark Installation auf einem Testrechner vorgenommen und konfiguriert werden. Die reine Funktionalität ist durch Unit-Tests auch mit einem einzigen Rechner leicht zu testen. Die Testbarkeit im Cluster wiederum ist mit erhöhtem Aufwand verbunden, da dieses zunächst eingerichtet und konfiguriert werden muss. Im Clustereinsatz ist es ebenfalls möglich, dass Fehler auftreten, die im Test mit einem einzelnen Rechner nicht aufgetreten sind. Die Testbarkeit ist gegeben, aufgrund des Aufwands allerdings nur als **mittelmäßig** zu bewerten.

7 Fazit und Ausblick

Abschließend werden die Ergebnisse der Arbeit zusammengefasst und ein Ausblick darüber gegeben, wie das hier entworfene System verbessert werden kann.

7.1 Fazit

Das „Apache Spark“-Framework eignet sich durchaus, um ein intelligentes Social Media Monitoring Programm darauf aufbauend zu entwickeln. Durch die RDD Abstraktion der Daten ist es möglich sämtliche Datenquellen einfach für die effiziente, verteilte Verarbeitung durch Spark aufzubereiten. Datenstreams können zwar nicht in Echtzeit, dafür allerdings sehr effizient und sicher, verarbeitet werden. Dies ist für die meisten Anwendungsgebiete ausreichend. Mit der MLlib bietet Spark einen einfachen Einstieg in das maschinelle Lernen. Zur Anwendung der Algorithmen und Modelle ist kein tiefgehendes Wissen notwendig. Dies hat den Vorteil, dass schnell ML-Aufgaben programmiert werden können, die gut funktionieren und verteilt verarbeitet werden können. Die Anpassungsfähigkeit ist daher allerdings nur an vordefinierten Stellschrauben möglich.
Zusammenfassend ist das „Apache Spark Framework“ ein gut funktionierendes, einfach zu erlernendes Framework für die verteilte Verarbeitung von großen Daten und absolut zu empfehlen.

Das in dieser Arbeit entworfene System ist in der Lage Erfolgsvorhersagen anhand von geschriebenen Tweets und gesammelten Informationen über die Filme zu erstellen. Diese Vorhersagen sind aufgrund einer sehr kleinen Datenmenge allerdings noch nicht zuverlässig. Wie erwartet besteht ein Zusammenhang zwischen Tweetanzahl und Erfolg eines Films. Allerdings ist die Abgrenzung, welche Tweets zu einem Film gehören und welche nicht, ein Problem, das noch nicht gelöst wurde.

7.2 Ausblick

Für die Zukunft des Systems ist es von größter Wichtigkeit den Datenbestand zu vergrößern. Nur mit steigender Datenmenge können Vorhersagen zuverlässig abgegeben werden. Ebenso

ist die bessere Filterung der zu untersuchenden Filme notwendig. Filme, die lediglich in einer stark begrenzten Anzahl an Kinos veröffentlicht werden, können zukünftige Vorhersagen verfälschen. Filme dieser Art werden ebenso in den sozialen Medien besprochen, können allerdings nicht von allen Interessenten gesehen werden. Daher sollten zumindest zunächst nur Filme untersucht werden, die National veröffentlicht werden. Dies sollte die Präzision der Vorhersage erhöhen.

Ebenso sollten weitere Daten in Betracht gezogen werden. Weitere Daten, die in dem hier erstellten System nicht beachtet werden, jedoch für den Erfolg einflussreich sind, sollten zur Datenmenge hinzugefügt werden. Dazu gehören die Altersfreigabe, das Genre und die Länge des Films.

Außerdem sollten weitere Datenquellen für Meinungen in Betracht gezogen werden. Facebook, Youtube, Tumblr und Instagram sind dabei beispielsweise Quellen, die wichtig sind, da sie mehr genutzt werden als Twitter, während Twitter immer weniger wächst.[Chaffey (2016)] Allerdings ist der Zugriff auf deren Daten teilweise sehr teuer.

Des Weiteren wäre es wünschenswert eine Sentiment Analyse zu nutzen, die mehr Sprachen abdeckt und auf für das Thema Filme spezialisiert ist. Im Idealfall ist sie auf die jeweiligen genutzten sozialen Medien zugeschnitten. Dies ist möglich, wenn sie eigenständig entwickelt wird.

Literaturverzeichnis

[AlchemyAPI 2016] AlchemyAPI: *AlchemyAPI*. 2016. – URL http://www.alchemyapi.com/products/alchemylanguage

[Apache Software Foundation 2016a] Apache Software Foundation: *Configuration - Spark 2.0.0*. 2016. – URL http://spark.apache.org/docs/latest/configuration.html. – Zugriffsdatum: 15.08.2016

[Apache Software Foundation 2016b] Apache Software Foundation: *Configuration - Spark 2.0.0*, 2016. – URL http://spark.apache.org/docs/latest/configuration.html. – Zugriffsdatum: 15.08.2016

[Apache Software Foundation 2016c] Apache Software Foundation: *Research | Apache Spark*. 2016. – URL http://spark.apache.org/research.html. – Zugriffsdatum: 04.04.2016

[Apache Software Foundation 2016d] Apache Software Foundation: *Spark Programming Guide*, 2016. – URL http://spark.apache.org/docs/latest/programming-guide.html#accumulators. – Zugriffsdatum: 06.06.2016

[Apache Software Foundation 2016e] Apache Software Foundation: *Spark Streaming*. 2016. – URL http://spark.apache.org/streaming/. – Zugriffsdatum: 18.04.2016

[ARD-ZDF 2015] ARD-ZDF: *ARD-ZDF Onlinestudie*. 2015. – URL http://www.ard-zdf-onlinestudie.de/. – Zugriffsdatum: 03.04.2016

[Chaffey 2016] Chaffey, Dave: *Global social media research summary 2016*. 2016. – URL http://www.smartinsights.com/social-media-marketing/social-media-strategy/new-global-social-media-research/. – Zugriffsdatum: 27.10.2016

[Christopher D. Manning 2014] Christopher D. Manning, John Bauer Jenny Finkel Steven J. Bethard David M.: *The Stanford CoreNLP Natural Language Processing Toolkit*. 2014.

– URL http://nlp.stanford.edu/pubs/StanfordCoreNlp2014.pdf. – Zugriffsdatum: 27.10.2016

[Lange 2011] Lange, M.: *Social Media - Social Media Monitoring.* Kap. 9, S. 655–659. In: *Leitfaden Online Marketing Band 2*, marketing Boerse, 2011

[Lexalytics 2016] Lexalytics: *Semantria API.* 2016. – URL https://www.lexalytics.com/semantria. – Zugriffsdatum: 07.06.2016

[Lorenz 2016] Lorenz, S.P.: *Big Data Analytics with Apache Spark - Grundlagen Teil 1.* 2016. – URL http://www.contexagon.com/2015/07/14/big-data-analytics-mit-apache-spark-grundlagen-teil-1/. – Zugriffsdatum: 04.04.2016

[Luger 2002a] Luger, G.F.: *Bayessches Schließen.* S. 371–375. In: *Künstliche Intelligenz*, 2002. – ISBN 3-827-37002-7

[Luger 2002b] Luger, G.F.: *Künstliche Intelligenz.* 4. Auflage. Pearson Studium, 2002

[Matei Zaharia 2010] Matei Zaharia, Michael J. Franklin Scott Shenker Ion S.: *Spark: Cluster Computing with Working Sets.* 2010. – URL https://cs.stanford.edu/~matei/papers/2010/hotcloud_spark.pdf. – Zugriffsdatum: 27.10.2016

[MongoDB, Inc. 2016] MongoDB, Inc.: *Spark Usage.* 2016. – URL https://github.com/mongodb/mongo-hadoop/wiki/Spark-Usage. – Zugriffsdatum: 07.04.2016

[Scott 2015] Scott, J.A.: *Getting Started with Apache Spark - From Inception to Production.* 2015. – URL http://info.mapr.com/rs/mapr/images/Getting_Started_With_Apache_Spark.pdf. – Zugriffsdatum: 05.04.2016

[Sen 2015] Sen, D.W.: *Definition: Social Media Monitoring.* 2015. – URL http://www.social-media-monitoring.org/definition.htm. – Zugriffsdatum: 03.04.2016

[Sentiment140 2016] Sentiment140: *API - Sentiment140.* 2016. – URL http://help.sentiment140.com/api. – Zugriffsdatum: 16.05.2016

[Statista.com 2013] Statista.com: *Twitter: most-used languages.* 2013. – URL http://www.statista.com/statistics/267129/most-used-languages-on-twitter/. – Zugriffsdatum: 16.06.2016

[Statistic Brain Research Institute 2015a] Statistic Brain Research Institute: *Statistic Brain - Facebook Statistics*. 2015. – URL http://www.statisticbrain.com/facebook-statistics/. – Zugriffsdatum: 03.04.2016

[Statistic Brain Research Institute 2015b] Statistic Brain Research Institute: *Statistic Brain - Twitter Statistics*. 2015. – URL http://www.statisticbrain.com/twitter-statistics/

[Sterne 2011] Sterne, J.: *Social Media Monitoring*. HÃ¼thig Jehle Rehm GmbH, 2011

[Twitter, Inc. 2016] Twitter, Inc.: *Streaming API*. 2016. – URL https://dev.twitter.com/streaming/overview/request-parameters. – Zugriffsdatum: 19.04.2016

[We Are Social 2016] We Are Social: *wearesocial.com*. 2016. – URL http://wearesocial.com/special-reports/digital-in-2016. – Zugriffsdatum: 03.04.2016

[Wikipedia 2016a] Wikipedia: *Bayes' Theorem*. 2016. – URL https://en.wikipedia.org/wiki/Bayes%27_theorem. – Zugriffsdatum: 09.08.2016

[Wikipedia 2016b] Wikipedia: *Machine learning*. 2016. – URL https://en.wikipedia.org/wiki/Machine_learning. – Zugriffsdatum: 04.04.2016

[Wikipedia 2016c] Wikipedia: *maschinelles Lernen*. 2016. – URL https://de.wikipedia.org/wiki/Maschinelles_Lernen. – Zugriffsdatum: 04.04.2016

[Wikipedia 2016d] Wikipedia: *Multinomialverteilung*. 2016. – URL https://de.wikipedia.org/wiki/Multinomialverteilung. – Zugriffsdatum: 09.08.2016

[Wikipedia 2016e] Wikipedia: *Naive Bayes classifier*. 2016. – URL https://en.wikipedia.org/wiki/Naive_Bayes_classifier. – Zugriffsdatum: 09.08.2016

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 11. November 2016 Daniel Müller