Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Implementation eines Website-Klassifikators
für die IPIB des Fraunhofer MOEZ

Bachelorarbeit

Leipzig, Februar 2015

vorgelegt von

Lang, Philipp
Studiengang Informatik

Universitärer Betreuer:

Prof. Dr. Gerhard Heyer
Fakultät für Mathematik u. Informatik
Institut für Informatik
Abteilung Automatische
Sprachverarbeitung

Externer Betreuer:

Prof. Dr. Lutz Maicher
Fraunhofer MOEZ
Gruppe Competitive Intelligence

Inhaltsverzeichnis

1 Einleitung

Diese Bachelorarbeit wurde am Fraunhofer-Zentrum für Mittel- und Osteuropa erstellt. Ihr Ziel war die Automatisierung eines Teilprozesses, der im Zusammenhang mit der dort entwickelten Web-Anwendung Intellectual Property Industry Base[1] steht. Bevor das Ziel im Folgenden genauer beleuchtet wird, erfolgt zunächst eine kurze Beschreibung des Umfeldes.

1.1 Das Fraunhofer MOEZ und die IPIB

Das Fraunhofer-Zentrum für Mittel- und Osteuropa (Fraunhofer MOEZ) ist fokussiert auf Internationalisierungsprozesse innerhalb der zunehmend interdependenten Bereiche Wirtschaft, Wissenschaft, Politik und Gesellschaft. Insbesondere ist es dabei auf die folgenden drei entscheidenden Trends spezialisiert: Die weltweite Entwicklung zur Wissensgesellschaft, die globale Verschiebung der Wertschöpfung und das globale und regionale Vorantreiben der nachhaltigen Entwicklung, deren Notwendigkeit zunehmend eingesehen wird. Zu den Aufgaben des Fraunhofer MOEZ zählt es, seinen Auftraggebern und Kunden aus Wirtschaft, Wissenschaft und Politik bei diesen Herausforderungen der Zukunft zu helfen. [1]

Unter anderem begleitet es in diesem Rahmen den Wissenstransfer von der Wissenschaft in die Wirtschaft, in dem es innovative Transfersysteme entwickelt und analysiert, um den weltweiten Zugang zu Wissen auszubauen. [2] Diesen Teil des Leistungsangebotes übernimmt die Abteilung Innovative Transfersysteme. [3]

Zu ihr gehört die Arbeitsgruppe Competitive Intelligence, welche nach dem gleichnamigen Prozess benannt ist, bei dem es um das Erkennen, Analysieren und Nutzen von Informationen über Wettbewerber und Marktentwicklung geht. [4]

Die Arbeitsgruppe leitet und realisiert, dass seit Januar 2012 fortlaufende Projekt Intellectual Property Industry Base (IP Industrie Base, IPIB). Dabei handelt es sich um „[…] eine analytische [Online-] Datenbank zu den Akteuren der weltweiten IP-Service-Industrie“ [5]. Das 90 Typen umfassende Dienstleistungsspektrum dieser Industrie ist vielen Unternehmen nicht bekannt und wird daher kaum genutzt. Die Entwicklung und aktive Verwertung des geistigen Eigentums stehe jedoch zunehmend im strategischen Fokus von Unternehmen. Das Ziel der IP Industry Base ist es deshalb, dieses Dienstleis-

1 http://ipib.ci.moez.fraunhofer.de

tungsspektrum auf dem globalen Markt transparent zu machen. Die in der IP Industry Base gesammelten Daten können dazu gezielt Unternehmen zur Verfügung gestellt werden, die daran interessiert sind, die Nutzung ihres geistigen Eigentums zu optimieren. Eine weitere Möglichkeit der Nutzung stellt die Erkennung von Trends dar. Dies wird durch das internationale Monitoring der Branche erlaubt. [5]

Die Datenbank der IPIB umfasste im Mai 2014 ca. 1.900 Profile von Firmen, die in der IP-Service-Industrie tätig sind. Diese Profile wurde händisch eingepflegt. Zu den Daten eines Firmenprofils gehört unter Anderem folgende Informationen: Eine Beschreibung der Firma, ihre Kontaktdaten, die URL ihrer offiziellen Website, ihr Twitter-Nutzername, ihre angebotenen IP-Dienstleistungen, ihre Standorte, ihre Besitzer und Besitzerinnen, ihre höheren Angestellten und die Märkte in denen die Firma tätig ist.

1.2 Aktueller Zustand

In fast allen der 1.900 gespeicherten Firmenprofile der IPIB ist die URL der offiziellen Firmen-Website hinterlegt. Für ca. 400 Firmen ist auch der Twitter-Nutzername bekannt. Um neue Firmen für die IP Industry Base ausfindig zu machen, werden die Tweets der ca. 400 bekannten Twitter-Nutzer automatisiert überwacht. Werden andere Twitter-Nutzer in den Tweets häufig referenziert, werden diese neuen Nutzernamen gespeichert. Die Liste der gesammelten, potentiell für die IPIB interessanten Twitter-Nutzernamen wird dann händisch abgearbeitet. Dabei wird in der Regel von jedem Twitter-Nutzer der Liste das Twitter-Profil aufgerufen, um die eventuell hinterlegte URL der Firmen-Website ausfindig zu machen, sofern es sich bei dem Twitter-Nutzer um eine Firma handelt. Anhand des Inhalts der Firmen-Website wird dann entschieden, ob die Firma im IP-Dienstleistungssektor tätig und somit für die IP Industry Base relevant ist oder nicht. Für eine, als relevant eingestufte, Firma wird dann ein Firmenprofil in der IP Industry Base erstellt.

1.3 Langfristiges Ziel

Da in der aktuellen Prozesskette viele manuelle Eingriffe nötig sind, welche viel Zeit kosten und mit steigender Anzahl neuer Daten nicht mehr zu bewältigen sind, soll diese Prozesskette auf lange Sicht vollautomatisiert ablaufen. Für jeden gesammelten Twitter-Nutzernamen der Liste, soll also die im Twitter-Profil hinterlegte URL automatisch extrahiert und die dazugehörige Firmen-Website automatisch klassifiziert werden. Für die

als relevant klassifizierten Firmen werden anschließend automatisch teilweise ausgefüllte Firmen-Profile in der IP Industry Base angelegt.

1.4 Ziel der Bachelor-Arbeit

Die Bachelor-Arbeit befasst sich mit der automatisierten Klassifikation der Firmen-Websites, wie sie in der langfristig angestrebten Prozesskette erwähnt wurde. Zu diesem Zweck soll ein Kommandozeilen-Programm entwickelt werden. Dieses wird im Rahmen der Bachelor-Arbeit als Klassifikationsprogramm bezeichnet.

Da die Klassifikation einer Website anhand ihres Textes geschieht, muss die Website zunächst heruntergeladen werden. Dieser Vorgang ist kein Bestandteil des Programms. Das Programm erwartet als Eingabe bereits die heruntergeladene Website bzw. den Pfad zum Ordner, der die heruntergeladenen Dateien der Website enthält.

Als Ausgabe liefert das Programm das Klassifikationsergebnis bzw. die Antwort auf die Frage, ob es sich bei der eingegebenen Website um eine für die IPIB relevante Firma handelt oder nicht.

In Kapitel 2 werden einige theoretische Grundlagen erklärt, die zur Umsetzung des Klassifikationsprogramms nötig sind. Dabei werden auch einige Entscheidungen bezüglich der Umsetzung getroffen. Eine grobe Beschreibung der einzelnen Schritt bis zum fertigen Klassifikationsprogramm erfolgt daher erst anschließend in Kapitel 2.9.

1.5 Vorgehensweise

Das Vorgehen beim Anfertigen der Bachelorarbeit orientierte sich an der Design-Science-Research-Methodology nach Peffers et al. [6] Sie ist eine Vorlage für das Durchführen wissenschaftlicher Arbeiten in der Informatik. Das Verfahren unterteilt den Arbeitsprozess in sechs Schritte. Ihre Reihenfolge kann je nach Anwendung variieren. In der ursprünglichen Reihenfolge stellt die Kommunikation der Forschungsergebnisse den sechsten und letzten Schritt dar. Er kann, z.B. durch das Schreiben einer wissenschaftlichen Arbeit, realisiert werden. Die ersten fünf Schritte dienen nicht nur der Gliederung des Arbeitsprozesses, sondern können auch als Gliederung des wissenschaftlichen Werkes verwendet werden. Bei den fünf Schritten handelt es sich um:

1. Identifizierung des Problems und Motivation
2. Definieren der Problemlösung
3. Entwerfen und Entwickeln der Problemlösung

4. Demonstration der Problemlösung
5. Evaluierung der Problemlösung

Die Schritte eins und zwei wurden bereits durch die Einleitung der Bachelorarbeit abgedeckt. Der erste Schritt entspricht Kapitel 1.3 und der zweite Schritt entspricht Kapitel 1.4. Im Rahmen der Arbeit wurde darauf verzichtet, die Bezeichnungen der Schritte als Kapitelüberschriften zu verwenden. Dies hat zwei Gründe. Zum Einen haben die Schrittbezeichnungen keinen Bezug zum spezifischen Thema der Arbeit. Zum Anderen flossen die Schritte vier und fünf mehrfach in den Arbeitsschritt 3 ein. Der Arbeitsschritt 3 befasst sich mit dem Entwurf und der Entwicklung des Klassifikationsprogramms. Er erstreckt sich über die Kapitel 3 bis 4.3. In Kapitel 4 werden mehrere Klassifikationsverfahren getestet, um zu entscheiden, welches die beste Problemlösung darstellt. Jeder Test stellt dabei eine Demonstration und eine Evaluierung einer möglichen Problemlösung dar, vollzieht also Schritt 4 und 5. Die anschließende Implementierung des besten Klassifikationsverfahrens ist wieder ein Bestandteil des dritten Schrittes. Auf einen ausführlichen Test bzw. die Demonstration und die Evaluierung des Klassifikationsprogramms wurde verzichtet. Diese Schritte wurden bereits beim Testen der verschiedenen Klassifikationsverfahren vollzogen. Es wurden exakt die Programmierbibliotheken implementiert, die bereits in den Tests zum Einsatz kamen.

2 Theoretische Grundlagen und getroffene Entscheidungen

Bevor näher auf die Realisierung der Problemlösung eingegangen wird, sollen zunächst einige theoretische Grundlagen erklärt und Entscheidung für die Realisierung erläutert werden.

2.1 Automatische Textklassifikation und Dokumente

Für die IPIB wurden Websites bisher anhand ihres Textes manuell Klassifiziert, wie in Kapitel 1.2 beschrieben. Das Ziel der Bachelorarbeit war es, diese Textklassifikation zu automatisieren. Das allgemeine Ziel einer automatischen Textklassifikations-Aufgabe ist es, einem Dokument durch einen Klassifikator eine Klasse zu zuweisen. Gegeben ist demnach mindestens ein Dokument und eine feste Menge von Klassen. Die vom Klassifikator vorhergesagte Klasse stellt die Lösung der Aufgabe dar. [7, p. 591]

Unter einem Dokument wird hierbei eine Texteinheit verstanden. Dies kann beispielsweise ein ganzes Buch oder auch nur ein einzelnes Kapitel sein. [8, p. 4] Im Rahmen der Bachelorarbeit ist ein Dokument der englische Text einer Website. Da die Klassifikation über den Text erfolgt, wird davon ausgegangen, dass die automatische Klassifikation weniger Fehlentscheidungen produziert, wenn die Dokumente einsprachig sind. Es wurde Englisch gewählt, weil der Großteil der Firmen-Websites aus der IPIB international ist und somit englischen Text beinhalten. Das Finden neuer Firmen über die Twitter-Feeds resultiert dadurch erfahrungsgemäß auch in englischsprachigen Websites. Des Weiteren besteht ein erhöhtes Interesse an englischsprachigen Firmen-Websites, da die Benutzeroberfläche der IPIB ausschließlich englisch ist.

Die feste Menge der Klassen besteht hier aus der Klasse der Firmen-Websites, die relevant für die IPIB sind, und der Klasse der Firmen-Websites, die es nicht sind. Hat die Menge der Klassen, wie hier, eine Kardinalität von zwei, spricht man auch von einer Binärklassifikation. Die Klasse, an der Interesse besteht, wird üblicherweise als positiv bezeichnet. Die andere Klasse heißt negativ. [9, p. 73]

2.2 Information-Retrieval und Korpus

Die automatische Klassifikation gehört zu dem Forschungsgebiet Information-Retrieval. Es beschäftigt sich mit der Findung unstrukturierter Daten innerhalb großer Datenmengen. Das Ziel ist es dabei ein Informationsbedürfnis zu befriedigen. [8, p. 1] Bei der

Textklassifikation sind die unstrukturierten Daten das, was oberhalb bereits als Dokumente eingeführt wurde. Wie die Definition des Information-Retrievals bereits implizierte, wird ein Information-Retrieval-System über eine Menge von Daten erstellt. Ein Textklassifikations-System wird daher über einer Menge von Dokumenten erstellt. Diese Dokumentenmenge wird als Korpus bezeichnet. [8, p. 4] Ein Korpus bezeichnet im Allgemeinen eine Sammlung von schriftlichen oder gesprochenen Äußerungen und die dazugehörigen Metadaten und Annotationen. [7, pp. 483-484]

2.3 Regelbasierter vs. maschineller Ansatz der automatischen Klassifikation

Eine automatische Klassifikation kann entweder nach dem Ansatz der regelbasierten oder nach dem Ansatz der maschinellen Klassifikation erfolgen. Die folgenden Ausführungen basieren auf [8, p. 236] und [10]. In beiden Quellen werden diese Ansätze vorgestellt und verglichen.

Beim regelbasierten Ansatz erfolgt die Klassifikation durch manuell erstellte Regeln. Diese enthalten Schlüsselwörter, deren Vorkommen oder Abwesenheit die Klasse bestimmen. Zum Erstellen der Regeln wird spezialisiertes Personal benötigt, welches sich mit der Domäne auskennt und gleichzeitig weiß, wie effektive reguläre Ausdrücke für die Regeln anzufertigen sind. Das Erstellen der Regeln ist zudem zeitintensiv. Des weiteren müssen die Regel aktuell gehalten werden.

Bei dem Ansatz der maschinellen Klassifikation werden die Entscheidungskriterien automatisch aus Trainingsdaten erlernt. Es sind daher keine Domänenexpertinnen oder -experten nötig. Es bleibt dennoch ein manueller Aufwand, denn die Trainingsdaten müssen mit einem Klassenlabel versehen werden. Diese manuelle Klassifikationsaufgabe ist jedoch bedeuten einfacher als das schreiben von Klassifikationsregeln. In der Textklassifikation handelt es sich bei den Trainingsdaten um eine Menge von gelabelten Dokumenten. Die Menge wird daher auch als Trainingskorpus bezeichnet.

Soll ein automatische Klassifikationsaufgabe gelöst werden und es sind keine Trainingsdaten vorhanden, sollte man über das Verwenden des regelbasierten Ansatzes nachdenken. Für die Bachelorarbeit wurde der Ansatz der maschinellen Klassifikation gewählt. Ein Grund dafür war, dass ein Teil des Trainingskorpus bereits indirekt vorhanden war. Dabei handelte es sich um die Teilmenge der positiven Klasse, denn die

URLs der relevanten Firmen-Websites waren bereits in der IPIB hinterlegt. Ein weiter Grund war, dass keine Domänenexpertinnen oder -experten benötigt werden. Dazu kommt, dass eine Aktualisierung der Entscheidungskriterien beim regelbasierten Ansatz erneut viel Aufwand bedeutet. Für das Aktualisieren beim maschinelle Ansatz muss dagegen nur das Trainingskorpus aktualisiert und die Lernphase erneut ausgeführt werden.

2.4 Maschinelle Klassifikation als Teilgebiet des maschinellen und überwachten Lernens

Der Ansatz der maschinellen Klassifikation ist ein Teilgebiet des maschinellen Lernens (engl. machine learning). Das maschinelle Lernen und somit auch die maschinelle Klassifikation behandeln das Erzeugen von Wissen aus Erfahrung. Ein System lernt dabei aus Beispielen und besitzt danach die Fähigkeit zu verallgemeinern. Auf diese Weise kann es, basierend auf den Erfahrungen, Entscheidungen für unbekannte Situationen treffen. Der Lernvorgang kann als das Finden einer Funktion angesehen werden. Der Ausgabewert der Funktion stammt dabei aus der diskreten Menge der definierten Klassen. [8, p. 237]

Im Bereich des maschinellen Lernens bildet die Klassifikation zusammen mit der Regression das Teilgebiet des überwachten Lernens (engl. supervised learning). [11] Wie in [9, pp. 55-56] und [8, p. 237] beschrieben, zeichnet sich das überwachte Lernen dadurch aus, dass im Lernvorgang Paare von Ein- und Ausgabewerten benötigt werden. Die verwendeten Paare bilden die bereits mehrfach erwähnte Trainingsmenge. Die Funktion selbst wird anhand dieser Paare generiert. Die so erlernte Funktion wird als Modell bezeichnet. Das überwachte Lernen hat seinen Namen daher, dass die Trainingsdaten vorab mit einer Klasse gelabelt werden müssen. Die Person, die diese Aufgabe ausführt, übernimmt dabei die Rolle des Aufsehers (engl. supervisors), der durch die gesetzten Klassenlabel den Lernprozess steuert. Im Rahmen der Textklassifikation stellt der Eingabewert eine Beschreibung eines Dokumentes dar. [8, p. 237] Was genau die Beschreibung eines Dokumentes ist, wird unterhalb im Kapitel 2.7 erklärt. Der Trainingsalgorithmus zum Trainieren des Modells variiert je nach Klassifikationsverfahren. Diese werden in Kapitel 2.8 grob vorgestellt. Allen gemein ist jedoch, dass sie das Klassifikationskriterium anhand der Trainingskorpus selbstständig erlernen bzw. dem Modell beibringen. Sie finden sozusagen die Gemeinsamkeiten innerhalb einer Klasse.

2.5 Evaluieren eines Modells

Bevor ein angelerntes Modell zum Klassifizieren benutzt wird, sollte seine Performanz bzw. Genauigkeit evaluiert werden, um festzustellen, in wie weit die getroffenen Klassifikationsentscheidungen richtig waren. Wie in [9, pp. 71] beschrieben, wird für die Testphase, wie auch schon für die Lernphase, eine gelabelte Dokumentenmenge benötigt. Nähere Ausführungen dazu erfolgen in Kapitel 2.5.2. Die aus der Testphase resultierenden Ergebnisse werden genutzt, um die Performanz des Modells für spätere Klassifikationsentscheidungen im Produktivbetrieb abzuschätzen.

2.5.1 Maße

Die gängigen Maße zur Evaluation eines Klassifikationsmodells werden in [9, pp. 71] erklärt. Im Folgenden werden daraus die, im Rahmen der Bachelorarbeit verwendeten, Maße vorgestellt. Als Erklärungsgrundlage wird dafür ein Textklassifikationsproblem binärer Natur angenommen. Zum Berechnen der Maße muss zunächst eine Wahrheitsmatrix aufgestellt werden. Sie stellt gegenüber wie viele der als positiv und negativ klassifizierten Dokumente wirklich positiv und negativ sind. Dies ist möglich, weil die Testmenge, wie bereits erwähnt, mit Klassenlabeln versehen ist. Das Grundgerüst einer Wahrheitsmatrix ist in Tabelle 1 dargestellt.

	Als positiv klassifiziert	**Als negativ klassifiziert**
Tatsächlich positiv	TP	FN
Tatsächlich negativ	FP	TN

Tabelle 1: Wahrheitsmatrix

TP steht für true positive bzw. richtig positiv und gibt an, wie viele positive Dokumente richtig als positiv klassifiziert wurden. FP steht für false positive bzw. falsch positiv und gibt an, wie viele negative Dokumente fälschlicherweise als positiv klassifiziert wurden. Die Definition von TN (true negative) und FN (false negative) aus der Spalte, der als negativ klassifizierten Dokumente, erfolgt analog.

Aus der Wahrheitsmatrix lässt sich die Accuracy berechnen. Sie ist das Hauptmaß zur Evaluation eines Modells. Sie beziffert die Genauigkeit eines Modells und nimmt einen reellen Zahlenwert von 0 bis einschließlich 1 an. Die Accuracy gibt den Anteil der Dokumente an, die richtig klassifiziert wurden. Die Formel lautet demnach:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.1}$$

Die Accuracy gibt zwar die Genauigkeit eines Modells an, sagt aber nichts über die Performanz des Modells im Bezug auf die einzelnen Klassen an. Sei z.B. ein schlechtes, binäres Textklassifikations-Modell gegeben, dass alle Dokumente als positiv klassifiziert. Dieses Modell würde auf einer Dokumentenmenge, bestehend aus einem negativen Dokument und 99 positiven Dokumenten, eine Accuracy von 99 % erzielen. Dabei würde es 100 % der positiven Dokumente richtig klassifizieren, aber 0 % der negativen. Es ist daher sinnvoll weitere klassenspezifische Maße zur Beurteilung eines Modells heranzuziehen.

Aus der Wahrheitsmatrix lassen sich ebenfalls die Maße Precision und Recall errechnen. Diese Maße beziehen sich im Gegensatz zur Accuracy auf nur eine Klasse und werden daher auch Class-Precision und Class-Recall genannt. Die aufgeführten Formeln beziehen sich exemplarisch auf die Precision und den Recall der Klasse positiv.

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN} \tag{2.2}$$

Precision und Recall nehmen demnach reelle Zahlenwerte von 0 bis einschließlich 1 an. Die Precision gibt an, wie viele positive Dokumente richtig als positiv klassifiziert wurden. Der Recall gibt an, wie viele der Dokumente, die als positiv klassifiziert wurden, tatsächlich positiv sind. Die Precision ist demnach ein Maß für die Genauigkeit und der Recall ein Maß für die Vollständigkeit mit der eine Klasse klassifiziert wird.

2.5.2 Verfahren

Im Abschnitt 2.5.1 wurde bereits erklärt, mit welchen Maßen ein Klassifikationsmodell beurteilt werden kann. Dazu wurde eine gelabelte Testmenge von Dokumenten benutzt. Damit die Evaluationsergebnisse eine möglichst genaue Abschätzung der Performanz des Modells im Produktiveinsatz ermöglichen muss die Testmenge möglichst repräsentativ für die, im Produktiveinsatz auftretenden, Dokumente sein. Des Weiteren darf sich die Testmenge nicht mit der Trainingsmenge überschneiden. Wird der gesamte Korpus zum trainieren und testen verwendet, führt dies, wie in [7, p. 154] beschrieben, zu einer Überanpassung (engl. overfitting, overtraining) des Modells. Das Modell ist in diesem Fall so sehr an den Korpus angepasst, dass es auf ihm eine sehr hohe Accuracy von z.B.

100 % liefert, aber bei unbekannten Dokumenten versagt. Aus diesem Grund gibt es verschiedene Evaluationsverfahren. Die im Folgenden erklärten Verfahren wurden [9, pp. 71-72] entnommen.

2.5.2.1 Holdout-Set

Bei dem Verfahren des Holdout-Sets wird das zur Verfügung stehende Korpus in zwei disjunkte Teilmenge auf gespaltet. Das Größenverhältnis ist dabei üblicherweise 50-50 oder 2/3 des Korpus als Trainingsmenge und 1/3 als Testmenge. Dieses Verfahren wird in der Regel auf große Korpora angewendet.

2.5.2.2 Cross-Validation

Die Cross-Validation wird auch X-Validation oder im Deutschen Kreuzvalidierung genannt. Sie wird in [7, p. 154] beschrieben. Demnach wird das Korpus in n gleichgroße disjunkte Mengen aufgespalten. Eine Menge dient dabei als Testmenge und der Rest als Trainingsmenge. Die Validierung wird n-mal wiederholt, wobei jede der n Mengen genau einmal die Testmenge ist. Das Ergebnis der Cross-Validation ist die durchschnittliche Accuracy der n Ausführungen. Zusätzlich liefert „[...] die Standardabweichung ein Maß für die Stabilität der Evaluationsergebnisse“ [7, p. 155]. Wie in [9, pp. 72-73] beschrieben, kommen je nach Größe des Korpus zwei unterschiedliche Methoden der Cross-Validation zum Einsatz. Bei einem kleinen Korpus findet üblicherweise die n-fold Cross-Validation Verwendung. Bei ihr erfolgt die Aufteilung des Korpus in n gleichgroße Mengen. Typische Werte für n sind fünf und zehn. Handelt es sich um einen sehr kleinen Korpus sollte die leave-one-out Cross-Validation verwendet werden. Bei ihr entspricht die Anzahl der Teilmengen, in die der Korpus zerlegt wird, der Anzahl der Dokumente in ihm. Die Testmenge eines Durchlaufs enthält demnach immer nur ein Dokument. Im Rahmen der Bachelorarbeit umfasste das Korpus bis zu 1490 Dokumente. Es wurde daher eine 10-fold Cross-Validation verwendet.

2.6 Linguistische Grundlagen (Wort und Wortform)

Bevor näher darauf eingegangen wird, wie ein Dokument bei der Textklassifikation beschrieben wird, sind die linguistischen Begriffe Wort und Wortform zu klären. Eine allgemeingültige Definition von Wort ist nicht existent. Im Rahmen der Bachelorarbeit wird die in [7, p. 236] getätigte Definition verwendet. Sie besagt, dass es sich bei einem Wort um eine abstrakte Einheit handelt. Sie liegt verschiedenen Formen zugrunde und

entspricht dem Eintrag eines Wörterbuchs. Eine Wortform dagegen ist die flektierte Form eines Wortes. [7, pp. 236-237] Sie ist als Instanz zu verstehen und demnach im Gegensatz zum Wort ein direkter Bestandteil eines Textes. Ein Wort kann als Äquivalenzklasse von Wortformen verstanden werden.

2.7 Beschreibung eines Dokumentes

Die Beschreibung eines Dokumentes, wie sie als Eingabewert für die Lernfunktion und das trainierte Modell verwendet wird, hängt von dem verwendeten Klassifikationsverfahren ab. Allen gemein ist jedoch, dass ein Dokument in Form eines Vektors repräsentiert wird, dessen Elemente als Merkmale bezeichnet werden. [9, p. 55] Jedes Merkmale nimmt dabei einen Zahlwert an, der als Gewicht bezeichnet wird. [9, p. 187] Bei der Textklassifikation werden die Merkmale mit eindeutigen Bestandteilen eines Dokumentes assoziiert.

2.7.1 Bestandteile eines Dokumentes

Als Bestandteile eines Dokumentes können z.B. Buchstaben, Wortformen oder Wörter verwendet werden. Da Dokumente bei der Textklassifikation aufgrund ihres thematischen Inhaltes klassifiziert werden sollen, sollten die Bestandteile jedoch semantische Repräsentanten der Sprache sein. Somit entfallen Buchstaben und Wortformen. Da Wörter Äquivalenzklassen von Wortformen sind, fassen sie Wortformen gleicher Semantik bzw. gleicher Bedeutung zu einer Klasse zusammen. Für die Bachelorarbeit wurden deshalb Wörter als Bestandteile verwendet.

Ein Merkmal verwendet die Bestandteile in Form eines n-Gramms. N-Gramme werden in [9, p. 204] als Folge von n aufeinanderfolgenden Wortformen definiert. Die Bildung von Wortform-n-Grammen aus einer Zeichenfolge geschieht, durch das Verschieben eines Fensters der Größe n um jeweils eine Wortform. Die Verschiebung wird dabei über die gesamte Länge der Zeichenfolge angewendet und beginnt an deren Anfang. N-Gramme müssen aber nicht aus Wortformen bestehen, sondern können ebenso aus anderen Bestandteilen, wie Buchstaben oder Wörtern, gebildet werden. [7, p. 558] Die Verwendung einzelner Bestandteile als Merkmal entspricht einem 1-Gram. Die in Kapitel 3.4 vorgestellt Spracherkennung von Google basiert z.B. auf Buchstaben-n-Grammen von n = 1, n = 2 und n = 3. [12] Die Länge der verwendbaren n-Gramme wird maßgeblich durch den Umfang des Korpus bestimmt. Je kleiner dieser ist, um so geringer ist

auch die Wahrscheinlichkeit, dass ein bestimmtes n-Gramm noch einmal in ihm vorkommt. [7, p. 270] Im Rahmen der Bachelorarbeit wurden 1-Gramme verwendet, da diese als angemessen für die Korpusgröße erachtet wurden. Des Weiteren wird davon ausgegangen das eine Unterscheidung einzelner Wortfolgen innerhalb der Intellectual-Property-Domäne keinen Mehrwert liefert. Es wird angenommen, dass die typischen Worte bereits einzeln, aber in ihrer Gesamtheit, die IP-Domäne erkennen lassen. Damit ist gemeint, dass keinen benachbarten Wörter nötig sind, um den Kontext des IP zu erkennen, sondern dass das generelle Auftreten einzelner Wörter an beliebigen Stellen im Dokument aussagekräftig genug ist.

2.7.2 Tokenisierung

Im Rahmen der Bachelorarbeit sollen Dokumente durch ihre Wörter repräsentiert werden. Unter Verwendung von [8, pp. 21], werden im Folgenden die, auch in der Bachelorarbeit verwendeten, Schritte vorgestellt, die nötig sind, um die Worte eines Dokumentes zu extrahieren. Die praktische Umsetzung dieses Vorgangs wurde mit dem Programm RapidMiner ausgeführt. Nähere Erläuterungen dazu erfolgen in Kapitel 4.1. Zunächst muss die Zeichenkette eines Dokumentes zerlegt werden. Die resultierenden Teile werden als Tokens bezeichnet. Der Vorgang des Zerlegens heißt Tokenisierung. Während der Tokenisierung werden üblicherweise auch Zeichen, wie z.B. Satzzeichen, der Zeichenkette verworfen. Im Rahmen der Arbeit sind diese Tokens zunächst die Wortformen des Dokumentes. In der Praxis entsprechen die Tokens allerdings je nach Tokenisierung und Dokument nicht immer Wortformen. Um die Wortformen in die abstrakte Form von Wörtern zu verwandeln, lohnt sich das Missachten von Groß- und Kleinschreibung. Dies wird durch eine Konvertierung aller Zeichen in z.B. Kleinschreibung erzielt. Auf diese Weise wird z.B. eine Wortformen, die am Satzanfang steht, aber üblicherweise kleingeschrieben wird, in die gleiche Zeichenfolge konvertiert, die auch die kleingeschriebenen Vorkommen der gleichen Wortform innerhalb eines Satzes aufzeigen. Nach diesem Schritt ist die syntaktische Ebene der Sätze entfernt. Als nächstes und letztes muss noch die syntaktische Ebene der Wortform entfernt werden. Eine mögliche Methode zur Umsetzung ist das Stemming bzw. die Stammformreduktion. Dabei werden in der Regel durch einen heuristischen Prozess, also ohne Lexikon, Suffixe von Wortformen abgetrennt, die nicht zum Wortstamm gehören. Eine exaktere Methode ist die Lemmatisierung bzw. die Grundformreduktion. Sie verwendet ein Vokabular und

eine morphologische Analyse der Wortformen. Der angestrebte Rückgabewert ist die Grundform eines Wortes bzw. die Wortform, die im Wörterbuch steht. Dadurch werden im Gegensatz zum Stemming z.B. Verben und deren Substantivierung getrennt, wenn sie sich in ihrer Grundform unterscheiden. Lemmatisierung vereint dementsprechend die unterschiedlichen Flexionsformen einer Grundform, während Stemming von einander abgeleitete Wortformen vereint. Im Rahmen der Bachelorarbeit wurde der Ansatz des Stemmings gewählt. Er ist laut [7, p. 383] die übliche Ansatz für die englische Sprache. Des Weiteren wird davon ausgegangen, dass Stemming durch seinen stärkeren Reduktionscharakter besser für die Korpusgröße geeignet ist. Ein Weiterer und entscheidender Grund war, dass Lemmatisierung in der Testumgebung für die Klassifikationsverfahren bzw. im Programm RapidMiner nicht zur Auswahl stand. Unter der Verwendung von Stemming ist die am Anfang des Absatzes getätigte Aussage, dass die Wörter eines Dokumentes extrahiert werden, nicht ganz richtig gewesen. Mit der verwendeten Definition von Wort, wäre die Aussage unter Verwendung der Lemmatisierung korrekt. Die Stammform stellt jedoch, wie bereits beschrieben wurde, eine noch abstraktere Einheit als ein Wort dar.

Durch die beschriebenen Verarbeitungsschritte wird die Anzahl unterschiedlicher Tokens innerhalb eines Dokumentes deutlich reduziert. Sie wurden sozusagen semantisch zusammengefasst. Wird im Folgenden von Tokens gesprochen, ist damit die Menge und Form der Tokens gemeint, die nach dem Stemming vorhanden ist. Um das Klassifikationsergebnis weiter zu verbessern, ist das Reduzieren der Tokens auf diese, die inhaltlich relevant sind, ein weiterer sinnvoller Schritt. Zu diesem Zweck wird die sogenannte Stoppwortfilterung angewendet. Sie entfernt „[...] Funktionswörter und andere sehr unspezifische Wörter (z.B. neu, riesig) [...]“ [7, p. 383]. Zu den Stoppwörtern zählen u. a. „[...] alle Wörter der geschlossenen Wortarten, wie Artikel oder Präposition“ [7, p. 588]. Die Stoppwortfilterung erfolgt über eine Stoppwortliste. [7, p. 588] Sie sollte vor dem Stemming oder der Lemmatisierung angewendet werden, da die Stoppwörter-Tokens durch diese Verfahren u. U. verändert und somit nicht mehr entfernt werden.

2.7.3 Vektor eines Dokuments

Wie bereits oberhalb erwähnt wurde, werden Dokumente durch Vektoren repräsentiert und die Elemente bzw. Merkmale dieses Vektors werden mit Bestandteilen eines Dokuments assoziiert. Ein Merkmal ist jedoch kein extrahierter Token, sondern ein sogenann-

ter Type. Während ein Token, ähnlich einer Wortform, eine Instanz ist, ist ein Type eine Klasse identischer Tokens. [8, p. 21] Der Vektor eines Dokuments enthält aber nicht nur die Types, die in seinem Dokument vorkommen, sondern die aller Dokumente der Trainingsmenge. Dafür werden die aus den Dokumenten der Trainingsmenge extrahierten Tokens verwendet, um ein Vokabular aus Types anzulegen. Die Anzahl der Types in der Trainingsmenge bzw. die Dimension des Vokabulars ist gleichzeitig die Dimension des Vektors. Jedes Merkmal des Vektors steht so für einen Eintrag im Vokabular. [9, p. 187] Die Tokens, die einen Eintrag im Vokabular haben, werden auch als Terme bezeichnet. [8, p. 3]

2.7.4 Gewichte der Merkmale und deren Maße

Wie bereits in der Einleitung des Kapitels angedeutet wurde, nimmt jedes Merkmal eines Vektors einen Zahlwert an, der Gewicht genannt wird. Kommt ein Merkmal bzw. Term des Vektors nicht in einem Dokument vor, wird sein Gewicht auf null gesetzt. [9, p. 187] Das Gewicht hat Einfluss darauf, welcher Klasse ein Dokument zugeordnet wird. Es beziffert sozusagen die Relevanz eines Terms für eine Klasse. Dafür können verschiedene Maße zum Einsatz kommen, die teilweise von Klassifikationsverfahren vorgegeben werden. Diese Maße werden im Folgenden vorgestellt. Ein sehr einfaches, binäres Maß ist das Vorkommen eines Terms in einem Dokument (engl. term occurrence). Dieses kommt z.B. in den Bernoulli-Varianten des Klassifikationsverfahrens Naive Bayes zum Einsatz. [9, pp. 96-97] Der Nachteil dieses Maßes ist, dass die Wahrscheinlichkeit, dass ein Term in einem Dokument vorkommt, mit der Länge des Dokumentes steigt. Des Weiteren sinkt die Relevanz eines einzelnen Terms für ein Dokument, je Länger dieses wird. Das schlichte Vorkommen eines Terms ist demnach kein guter Indikator für seine Relevanz innerhalb des Dokuments. Ein weiteres Maß ist die Termfrequenz (engl. term frequency). Es ist die absolute Häufigkeit eines Terms innerhalb eines Dokuments. Die Termfrequenz hat den Vorteil, dass Terme, die häufig in einem Dokument vorkommen, höher gewichtet werden. Es gilt auch umgekehrt, dass Dokumente, die einen Term häufiger enthalten als andere, im Bezug auf diesen Term höher gewichtet werden. [8, p. 15] Die Termfrequenz findet beispielsweise bei den multinomialen Varianten des Klassifikationsverfahrens Naive Bayes Anwendung. Bei der Termfrequenz erfolgt die Beurteilung der Relevanz nur anhand ein einzelnen Dokuments. Der Korpus wird dabei nicht beachtet. Dies ist ein Nachteil, wenn davon ausge-

gangen wird, dass ein Term weniger relevant für ein einzelnes Dokument ist, wenn er in vielen Dokumenten des Korpus vorkommt. Diesen Nachteil beseitigt das tf-idf-Maß. Für einen Term multipliziert es die Termfrequenz mit der inversen Dokumentenfrequenz. [8, pp. 108-109] Die inverse Dokumentenfrequenz ist dabei der Logarithmus aus dem Verhältnis der Korpusgröße zur Dokumentenfrequenz des Terms. Mit Korpusgröße ist die Anzahl aller im Korpus befindlichen Dokumente gemeint. Die Dokumentenfrequenz ist die Anzahl der Dokumente im Korus, die den Term enthalten. Der Nachteil des tf-idf-Maßes ist, dass bei der inversen Dokumentenfrequenz nur das bloße Vorkommen des Terms in anderen Dokumenten beachtet wird und nicht die Häufigkeit des Vorkommens. Das tf-idf-Maß und auch die Termfrequenz kommen typischerweise bei den Klassifikationsverfahren der Support-Vector-Machines zum Einsatz.

2.8 Klassifikationsverfahren

Durch die Wahl des maschinellen Klassifikationsansatzes, musste eine Entscheidung für ein konkretes Klassifikationsverfahren getroffen werden. Im Rahmen der Bachelorarbeit wurden die populären Klassifikationsverfahren k-Nearest-Neighbor (k-NN), logistische Regression, Naive Bayes und Support-Vector-Machines (SVM) auf dem Korpus getestet. Für die Tests wurde das Programm RapidMiner verwendet. Nähere Ausführungen dazu erfolgen im Kapitel 4. Von den vier genannten Klassifikationsverfahren existieren vielen, verschiedenen Abwandlungen. Dies ist auch aus den unterschiedlichen Klassifikations-Operatoren ersichtlich, die in RapidMiner zur Verfügung stehen und getestet wurden. Die vier genannten Klassifikationsverfahren werden im Folgenden sehr grob vorgestellt. Auf eine detaillierte, tiefgreifende Erklärungen wurde verzichtet. Sie würden den Umfang der Bachelorarbeit übersteigen. Des Weiteren ist dieses tiefere Verständnis für das Verstehen der restlichen Arbeit nicht notwendig. Eine detailliertere Beschreibung der Verfahren k-NN, Naive Bayes und SVM findet sich in [9, pp. 91] sowie [8, pp. 234]. Eine detailliertere Beschreibung der logistischen Regression erfolgt in der Online-Vorlesung „Machine Learning“ von Andrew Ng unter [13]. Aus den genannten Gründen, wurde auch auf eine Erklärung und Differenzierung der einzelnen Klassifikation-Operatoren in RapidMiner verzichtet.

2.8.1 k-NN

Das Klassifikationsverfahren k-Nearest-Neighbor (k-NN) interpretiert den Vektor eines Dokuments als Punkt in einem hochdimensionalem Raum. Unterverwendung eines Distanzmaßes berechnet es die k Dokumente, die in diesem Raum den kleinsten Abstand zum Punkte des Eingabedokumentes haben. Die Klasse, der die Mehrheit der k Punkte angehören, wird auch dem Eingabedokument zugewiesen. [9, pp. 112]

2.8.2 Logistische Regression

Die logistische Regression basiert auf der linearen Regression. Diese unterteilt die Dokumente nicht in zwei Klassen sondern nähert eine Funktion an. Sie nimmt dann das Ergebnis der linearen Regression als Eingabe für die Sigmoid-Funktion. Diese liefert einen Wert zwischen null und eins zurück. Dieser Wert dient zum zuweisen einer Klasse an das Dokument. [13]

2.8.3 Naive Bayes

Naive Bayes legt jeder Klasse eine Verteilung zugrunde. Es wird angenommen, dass eine Klasse ein Dokument aus den Merkmalen unter Berücksichtigung der Verteilung generiert. Für ein gegebenes Dokument wird dann für jede Klasse die Wahrscheinlichkeit berechnet, mit der diese Verteilung das Dokument generiert hat. Das Eingabedokument wird der Klasse zugewiesen, von deren Verteilung es am wahrscheinlichsten generiert worden wäre. [9, pp. 87]

2.8.4 Support-Vector-Machines

Das Verfahren der Support-Vector-Machines (SVM) stellt die Dokumentvektoren in einem hochdimensionalem Raum dar und versucht ein bestmöglich Hyperebene zu finden, die die Dokumente der beiden Klassen im Raum separiert. [9, pp. 97]

2.9 Getroffene Entscheidungen und deren Folgen

In den vorangehenden Unterkapiteln wurden Entscheidungen getroffen, die Einfluss auf die Erarbeiten der Problemlösung hatten. Dazu zählte maßgeblich die Wahl des maschinellen Klassifikationsansatzes. Aus ihr resultierte die Notwendigkeit einen Korpus zu erstellen. Die Dokumente sollten aus Websites generiert werden. Zum Erstellen des Korpus mussten daher diverse Websites heruntergeladen werden, um an ihren Texte zu gelangen. Eine Beschreibung dieses Vorgangs befindet sich in Kapitel 3.3. Des Weiteren wurden der Inhalt der Dokumente insofern eingeschränkt, dass sie nur englischen

Text enthalten sollten. Es war demnach ein Mechanismus nötig, der bei der Dokumentenbildung alle heruntergeladenen Dateien verwirft, welche nicht ausschließlich englischen Text beinhalten. Die Entwicklung dieses Mechanismus wird in Kapitel 3.4 beschrieben. Die Wahl des maschinellen Klassifikationsansatzes hatte außerdem zur Folge, dass ein Klassifikationsverfahren gewählt werden musste. Die Auswahl des besten Verfahrens wurde durch Testen getroffen. Dazu wurde das Programm RapidMiner verwendet. Eine Beschreibung der Tests und deren Auswertung erfolgt in Kapitel 4.

3 Erstellen des Korpus

Wie bereits im Kapitel 2 beschrieben, wird ein Korpus benötigt, um die Klassifikationsmodelle zu trainieren und zu testen. Ebenso muss das zu erstellende Klassifikationsprogramm später aus seiner Eingabe, der heruntergeladenen Website, ein Dokument generieren können. Der hier entwickelte Mechanismus, mit dem das Korpus für die RapidMiner-Analysen aus Kapitel 4 erstellt wurde, ist dabei der selbe Mechanismus, der später auch im Kommandozeilen-Programm verwendet wurde, um aus einer Eingabe ein Dokument zu generieren. Bei der Generierung sollten nur englische Texte verwendet werden. Alle menschlichen Sprachen, die nicht Englisch sind, werden folglich im Rahmen dieser Arbeit als Fremdsprachen bezeichnet.

Ein Dokument wurde als der englische Teil einer Website festgelegt. Um das Korpus zu erstellen bedarf es demnach lokaler Kopien der Websites. Als Erstes ist somit eine Menge von URLs nötig, um damit die entsprechenden Websites zur Weiterverarbeitung herunterzuladen. Das Korpus muss beide Klassen abdecken. D. h. auch die Menge der Websites bzw. URLs muss sowohl für die IPIB relevante Firmen, als auch nicht relevante Firmen enthalten.

Wie im Kapitel 2.1 beschrieben, wird bei einem zwei-klassigen Klassifikationsproblem die Klasse als positiv bezeichnet, an der Interesse besteht. Die zweite Klasse wird folglich als negativ betitelt. [9, p. 73] Im Folgenden wird dementsprechend die Klasse als postiv Bezeichnet, welche die für die IPIB relevanten Websites bzw. Firmen vereint. Selbiges gilt auch für die mit der Klasse in Verbindung stehenden Elemente, wie die Dokumente, die URLs der Firmen-Websites und die heruntergeladenen Dateien und Ordner der Websites. Nicht relevante Firmen bzw. Websites und damit in Verbindung stehende Elemente sind somit negativ.

3.1 Bestandteile einer URL (Exemplarisch)

Im den folgenden Kapiteln wird u. a. auf ein paar Bestandteile einer URL eingegangen. Daher sollen die hier wichtigsten zunächst exemplarisch vorgestellt werden.

Gegeben sei die URL „http://www.somewebsite.com/folder/subfolder/index.html?page=2#fazit“, wie auch unterhalb in Abbildung 1 zu sehen. Am Anfang dieser URL steht der Scheme-Teil, welcher hier dem Netzwerkprotokoll HTTP entspricht. „www.somewebsite.com“ bildet den sogenannten Authority-Teil, der hier wiederum nur

aus dem Host-Namen besteht. „/folder/subfolder/index.html“ wird als Path-Teil bezeichnet. Im Folgenden wird hier der Path-Teil in zwei Bereiche unterteilt. Den hinteren Teil bildet das letzte Segment des Path-Teils. Es handelt sich dabei um den Dateinamen „index.html“ mit der Dateierweiterung „html“. Das restliche fordere Stück des Path-Teils wird im folgenen als Ordnerstruktur bezeichnet. Eine URL muss nicht zwingend einen Path-Teil enthalten und im Path-Teil selbst ist wiederum die Ordnerstruktur oder der Dateiname optional. Die restlichen Glieder dieser exemplarischen URL sind der Query-Teil, beginnend mit einem „?“, und der Fragment-Teil, beginnend mit einem „#“. Auf diese beiden Teile soll hier aber nicht näher eingegangen werden. Eine vollständige Beschreibung zum Aufbau einer URL ist im Standardisierungsentwurf RFC 3986 [14] festgehalten. Eine Beschreibung der Syntax, die auch im hier erwähnten Beispiel verwendet wurde, ließ sich auch schon im mittlerweile obsoleten RFC 1738 [15, pp. 2-9] finden.

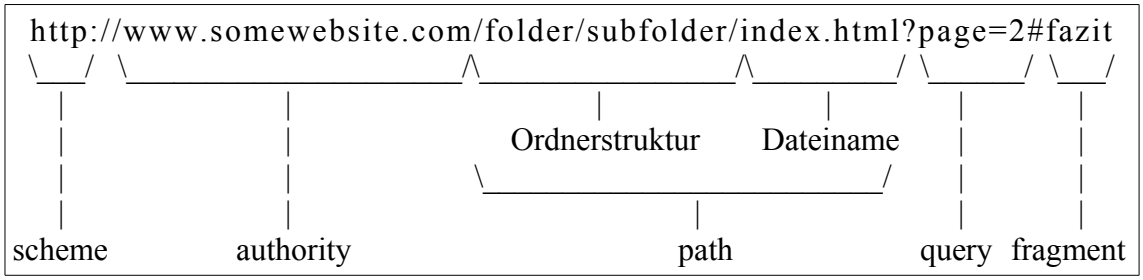

Abbildung 1: Beispielhafte Darstellung einer URI und ihrer Einzelteile basierend auf der Abbildung aus dem RFC 3986 [14, p. 16]. Die originale Abbildung wurde modifiziert und an das hier verwendete Textbeispiel angepasst.

Eine URL verweist immer nur auf eine einzelne Seite einer Website, eine sogenannte Webpage. Eine Website ist dabei die Gesamtheit aller Webpages und Dateien, die zu einem Host gehören, im folgenden als Website eines Hosts bezeichnet.

Wird im folgenden jedoch von der zu einer URL gehörenden Website gesprochen, ist damit nur die Gesamtheit aller Webpages und Dateien eines Hosts gemeint, deren Ordnerstruktur mit der Ordnerstruktur der gegebenen URL übereinstimmt oder beginnt. Also alles, was sich innerhalb oder unterhalb des hierarchisch tiefsten Ordners der Ordnerstruktur befindet. Für die URL „http://www.somewebsite.com/folder/subfolder/index.html“ wäre das z. B. die Datei „http://www.somewebsite.com/folder/subfolder/index2.html“ und die Datei unter

„http://www.somewebsite.com/folder/subfolder/subsubfolder/index.html“, aber nicht „http://www.somewebsite.com/folder/index.html“.

3.2 Beschaffung der URLs

3.2.1 URLs der Klasse positiv

Als positive URLs dienten die URLs aus den Firmen-Profilen der IP Industry Base. Es konnten insgesamt 1854 positive URLs extrahiert werden. Diese wurden teilweise abgeändert, um später bessere Ergebnisse zu erzielen. Zum Beispiel wurden einige URLs, die auf die Kontaktseite einer Firmen-Website zeigten, zu der entsprechenden URL der Startseite abgeändert. Dieser Eingriff war nützlich, da beim Herunterladen der Firmen-Websites später nur die zur URL gehörende Website und nicht die gesamte Website des Hosts heruntergeladen wurde. An dieser Stelle sei bereits auf die beim Herunterladen verwendete Option „--no-parent“ verwiesen, die im Kapitel 3.3.2 erklärt wird.

Des Weiteren wurden einige URLs mehrsprachiger Websites auf ihr englisches Äquivalent geändert, wenn die Ordnerstruktur des Path-Teils darauf hinwies. Zum Beispiel wurde „http://www.wurzer-kollegen.de/de/“ in „http://www.wurzer-kollegen.de/en/“ geändert.

Wie viele Dokumente aus diesen 1854 URLs später tatsächlich generiert werden konnten, war zu diesem Zeitpunkt noch nicht absehbar. Eine Betrachtung der Top-Level-Domains lies jedoch mindestens 1000 englische Websites bzw. Dokumente vermuten.

3.2.2 URLs der Klasse negativ

Neben den Firmen-Profilen verfügt die IPIB unter anderem auch über eine Datenbank-Tabelle mit vorgeschlagenen, neuen Twitter-Nutzern. Diese werden, wie bereits in Kapitel 1.2 beschrieben, durch das überwachen der bekannten Twitter-Nutzer gesammelt.

Eine Betrachtung der bereits manuell klassifizierten Firmen bzw. Datensätze dieser Tabelle zeigte, dass sich die positiven und die negativen Datensätzen in einem Verhältnis von 3:2 befinden. Dieses Verhältnis wurde als Maßstab genommen, um die Anzahl der nötigen negativen URLs festzulegen.

Die Datenbank-Tabelle der vorgeschlagenen Twitter-Accounts enthielt lediglich 198 negative URLs. Die ursprüngliche Idee war diese URLs für den Aufbau des Korpus zu verwenden und mindestens 470 weitere zu beschaffen, um den Verhältnis von 3:2 ge-

recht zu werden. Die abgelehnten Twitter-Accounts wurden schließlich doch nicht verwendet, weil es schneller war alle nötigen negativen URLs über die Website „www.crunchbase.com“ zu beziehen. CrunchBase verfügt über eine Vielzahl von Firmenprofilen. Im Developer-Teil der Website[1] wird die CrunchBase Open Data Map zum Download angeboten. Sie beinhaltet grundlegende Daten der Firmenprofile, darunter auch die URL der Firmenwebsite. Die CrunchBase Open Data Map umfasste zum Zeitpunkt des Downloads 218.924 Profile, wovon 218.407 bzw. ca. 99,76 % die Firmen-URL enthielten. Die Open Data Map wurde im CSV-Format heruntergeladen und stand somit zur Bearbeitung in einem Text-Editor zur Verfügung.

Zunächst mussten alle URLs möglicher IP-Dienstleister und somit positive URLs aus der heruntergeladenen Open Data Map entfernt werden. Dazu wurden alle Profile bzw. Textzeilen gelöscht, die, ohne Beachtung von Groß- und Kleinschreibung, die Zeichenkette „intellect“ oder „patent“ oder den regulären Ausdruck „[^a-z]ip[^a-z]“ enthielten. Der reguläre Ausdruck dient dazu nur Worte zu löschen die, die Abkürzung „ip“ selbst sind oder sie beinhalten, wie z. B. „IP-based“. Worte, wie „partnership“ oder „iPhone“, wurden somit verschont, da sie „ip“ nur als Zeichenfolge beinhalten. Der Nachteil dieser Methode ist, dass auch Profile entfernt werden, deren Texte die Abkürzung IP in einem anderen Sinn beinhalten, z. B. für das Wort Internetprotokoll. Das Löschen zu vieler Profile stellte jedoch kein Problem dar, da nur ein kleiner Bruchteil der Open Data Map verwendet werden sollte. Das Löschen der Profile konnte mit Hilfe der Makro-Aufzeichnungs-Funktion des Texteditors Notepad++ schnell und einfach realisiert werden. Auf diese Weise wurden 387 Profile entfernt, die eventuell IP-Dienstleistern gehörten. Es blieben somit 218.020 URLs übrig.

Aus den übrigen URLs wurde pseudo-zufällig die Anzahl nötiger URLs ohne zurücklegen gezogen. Dazu wurde die Interactive Ruby Shell (IRB) und die Random-Funktion rand() der Klasse Random benutzt. Die folgenden Befehle realisierten den Vorgang:

```
irb(main):001:0> all_urls =
File.readlines('crunchbase_all_urls.txt').map{ |l| l.chomp}
```

1 https://developer.crunchbase.com/

```
irb(main):002:0> selected_urls = [];
i = 0;
while i < 1220 do
  selected_urls << all_urls[Random.rand(all_urls.count)];
  selected_urls.uniq!;
  i = selected_urls.count
end

irb(main):003:0> File.write('negative_urls.txt',
selected_urls.join("\n"))
```

Als Erstes wurde die Textdatei „crunchbase_all_urls.txt“, welche alle übrigen URLs der CrunchBase Open Data Map enthielt, eingelesen. Dabei wurde das String-Array „all_urls“ erstellt. Ein Array-Element entsprach dabei einer URL. Im zweiten Schritt wurde aus diesem Array mittels einer Random-Funktion ein Element bzw. eine URL gezogen und in das neue Array „selected_urls“ geschrieben. Dies wurden solange wiederholt, bis das neue Array die gewünschten 1220 URLs umfasste. Nach jedem Hinzufügen einer URL wurden dabei Duplikate im neuen Array entfernt, um sicherzustellen, dass mehrfach gezogene URLs am Ende nur einmal vorkommen. Im letzten Schritt wurde das neue Array zu einem String zusammengefasst, wobei zwischen zwei Elemente ein Zeilenumbruch eingefügt wurde. Dieser String wurde schließlich als Textdatei „negative_urls.txt“ abgespeichert. Die URLs wurden durch Zeilenumbrüche separiert, da das Programm, welches später zum Herunterladen der Websites der URLs verwendet werden sollte, dieses Format verlangt.

Das Beschaffen der URLs ging, dank der CrunchBase Open Data Map und dem Random-Verfahren, nun in Sekundenschnelle und bedeutete kein langwieriges, händisches Suchen und Prüfen einzelner URLs im Internet. Deshalb wurden auch nicht nur die geplanten 470 URLs gezogen, die 2/3 der auf mindestens 1000 geschätzten positiven, englischen Websites bzw. deren URLs entsprochen hätten, sondern 1220. Dies entsprach etwas mehr als 2/3 der, zu diesem Zeitpunkt bereits fertig heruntergeladenen, nicht nur englischen Websites der positiven URLs und enthielt einen zusätzlichen Puffer. Beim Herunterladen der positiven Websites zeigte sich nämlich, dass ca. 4% der Websites nicht heruntergeladen werden konnten, weil HTTP-Fehler, wie „403 Forbidden“, auftraten.

3.3 Herunterladen der Websites:

Nach dem die positive und die negative URL-Liste erstellt worden war, konnte es an das Herunterladen der Websites gehen. Zum Herunterladen wurde das Kommandozeilen-Programm GNU Wget[1] in der Version 1.13.4 verwendet. Die folgenden Informationen über diese Programmversion, wurden dem Handbuch [16] entnommen. Statt einer einzelnen URL, kann Wget auch eine Textdatei übergeben werden, welche es Zeile für Zeile abarbeitet. Dabei wird jede Zeile als URL interpretiert. Bei dem Erstellen der URL-Listen wurde bereits darauf geachtet diese Form einzuhalten. Dennoch mussten vor dem Herunterladen ein paar Veränderungen an den beiden URL-Listen vorgenommen werden. Diese werden im folgenden Unterkapitel erläutert.

3.3.1 Vorbereitungen

Zum Herunterladen der Websites sollte das Programm Wget benutzt werden. Die verwendete Version interpretiert URLs nach der Syntax des obsoleten RFC 1738. Der Scheme-Teil der URL muss daher explizit definiert werden. Bei diversen URLs fehlte dieser jedoch. Um diesen zu ergänzen, wurde allen URLs, die nicht mit „http“ begannen, ein „http://“ vorangestellt. Dieser Schritt wurde erneut mit Hilfe der Interactive Ruby Shell realisiert. Für jede der URL-Listen wurde dabei folgender Befehl ausgeführt:

```
irb(main):001:0> url_array.map!{|x| if x.start_with?('http')
then x else 'http://'+x end}
```

Der Path-Teil einiger URLs endete nicht auf einen Dateinamen, sonder auf eine Ordnerstruktur. Die URLs hatten jedoch kein „/“ als letztes Zeichen. Der Schrägstrich ist allerdings notwendig, damit Wget das letzte Segment des Path-Teils als Ordner interpretiert und nicht fälschlicherweise als Datei. Die richtige Interpretation der Segmente des Path-Teils war wichtig, für die weiter unten erklärte „--no-parent“-Option des rekursiven Downloads mit Wget. Mit Hilfe der IRB wurden die fehlenden Schrägstriche bei den URLs angehängt, deren letztes Segment des Path-Teils keine Datei war. Für jede der URL-Listen wurde dabei folgender Befehl ausgeführt:

```
irb(main):001:0> url_array.map!{|x| if x.end_with?('/') ||
x.split('/').last.include?('.') then x else x+'/' end}
```

Um bei URLs, die nicht auf „/“ endeten, festzustellen, ob das letzte Segment des Path-Teils eine Datei ist, wurde das Segment lediglich auf das Vorhandensein eines Punktes

1 http://www.gnu.org/software/wget/

untersucht. Ein vorhandener Punkt wurde dabei als Anfang einer Dateierweiterung interpretiert.

Da Wget die URLs der übergebenen Textdatei seriell verarbeitet, wurden die beiden Listen jeweils noch in zehn Textdateien aufgesplittet, so dass mehrere Wget-Instanzen parallel ausgeführt werden konnten, um das Herunterladen der Websites der URLs zu beschleunigen.

3.3.2 Herunterladen der Websites der URLs

Nach den oberhalb beschriebenen Vorbereitungsschritten konnte das Herunterladen der Websites der URLs beginnen. Zunächst wurde für jede URL die dazugehörige Webpage heruntergeladen. Es wurde aber davon ausgegangen, dass der Text der Webpage, auf die eine URL zeigt, in der Regel nicht aussagekräftig genug sein wird, um ausschließlich daraus ein Dokument zu erstellen. Aus diesem Grund wurden weitere Downloads durchgeführt, die mehr als nur die einzelne Webpage einer URL herunterluden. Zu diesem Zweck wurde in Wget die Funktion des rekursiven Downloads benutzt. Dabei wird die zur URL gehörende Webpage bzw. Datei heruntergeladen und ihr Inhalt auf Verlinkungen bzw. weitere URLs untersucht. Die Dateien der gefundenen URLs werden dann ebenfalls heruntergeladen und erneut untersucht. Dies geschieht so lange, bis die festgelegte Rekursionstiefe erreicht ist. Das Untersuchen der Dateiinhalte wird dabei nur auf HTML- und CSS-Dateien angewendet.

Der rekursive Download wurde mit den Tiefen eins und zwei durchgeführt. Welche Tiefe die besseren Ergebnisse bei der Klassifikation liefert, sollte sich später zeigen. Eine stichprobenartige Untersuchen einiger Websites ergab, dass eine Rekursion der Tiefe zwei in der Regel alle interessanten Webpages liefert. Außerdem waren tiefere rekursive Downloads aufgrund der Größe der heruntergeladenen Websites bzw. deren Ordnern, Dateianzahlen und Download-Zeiten unpraktikabel für die rund 3.000 URLs. Alle heruntergeladenen Dateien der Tiefe zwei waren zusammen bereits über 60 GB groß. Einzelne Websites der Tiefe zwei belegten dabei schon mehr als 1 GB Speicherplatz und enthielten mehr als 20.000 oder sogar 35.000 Dateien. Der rekursive Download einer solchen Website dauerte dann teilweise mehrere Stunden.

Die Websites der URLs mussten für jede Rekursionstiefe erneut heruntergeladen werden. Das liegt daran, dass die Pfade der Rekursion nicht zwangsweise den Pfaden innerhalb des Download-Ordners entsprechen. Nur den Download mit der tiefsten Rekur-

sion auszuführen und die restlichen Downloads durch schnelles Kopieren zu ersetzten, ist demnach keine Alternative. Ginge dies, wäre es möglich, den vorhandenen Ordner bis zur gewünschten Ordnerpfadtiefe zu kopieren, statt einen weiteren Download durchzuführen, der die entsprechend kleinere Rekursionstiefe besitzt.

3.3.2.1 Einfacher Download

Zunächst wurde für jede URL nur die Webpage heruntergeladen auf die sie zeigt. Dazu wurde Wget, wie folgt, in der Kommandozeile aufgerufen:

```
$ wget -x -E -R txt,pdf,PDF,ico,gif,jpeg,JPEG,jpg,JPG,png,js,
css,asc,exe,mp4,flv,avi,mpeg,mpg,wmv,wav,ogg,mp3,doc,swf,ppt,
xls -X /cn,/de,/es,/fr,/hk,/hu,/it,/ja,/jp,/ko,/nl,/pl,/pt,
/ru,/swe,/tr,/tw,/zh --input-file=url_list.txt --tries=3
--timeout=120 --no-verbose -nc --restrict-file-names=ascii
```

Die Mehrzahl der hier benutzten Optionen ist erst für die rekursiven Downloads von größerer Bedeutung. Sie werden daher erst im nächsten Abschnitt erläutert.

Die Option „-x“, alternativ auch „--force-directories“, legt fest, dass für jede URL ein Ordner erstellt wird. Dies geschieht sonst nicht, wenn der Download nur eine Datei umfasst. Der angelegte Ordner wird nach dem Host-Teil der URL benannt. Es werden auch alle Ordner der Ordnerstruktur des Path-Teils angelegt.

Die Option „-E“, alternativ auch „--adjust-extension“, ergänzt fehlende Dateierweiterungen von HTML- und CSS-Dateien. So wird für eine Datei des Typs „application/xhtml+xml“ oder „text/html“ die Endung „.html“ ergänzt, wenn die URL nicht auf den regulären Ausdruck „\.[Hh][Tt][Mm][Ll]?“ endet. Analog wird für den Dateityp „text/css“ die Endung „.css“ angehängt.

Die Option „--input-file=DATEI“, alternativ auch „-i DATEI“, dient dazu die Textdatei zu übergeben, welche die URLs für den Download enthält.

Die Option „--no-verbose“, alternativ auch „-nv“, reduziert die Konsolenausgaben von Wget auf Fehlermeldungen und grundlegende Informationen. Diese Option wurde benutzt, um die Log-Datei übersichtlicher zu gestalten.

Die Option „restrict-file-names=MODUS“ führt dazu, dass, je nach Modus, bestimmte Zeichen der URL für die Verwendung als lokalen Dateinamen escaped werden. Zur Auswahl stehen die Modi „unix“, „windows“, „nocontrol“, „ascii“, „lowercase“ und „uppercase“. Wird der verwendete Zeichensatz vom Betriebssystem nicht unterstützt, kann es dazu führen, dass Dateien aufgrund ihres Namens nicht verwendbar sind. Das

Öffnen einer solchen Datei führte z.B. zu einer „NoSuchFileException“, innerhalb des Java-Programms aus Kapitel 3.4.1.

Der Wget-Befehl wurde zusätzlich um den Linux-spezifischen Befehl „2>&1 | tee -a log.log“ erweitert, um alle Kommandozeilenausgaben von Wget in eine Log-Datei zu schreiben. Wget verfügt zwar über eine Option alle Ausgaben in eine Datei zu schreiben, diese ermöglicht jedoch nicht die gleichzeitige Ausgaben in der Kommandozeile.

Der Teil „2>&1“ dient dazu eine Ausgabeumleitung („>“) von „2“, dem Standard-Fehlerkanal (stderr), nach „1“, dem Standardausgabe-Kanal (stdout), durchzuführen. Das „&“ dient dabei zur Referenzierung des Standardausgabe-Kanals. [17]

Der Pipe-Operator „|“ dient dazu den bisherigen Befehl um den Standard-Unix-Befehl „tee“ zu erweitern. „tee“ fungiert als eine Art T-Stück für Kanäle. Es ermöglicht hier den Standardausgabe-Kanal, über den nun auch der Standard-Fehlerkanal läuft, abzuzweigen. Er wird damit nicht nur auf der Kommandozeile ausgegeben, sondern mittels „tee -a log.log“ auch in die Datei „log.log“ geschrieben. Durch die Option „-a“ wird „tee“ dazu veranlasst die Textdatei „log.log“ fortzuführen, statt sie zu überschreiben. [17]

3.3.2.2 Rekursiver Download

Der benutzte Wget-Befehl für die Rekursionstiefe eins lautete:

```
$ wget -r -l 1 --no-parent -E -R txt,pdf,PDF,ico,gif,jpeg,
JPEG,jpg,JPG,png,js,css,asc,exe,mp4,flv,avi,mpeg,mpg,wmv,wav,
ogg,mp3,doc,swf,ppt,xls -X /cn,/de,/es,/fr,/hk,/hu,/it,/ja,/
jp,/ko,/nl,/pl,/pt,/ru,/swe,/tr,/tw,/zh –input-
file=url_list.txt --tries=3 --timeout=120 --no-verbose -nc
--restrict-file-names=ascii
```

Die Option „-r“, alternativ auch „--recursive“, aktiviert den rekursiven Download. Die Standard-Rekursionstiefe ist fünf. Die Option „-x“ wird implizit verwendet.

Die Option „-l TIEFE“, alternativ auch „--level=TIEFE“, legt die maximale Rekursionstiefe fest.

Die Option „--no-parent“, alternativ auch „-np“, begrenzt die Rekursion auf Dateien und Ordner innerhalb und unterhalb des letzten und somit tiefsten Ordners des Path-Teils einer URL. Ist der Path-Teil leer, bewirkt der Befehl dennoch, dass keine URLs anderer Hosts verfolgt und heruntergeladen werden. Die Option war außerdem nötig, da einige Firmen-Websites nur Unterseiten einer anderen Website waren. So z. B. die Fir-

ma IP Venture Fund mit ihrer Website „http://www.ipgroupplc.com/about-us/capital/ip-venture-fund/“ innerhalb der Website der Firma IP Group plc.

Die nächsten beiden Optionen bilden jeweils eine Blacklist für den Download. Sie wurden ihren beiden gegenteiligen Optionen mit Whitelist-Verhalten vorgezogen. Auf diese Weise sollten möglichst viele unnütze Dateien und keine bis wenige nützliche Dateien verworfen werden. Die beiden Listen entstanden erst im Laufe des Downloadprozesses. Einige dieser verworfenen Dateien und Ordner waren daher dennoch vorhanden.

Die Option „-R LISTE“, alternativ auch „--reject LISTE“, führt dazu, dass alle Dateien, die auf eine Dateierweiterung der LISTE enden, nach dem herunterladen gelöscht oder gar nicht erst heruntergeladen werden. Der Query-Teil einer URL wird dabei nicht berücksichtigt. Die hier angegebene Liste beruht u. a. auf Dateierweiterungen, die während des rekursiven Downloads der positiven URLs auftauchten. Die Dateitypen der Liste können ignoriert werden, da sie, nach der Definition aus Kapitel 3.4.1, keine Textdateien sind. Sie liefern somit keinen Mehrwert für das spätere Erstellen der Dokumente. Durch diese Option wurden wahrscheinlich alle Textdateien heruntergeladen, aber auch einige Nicht-Textdateien. Die gegensätzliche Option lädt nur die Dateien mit den angegebenen Dateiendungen herunter. Dadurch wären zwar evtl. nur Textdateien heruntergeladen worden, aber dafür nicht alle. In beiden Fällen wäre es wahrscheinlich zu Nicht-Textdateien gekommen, da eine Dateiendung optional und vor allem bei URLs kein sicherer Indikator für einen Dateityp ist. Als Beispiel sei hier „http://hamburg-innovation.net/file.php/HI_Flyer_engl.pdf-2011-04-20“ genannt.

Die Option „-X LISTE“, alternativ auch „--exclude LISTE“, führt dazu, dass Ordner, die in der Liste aufgeführt sind, nicht heruntergeladen und nicht verfolgt werden. Die hier angegebene Liste beruht auf Ordnernamen, die während des rekursiven Downloads der positiven URLs auftauchten. Diese Ordner kamen auf mehrsprachigen Websites vor. Ihre Namen sind Abkürzungen für Sprachen und so sollten sie auch nur Dateien bzw. Text in der jeweiligen Sprache enthalten. Die angegebenen Ordner wurden ignoriert, da sie, ihrem Sinn nach, keine englischsprachigen Dateien enthalten sollten und somit für das Erstellen der Dokumente nicht von Bedeutung sind. Auch hier wäre die gegensätzliche Option die schlechtere Wahl. Sie würde nur Ordner namens „/en“, „/eng“, „/us“ usw. herunterzuladen. Viele Websites benutzen derartige Unterordner jedoch nicht, vor

allem die einsprachigen Websites. Des Weiteren gibt es auch keine derartige Standardisierung für Ordnernamen.

Die folgenden zwei Optionen wurden gesetzt, da deren Standardwerte dazu führten und führen, dass eine fehlschlagender Download bis zu 5 h in Anspruch nehmen kann.

Die Option „--tries=ZAHL“, alternativ auch „-t ZAHL“, legt fest, wie oft versucht werden soll eine Datei herunterzuladen, bis ihr Download erfolgreich ist. Ausgenommen hiervon sind Downloads mit fatalen Fehlern, wie z. B. der HTTP-Fehler 404 „Not found“. Der Standardwert liegt bei 20 Versuchen.

Die Option „--timeout=SEKUNDEN“, alternativ auch „-T SEKUNDEN“, legt fest, nach wie vielen Sekunden eine Operation abgebrochen werden soll. Als Operation werden dabei folgende Vorgänge verstanden: Der DNS-Lookup, der Aufbau einer TCP-Verbindung und der Leerlauf eines Downloads. Der Standardwert liegt bei 900.

Die Option „-nc“, alternativ auch „--no-clobber“, führt in diesem Befehl dazu, dass Dateien, die bereits heruntergeladen wurden, bei einem erneuten Download nicht noch einmal heruntergeladen werden.

Wie schon im einfachen Download wurde der Wget-Befehl auch hier wieder um „2>&1 | tee -a log.log“ erweitert.

Der angewandte Befehl für das Herunterladen mit Rekursionstiefe zwei unterschied sich nur in der Option „-l TIEFE“.

3.3.3 Nachbesserung

Wie bereits erwähnt befanden sich einige Firmen-Websites innerhalb einer anderen Website, wie z. B. die Firma IP Venture Fund mit der URL „http://www.ipgroupplc.com/about-us/capital/ip-venture-fund/“ innerhalb der Website der Firma IP Group plc. Für einige dieser URLs waren weitere URLs des selben Hosts vorhanden. So z. B. die zur Firma IP Group plc gehörende URL „http://www.ipgroupplc.com“. Der Download mit Wget speichert alle URLs des selben Hosts auch im selben Ordner, da dieser nach dem Host benannt wird. Die Dateien jeder URL sollten jedoch in einem separaten Ordner abgespeichert werden, um später aus jeder URL bzw. jedem Ordner ein einzelnes Dokument zu generieren. Dafür wurden die URLs gleicher Hosts erneut und einzeln heruntergeladen, so dass deren Ordner per Hand eindeutig benannt werden konnten.

Mehrfach vorkommende Hosts wurden mit Hilfe der Interactive Ruby Shell ausfindig gemacht. Dazu wurde der folgende Befehl ausgeführt:

```
irb(main):001:0> url_array.map{|x| x.split('//').last}.map{|x|
x.split('/').first}.group_by{|e| e}.select{|k,v|
v.size>1}.keys
```

Jede URL-Liste wurde zuvor, wie üblich, als Array eingelesen. Für jedes Element des Arrays wurden dann die Zeichen vor und incl. des „//“ entfernt, um das Schema zu löschen. Aus dem restlichen String wurden danach alle Zeichen nach und incl. des ersten „/“ entfernt. Auf diese Weise war nur noch der Host-Teil der URL übrig. Dieses Host-Array wurde dann in einen Hash konvertiert. Als Key diente dabei der Host und als Value die absolute Häufigkeit des Hosts im Array. Der Hash wurde danach auf alle mehrfach vorkommenden Hosts reduziert. Am Ende wurde ein Array aus allen übrigen Key-Werten gebildet, um eine Liste der mehrfach vorkommenden Hosts zu erhalten.

Für die als negativ klassifizierten URLs wurde bereits beim Erstellen der URL-Liste darauf geachtet, dass keine URLs mit identischen Hosts vorkommen. So wurden weitere separate Downloads vermieden.

Bei den Websites der negativen URLs fiel auf, dass 128 URLs kein „www.“ enthielten und dass für sie nur eine Datei heruntergeladen wurde, in allen Rekursionstiefen. Um herauszufinden, ob dieses Verhalten dem fehlenden „www.“ zu zuschreiben war, wurden diese URLs mit ergänztem „www.“ erneut heruntergeladen. Dadurch reduzierte sich die Anzahl dieser Host-Ordner von 128 auf 67. Bei den URLs der IPIB trat derartiges nicht auf. Das Problem lag darin begründet, dass die Websites auch ohne „www.“ erreichbar waren, aber umgeleitet wurden. Dabei wurde das fehlende „www.“ ergänzt und bildete somit einen neuen Hostnamen. Dieser wurde von Wget aufgrund der „--no-parent“-Option jedoch nicht weiter verfolgt.

Das Herunterladen der Firmen-Websites erfolgte, wie beschrieben, in drei verschiedenen Tiefen. Um diese unkomplizierter zu Beschreiben, wird das einfache Herunterladen fortan als Download mit Rekursionstiefe null verstanden. Des Weiteren werden die drei Downloads im Folgenden als Level-0-, Level-1- und Level-2-Download bezeichnet.

Die heruntergeladenen Websites wurden, den drei Rekursionstiefen entsprechend, für jede Klasse in drei Ordner aufgeteilt. Die drei Ordner einer Klasse wurden untereinander Verglichen. Dadurch sollte festgestellt werden, ob der Download vereinzelter URLs für

bestimmte Rekursionstiefen fehlgeschlagen war. Dazu wurden die folgenden Linux-spezifischen Kommandozeilen-Befehle benutzt:

```
$ diff Download_l0 Download_l1 | grep "Nur in"
$ diff Download_l1 Download_l2 | grep "Nur in"
```

Das Kommandozeilen-Programm „diff“ liefert inhaltliche Gemeinsamkeiten und Unterschiede der beiden übergebenen Ordner. Zeilen, die gemeinsame Ordner Beschreiben, fangen dabei immer mit „Gemeinsam“ an. Zeilen, die individuelle Ordner beschreiben, fangen dagegen immer mit „Nur in“ an. Mit dem Pipe-Operator „|“ und dem Programm „grep“ wurde die Ausgabe von „diff“ auf eben diese Zeilen begrenzt.

Alle Ordner bzw. Websites, die im Vergleich fehlten, wurden erneut heruntergeladen, der Rekursionstiefe entsprechend.

Es wurde ebenfalls ein klassenübergreifender Ordnervergleich durchgeführt, um festzustellen, ob sich gemeinsame Hosts unter den positiven und negativen URLs befanden. Dazu wurde wieder der Befehl oberhalb verwendet. Dieses mal jedoch mit der Beschränkung auf die gemeinsamen Ordner. Beim ersten Durchlauf wurden 4 gemeinsame Ordner entdeckt. Das lag daran, dass die Zeichenkette „patent“, zu Beginn, noch kein Ausschlusskriterium für einen Datensatz der CrunchBase Open Data Map war, so wie in Kapitel 3.2.2 beschrieben. Die negative URL-Liste und die heruntergeladenen Websites wurden nach der Entdeckung entsprechend angepasst und die Verluste ersetzt.

Wie bereits oberhalb erwähnt, wurden einige URLs mit ergänztem „www.“ erneut heruntergeladen. Nach der Entdeckung, dass manche Hosts auch ohne den „www.“-Präfix erreichbar sind, wurde ein weiterer klassenübergreifender Ordnervergleich durchgeführt. Dafür wurde diesmal bei jedem Host der Präfix „www.“ entfernt. Die modifizierten Namen der Host-Ordner wurden wieder in der Interactive Ruby Shell analysiert. Dabei wurde der gleiche Befehl verwendet, der oberhalb auch schon für das Auffinden mehrfach vorkommender Hosts eingesetzt wurde. Es wurden keine gemeinsamen Ordner gefunden.

3.4 Erzeugen der Dokumente

Nachdem die Websites aller URLs fertig heruntergeladen waren, konnte es an das Erzeugen der Dokumente gehen. Zu diesem Zweck wurde ein Java-Programm entwickelt. Es sollte aus jedem heruntergeladenen Host-Ordner, also für jede URL, ein Dokument

generieren. Dazu musste es den englischen Text einer Website extrahieren. Das so erzeugte Korpus wurde danach genutzt, um die verschiedenen Klassifikationsmodelle zu trainieren und zu testen.

3.4.1 Programmübersicht

Das entwickelte JAVA-Programm verlangt beim Aufruf die Angabe eines Quellpfades. Die Hierarchie eines solchen Ordnerpfades kann als Baum verstanden werden. Seine Knoten sind dabei Ordner und die Blätter Dateien. Der Wurzelknoten entspricht hier dem Quellordner. Das Programm traversiert den Baum des Quellordners mit einem Tiefendurchlauf. Für alle Kinder des Wurzelknotens wird während der Traversierung ein Dokument erzeugt. Der Quellpfad sollte daher auf einen Ordner verweisen, der nur die Host-Ordner als direkte Unterordner besitzen. Da die heruntergeladenen Websites in drei unterschiedliche Rekursionstiefen und zwei Klassen aufgeteilt waren, musste das Programm sechsmal mit den unterschiedlichen Quellpfaden aufgerufen werden.

Beim Traversieren des Baumes eines einzelnen Kindes, hier Host-Ordners, wird für jede Datei anhand der Dateierweiterung entschieden, ob es sich um eine Textdatei handelt oder nicht. Ist es eine Textdatei, wird sie als String eingelesen. Auf die hier verwendete Definition einer Textdatei wird unterhalb eingegangen.

Beim Einlesen der Datei wird ihr Byte-Stream mit dem Zeichensatz UTF-8 dekodiert. Es wurde UTF-8 gewählt, weil dies die gängige Unicode-Zeichen-Kodierrung im World-Wide-Web ist. Unter den heruntergeladenen Dateien befinden sich auch nur sehr wenige nicht UTF-8 kodierte Dateien. In der Regel enthielten diese Dateien Sprachen aus dem asiatischen Raum. Der Arbeitsaufwand für den Einbau einer Erkennung des richtigen Encodings wäre demnach unwirtschaftlich.

Nach dem Einlesen des Strings werden aus ihm HTML- und XML-Tags sowie URLs entfernt. Die Entfernung der Tags wird mit der Java-Bibliothek jsoup[1] in der Version 1.7.3 realisiert.

Im nächsten Schritt prüft das Programm, in welcher Sprache der String verfasst wurde. Der String, der an die Spracherkennung übergeben wird, wird fort an als Dateiinhalt bezeichnet. Wird der Inhalt als englisch identifiziert, wird er an das zum Host-Ordner gehörende Dokument angehängt. Zur Erkennung der Sprache wird die Language-Detec-

1 http://jsoup.org/

tion-Bibliothek[2] von Google in Version vom 03.03.2014 benutzt. Die auch jetzt noch aktuell ist. Die Download-Version enthielt bereits Profile für 53 verschiedene, menschliche Sprachen. Die 53 Sprachprofile wurden um zwei extra angefertigte Sprachprofile für Nicht-Textdateien ergänzt. Sie dienen dazu die Dateien zu verwerfen, die anhand ihrer Dateierweiterung fälschlicherweise als Textdatei identifiziert wurden.

3.4.2 Definition Textdatei, Nicht-Textdatei

Die Definition einer Textdatei weicht, im Rahmen dieser Arbeit, vom üblichen Verständnis ab. Sie ist angelehnt an den im RFC 2046 [18] definierten MIME Media-Type „text/plain“ und wird folglich auch als plain Textdatei bezeichnet. Der Media-Type „text/plain“ bezeichnet, nach Definition, Dateien, die ausschließlich aus schlichtem Text (plain text) bestehen und somit keine Formatierungsbefehle oder Anweisungen jeglicher Art beinhalten. Des Weiteren ist, abgesehen von der entsprechenden Zeichenkodierung, keine spezielle Software nötig, um den Text der Datei zu lesen. Nach dieser Definition besteht der Text einer Datei ausschließlich aus menschlicher Sprache und es gehören weder HTML-Dateien, noch PDFs oder Microsoft-Word-Dateien zu diesem Media-Type. [18, p. 4]

Die im Rahmen dieser Arbeit verwendete Definition von Textdateien, weicht etwas vom Media-Type „text/plain“ ab. Es wird nicht der ganze Text einer Datei verwendet, um sie als Textdatei zu erkennen. Die Dateien werden nur anhand des Dateiinhaltes bewertet, so wie er in Kapitel 3.4.1 definiert wurde. Da die Dateiinhalte demnach keine HTML-Tags mehr enthalten, zählen hier auch HTML-Dateien zu den Textdateien. Des Weiteren werden Dateiinhalte per Definition immer UTF-8 dekodiert eingelesen. Dadurch sind die Dateiinhalte anders kodierter „text/plain“-Dateien in der Regel nicht mehr lesbar und zählen hier somit nicht zu den Textdateien.

Alle Dateien, die keine Textdateien sind, werden hier als Nicht-Textdateien bezeichnet. Nicht-Textdateien werden unterteilt in Binärdateien und technische Dateien.

Als Binärdateien werden hier Dateien bezeichnet, deren Inhalt eine entstellte Folge von Zeichen, wie z. B. „¦¤¿‡NÒ~žå©:þ#!ûÀf¤“, darstellt. Zu ihnen gehören alle Dateien, die nicht dem MIME Media-Type „text“ entsprechen. Dazu gehören u. a. Archiv-, Audio-, Bild-, und Videodateien, aber auch Office-Dateien, wie PDF, DOC, XLS, etc. Des Weiteren werden auch die nicht UTF-8 kodierten „text/plain“-Dateien zu den Bi-

2 http://code.google.com/p/language-detection/

närdateien gezählt, da ihr Dateiinhalt im Java-Programm ebenfalls eine entstellte Zeichenfolge liefert.

Als technische Dateien werden hier Dateien bezeichnet, deren Dateiinhalt Formatierungsbefehle oder Anweisungen jeglicher Art beinhalten. Es sind demnach die Dateien deren Dateiinhalte dem MIME Media-Type „text“ entsprechen, aber nicht dem Untertyp „plain“ angehören. Sie bestehen entweder aus Computersprache oder einer anderen Art von technischen Anweisungen für ein Programm. Zu letzteren gehören beispielsweise die „robots.txt“-Dateien. Als Computersprache werden hier Programmiersprachen, Skriptsprachen, Beschreibungssprachen, usw. verstanden. Dazu zählen u. a. die Sprachen CSS, JavaScript und PHP. Wie bereits erwähnt zählen HTML-Dateien hier nicht dazu, da die Computersprache HTML durch die Entfernung der HTML-Tags theoretisch in keinem Dateiinhalt vorkommt.

3.4.3 Entstehung des Programms

Das Grundgerüst des Programms bildete zu Beginn die Traversierung. Mit ihrer Hilfe wurden die Dateien einer heruntergeladenen Website zu einer einzelnen Datei zusammengefasst. Die Dateien wurde von Beginn an als UTF-8 dekodierter String eingelesen. Der nächste Schritt zum Extrahieren der Dateiinhalte, war das Einbinden des Java HTML-Parsers jsoup. Mit seiner Hilfe wird der eingelesene String einer Datei als HTML-Dokument geparst und ihr Text extrahiert. Da nicht bekannt ist, ob der eingelesene Dateiinhalt einem HTML-Dokument entstammt, wird diese Methode auf alle Dateiinhalte angewendet. Dies ist kein Problem, da diese ggf. unverändert bleiben. Zu diesem Zeitpunkt wurden noch keine XML-Tags und URLs entfernt. Als Nächstes wurde die Language-Detection von Google integriert, um fremdsprachige bzw. nichtenglische Dateiinhalte zu verwerfen.

Googles Language-Detection kommt mit 53 verschiedenen Sprachprofilen und liefert, nach eigenen Angaben, eine Genauigkeit von über 99%. [19] Die Klassifikation basiert auf Naive Bayes und verwendet n-Gramme bis zu einer Länge von drei. Die Language-Detection berechnet die Wahrscheinlichkeit für jede Sprach bzw. jedes der Sprachprofile und gibt die wahrscheinlichste zurück. Die mitgelieferten Sprachprofile wurden aus dem Wikipedia-Korpus der jeweiligen Sprache generiert. Noch offene Probleme sind u. a. mehrsprachige Texte und Quellcode innerhalb eines Textes. [12]

Diese erste Version des Java-Programms zeigte durch die generierten Dokumente auf, dass die Language-Detection in ihrer ursprünglichen Form nicht ausreichte, um alle nichtenglischen Dateiinhalte zu verwerfen. Unter den heruntergeladenen Dateien befanden sich nämlich einige Nicht-Textdateien, die fälschlicherweise als englisch identifiziert wurden. Dies konnte passieren, da die Spracherkennung nur für menschliche Sprachen ausgelegt ist und stets eine Sprache zurückgibt.

Um einen Überblick über die Vielfalt der heruntergeladenen Datei zu bekommen, wurden daraufhin alle positiven Dateien des Level-2-Downloads auf ihre Dateierweiterungen untersucht. Es handelte sich nur um die positiven Dateien, da das Herunterladen der negativen URLs zu diesem Zeitpunkt noch nicht beendet war. Auch die unterhalb folgenden Tests basierten alle nur auf den positiven Dateien. Dieses Vorgehen wird nicht als kritisch betrachtet, da die positiven Dokumente alle das gleiche Thema behandeln – Intellectual-Property. Die negativen Dokumente behandeln hingegen potentiell jedes andere Thema. Aus diesem Grund, ist es vor allem wichtig, saubere positive Dokumente zu generieren. Die positiven Dokumente sollten demnach keine fremdsprachigen Dateiinhalte oder die Inhalte von Nicht-Textdateien enthalten, damit sie bei der Klassifikation eindeutig als positiv erkannt werden. Bei den negativen Dokumenten wurde es hingegen als weniger wichtig erachtet. Des Weiteren wurde angenommen, dass keine großen Unterschiede zwischen den Dateien der positiven und negativen Downloads zu erwarten sind. Die durchgeführte Untersuchung, der positiven Dateien des Level-2-Downloads, ergab drei Listen von Dateierweiterungen. Sie sind in Tabelle 2 dargestellt. Die Dateierweiterungen wurden, anhand des für sie typischen Dateiinhaltes, der entsprechenden Liste zugewiesen. Die Listen entsprechen den im Kapitel 3.4.2 definierten Typen von Text- und Nicht-Textdateien.

Art des Dateiinhaltes		**Dateierweiterungen**
Nicht-Textdatei:	Technische Datei:	ashx, aspx, axd, cgi, css, js, php, rdf, rss, txt, vcf, xml
	Binärdatei:	7zip, gzip, rar, zip, mp3, wav, ogg, cfm, pdf, doc, docx, ppt, pptx, xls, xlsx, eot, ttf, woff, bmp, gif, ico, jpeg, jpg, png, svg, svgz, swf, flv, mp4, wmv
Textdatei:		html, htm, xhtml

Tabelle 2: Übersicht der Dateierweiterungen des positiven Level-2-Downloads.

Die Inhalte von Dateien mit den Erweiterungen „rdf“, „rss“ und „xml“ basierten in der Regel auf XML. Sie enthalten daher XML-Tags. Da jsoup ein HTML-Parser ist, wurden diese Erweiterungen als technische Dateien eingestuft, da die XML-Tags nicht aus den Dateiinhalten entfernt wurden. Wie aufwendig und lohnenswert eine zusätzliche Entfernung der XML-Tags ist, zeigte sich erst in Kapitel 3.4.4.1.

Für das Erzeugen der Dokumente sollten nur englische Textdateien verwendet werden. Da Nicht-Textdateien jedoch fälschlicherweise als englisch klassifiziert wurden, war es nötig für einen zusätzlichen Mechanismus zu sorgen, der dies verhindert. Für die Realisierung wurden die zwei, im Folgenden vorgestellten, Vorgehensweisen in Betracht gezogen.

3.4.3.1 Exklusion anhand der Dateierweiterung

Dateien könnten anhand ihrer Dateierweiterung von der Spracherkennung ausgeschlossen werden. Dieses Blacklist-Verhalten würde für die hier heruntergeladenen Dateien funktionieren, da eine vollständige Liste aller Dateierweiterungen erstellt werden kann. Der Mechanismus zum Erstellen der Dokumente soll jedoch später auch im Klassifikationsprogramm produktive Verwendung finden. Dies bringt das Auftreten neuer, unbekannter Dateierweiterungen mit sich. Eine für den Produktiveinsatz vollständige Liste von Dateierweiterungen zu erstellen, ist aufgrund der stetig wachsenden Masse an weltweit existierenden Dateitypen mit einem kontinuierlichen und unwirtschaftlich hohem Aufwand verbunden. Dieser Anspruch auf Vollständigkeit wäre auch, falls überhaupt, nur für eine kurze Zeit erfüllt. Das Problem der Vollständigkeit einer List würde an Bedeutung verlieren, wenn statt der Blacklist eine Whitelist verwendet werden würde, da die Liste der Textdateien sehr kurz wäre.

Es existiert jedoch ein weiteres Problem, dass sowohl für das Blacklist- als auch für das Whitelist-Verhalten gilt. Eine Dateierweiterung stellt keinen sicheren Indikator für einen Dateityp dar. Manche der heruntergeladenen Dateien hatten keine Dateiendung oder eine verfälschte Dateiendung, wie z. B. „HI_Flyer_engl.pdf-2011-04-20“. Andere hatten wiederum eine zweite Dateiendung, die den eigentlichen Dateityp verdeckt, z. B. „richtext.js.php“ oder „andrew-baker_150x150_Square.jpg.width.130.ashx“. Die heruntergeladenen Dateien zeigten, dass dieses Problem bei nur wenigen Hosts auftrat. Wenn es auftrat, geschah dies jedoch für die Mehrheit der Dateien des Hosts. Es ist demnach wenig von Bedeutung im Bezug auf alle Hosts, aber signifikant für einen einzelnen.

Unbekannte Dateierweiterungen würden mit einer Whitelist ausgeschlossen werden. Dies ist problematisch, wenn es sich um eine englische Textdatei handelt, deren Erweiterung fehlt oder verfälscht ist. Mit einer Blacklist würden unbekannte Dateierweiterungen zugelassen werden. Dies ist problematisch, wenn es sich um eine Nicht-Textdatei handelt, deren Erweiterung fehlt oder verfälscht ist.

Eine Exklusion anhand der Dateierweiterung ist unsicher, vor allem wenn die Menge der vorkommenden Dateierweiterungen uneingeschränkt ist. Ob sich eine derartige Exklusion, als zusätzliche Vorgehensweise, nützlich für das Programm erweist, wird in Kapitel 3.4.4 geklärt.

3.4.3.2 Exklusion anhand des Dateiinhalts

Eine weitere Vorgehensweise zum Identifizieren von Nicht-Textdateien stellt das Analysieren des Dateiinhaltes dar. Eine Betrachtung der Dateiinhalte von Nicht-Textdateien zeigte, dass sie entweder aus einer Computersprache oder einer entstellten Folge von Zeichen bestand. Aus diesen Beobachtungen resultierten die bereits erwähnten Definitionen von Text- und Nicht-Textdateien sowie technischen Dateien und Binärdateien.

Die Google Language-Detection bietet die Möglichkeit neue Sprachprofile aus Texten zu generieren. Die Java-Bibliothek muss dazu wie folgt aufgerufen werden:

```
$ java -jar lib/langdetect.jar --genprofile-text -l [language
code] [text file path]
```

An der Stelle `[language code]` steht dabei der gewünschte Name des zu generierendenl Sprachprofils. An der Stelle `[text file path]` wird ein Pfad erwartet. Der Text der Datei, auf die der Pfad zeigt, bildet die Grundlage der Generierung.

Um ein Sprachprofil zu generieren muss demnach eine Datei erstellt werden, welche die Sprache repräsentiert. Wie im Folgenden beschrieben, wurden der Reihe nach zwei Sprachprofile erstellt und getestet.

3.4.3.3 Das Sprachprofil „none“

Erstellen des Sprachprofils

Die Inhalte der Binärdateien enthielten oft sehr viele „�“ und keine Buchstaben des englischen Alphabets. Diese Besonderheit führte zu der Idee, die Spracherkennung um ein neues Sprachprofil zu erweitern. Dieses sollte Binärdateien repräsentieren und wurde „none“ genannt. Der Name kommt daher, dass die zu identifizierende Sprache eigentlich keine (engl. none) Sprache ist.

Die Datei zum Generieren des Sprachprofils „none“ sollte entsprechend aus allen Dateiinhalten der Binärdateien bestehen. Als Grundlage für die Inhalte dienten die positiven Dateien des Level-2-Downloads.

Das Zusammentragen der nötigen Dateiinhalte sollte anhand der Dateierweiterungen geschehen. Dazu wurde das bis zu diesem Zeitpunkt entwickelte Java-Programm vorübergehend abgewandelt. Die bereits integrierte Spracherkennung wurde deaktiviert und es wurde eine Erkennung für Dateierweiterungen entwickelt und hinzugefügt. Die Dateierweiterungs-Erkennung basiert auf den drei weiter oben eingeführten Listen und erfolgt in eins bis zwei Schritten. Im ersten Schritt wird die Teilzeichenfolge nach dem letzten Punkt bis zum Ende des Dateinamens als Dateierweiterung angesehen. Anschließend wird geprüft, ob sich diese Dateierweiterung in einer der Listen befindet. Ist dem nicht so, wird der zweite Schritt vollzogen. In ihm entfernt die Erkennung den Query-Part und den Fragment-Part des Dateinamen. Diese ursprünglichen URL-Bestandteile wurden beim Download ggf. von Wget in den Dateinamen übernommen. Um den Fragment-Part zu entfernen, werden alle Zeichen ab und inklusive des ersten Hash-Zeichens entfernt. Um den Query-Part zu entfernen, werden alle Zeichen ab und inklusive des ersten Fragezeichens entfernt. Die übrige Zeichenkette wird, wie auch schon im ersten Schritt, am letzten Punkt aufgespalten. Es wird erneut geprüft, ob sich die erkannte Dateierweiterung in einer der Listen befindet. Bei den Überprüfungen wird nicht auf Groß- und Kleinschreibung der Dateierweiterungen geachtet.

Für das Generieren des Sprachprofils „none“ sollten, wie bereits erwähnt, die Dateiinhalte aller Binärdateien verwendet werden. Dementsprechend wurden die Dateiinhalte aller Dateien zusammengefasst, die durch die Dateierweiterungs-Erkennung der Liste der Binärdateien zugeordnet wurden. Die Inhalte der gefundenen Dateien wurden noch einmal händische überprüft. Es mussten dadurch keine Dateiinhalte verworfen werden.

Die Liste der technischen Dateien wurde für die Generierung absichtlich weggelassen, da diese Dateien viele Buchstaben des englischen Alphabets enthalten. Es wurde davon ausgegangen, dass dies dazu beigetragen hätte, dass englische Dateiinhalte vermehrt als „none“ klassifiziert worden wären.

Testen des Sprachprofils

Nachdem das Sprachprofile generiert wurde, wurde es getestet. Der Test sollte zeigen, dass das Sprachprofil Binärdateien auch als solche erkennt. Dabei wurde aber vor allem

Wert darauf gelegt, dass dem neuen Sprachprofil nicht fälschlicherweise englische Dateiinhalte zugeordnet werden. Für den Test wurde die Spracherkennung im Java-Programm wieder aktiviert und das Sprachprofil entsprechend hinzugefügt. Die Dateierweiterungs-Erkennung wurde dagegen wieder deaktiviert. Das Programm wurde durchlaufen. Dabei wurden aber keine Dokumente aus englischen Dateiinhalten gebildet, sondern aus den als „none“ erkannten. Die gebildeten Dokumente basierten auf den positiven Dateien des Level-1-Downloads. Des Weiteren wurde nicht für jeden Host-Ordner ein Dokument gebildet, sondern für einzelne Dateiinhalte bzw. Dateien. Der Programmdurchlauf klassifizierte 678 Dateiinhalte als „none“. Diese wurden noch einmal händisch überprüft und klassifiziert, um zu sehen, ob sie zurecht als „none“ erkannt wurden. Das Ergebnis ist in Tabelle 3 dargestellt.

Arten des Dateiinhaltes		**Absolute Häufigkeit**	**Relative Häufigkeit**
Nicht-Textdateien:	Binärdateien:	670	ca. 98,82 %
Textdateien:	Fremdsprachig:	2	ca. 0,29 %
Hybride:		6	ca. 0,88 %

Tabelle 3: Ergebnis der manuellen Klassifikation der Dateiinhalte des positiven Level-1-Downloads. Es wurden die Dateien klassifiziert, die von der Spracherkennung als „none“ erkannt wurden. Die Spracherkennung verwendete dabei das Sprachprofil „none“ und die Standard-Profile.

Wie die Tabelle Tabelle 3 zeigt, hat das Sprachprofil „none“ mit ca. 98,82 % einen sehr guten Recall. Es stellt somit eine Verbesserung des Programms dar. Dazu kommt, dass weder die fremdsprachigen noch die hybriden Dateien von Bedeutung sind. Dies liegt daran, dass die Dokumente für die Klassifikation nur auf Dateiinhalten von englischen Textdateien beruhen sollten. Im Hinblick darauf besitzt das Sprachprofil sogar einen Recall von 100 %. Im Vergleich dazu sei erwähnt, dass die Dateierweiterungs-Erkennung nur 378 Nicht-Textdateien als solche erkannte. Sie lieferte somit eine Recall von ca. 55,75 %.

Der Recall des „none“-Sprachprofils wurde aufgrund des hohen Aufwands nicht ermittelt. Statt dessen wurde eine stichprobenartige Prüfung der als englisch erkannten Dateiinhalte vollzogen, um zu testen, ob sich unter ihnen noch Nicht-Textdateien befanden. Dabei wurden technische Dateien entdeckt.

3.4.3.4 Das Sprachprofil „java_script“

Erstellen des Sprachprofils

Nach dem Test des Sprachprofils „none“ zeigte eine stichprobenartige Untersuchung der als englisch erkannten Dateiinhalte, dass sich unter ihnen noch technische Dateien befanden.

Die Inhalte technischer Dateien enthielten in der Regel Funktionswörter, die an das Englische angelehnt waren. Im Vergleich zur englischen Sprache enthielten die Inhalte jedoch üblicherweise auch signifikant mehr Klammerzeichen, wie „(“, “)“, „{“, „}“, „[“, „]“, Semikolons und Gleichheitszeichen. Diese Besonderheit führte auch hier zu der Idee, die Spracherkennung um ein neues Sprachprofil zu erweitern. Dieses sollte technische Dateien repräsentieren und wurde „java_script“ genannt, weil es aus JavaScript-Dateien generiert wurde. Die JavaScript-Dateien wurden stellvertretend benutzt, da JavaScript die hier am häufigsten vorkommende Computersprache war und sie die erwähnten, signifikant häufigeren Sonderzeichen enthält. Das Zusammentragen der Dateiinhalte wurde wieder mit der abgewandelten Version des entwickelten Java-Programms durchgeführt. Für das Sprachprofil „java_script“ wurden alle Dateiinhalte zusammengetragen, deren Dateierweiterung als „.js“ erkannt wurde. Als Grundlage dienten erneut die positiven Dateien des Level-2-Downloads. Die Dateierweiterungs-Erkennung fand nur elf JavaScript-Dateien. Eine händische Prüfung der Java-Script-Dateiinhalte reduzierte die Anzahl auf zehn. Es handelte sich dabei jedoch um relativ lange Dateiinhalte. Sehr viel später stellte sich heraus, dass nur so wenige Dateien gefunden wurden, weil die meisten Dateien mit JavaScript-Inhalten nicht die Dateierweiterung „js“ besaßen, sondern z.B. „axd“.

Testen des Sprachprofils

Nachdem das Sprachprofile generiert wurde, wurde es ebenfalls getestet. Der Test sollte zeigen, dass das Sprachprofil technische Dateien auch als solche erkennt. Wieder wurde dabei vor allem Wert darauf gelegt, dass englische Dateiinhalte nicht fälschlicherweise dem neuen Sprachprofil zugeordnet werden. Für den Test wurde das Java-Programm nochmals entsprechend abgewandelt und das Sprachprofil „java_script“ zur Spracherkennung hinzugefügt. Das Sprachprofil „none“ blieb weiterhin in der Spracherkennung. Der Durchlauf des Programmes bildete für jeden Dateiinhalt ein Dokument, wenn er als „java_script“ erkannt wurde. Die gebildeten Dokumente basierten auch hier auf den po-

sitiven Dateien des Level-1-Downloads. Der Programmdurchlauf klassifizierte 462 Dateiinhalte als „java_script“. Diese wurden nocheinmal händisch überprüft und klassifiziert, um zu sehen, ob sie zurecht als „java_script“ erkannt wurden. Das Ergebnis ist in Tabelle Tabelle 4 dargestellt.

Arten des Dateiinhaltes				**Absolute Häufigkeit**	**Relative Häufigkeit**
Nicht-Textdateien:	Binärdateien:			4	ca. 0,87 %
	Technische Dateien:	JavaScript-Dateien:		145	ca. 31,39 %
		„robots.txt“-Dateien:		62	ca. 13,42 %
		XML-Dateien:		11	ca. 2,38 %
		Andere Dateien:		8	ca. 1,73 %
Textdateien:	≤ 250 Zeichen:			48	ca. 10,39 %
	> 250 Zeichen:	Fremdsprachig:		26	ca. 5,63 %
		Mehrsprachig mit englisch:	≤ 50 % englisch:	119	ca. 25,76 %
			> 50 % englisch	39	ca. 8,44 %

Tabelle 4: Ergebnis der manuellen Klassifikation der Dateiinhalte des positiven Level-1-Downloads. Es wurden die Dateien klassifiziert, die von der Spracherkennung als „java_script“ erkannt wurden. Die Spracherkennung verwendete dabei die Sprachprofile „java_script“ und „none“ sowie die Standard-Profile.

Wie aus Tabelle 4 ersichtlich, erzielte das Sprachprofil „java_script“ mit ca. 31,39 % für JavaScript-Dateien bzw. ca. 48,92 % für technischen Dateien einen sehr schlechten Recall. Die Untersuchungen zeigten jedoch, dass die Mehrheit der restlichen Dateien für das Generieren der positiven Dokumente auch nicht nützlich gewesen wäre. Ein Dateiinhalt wird hier als nützlich verstanden, wenn er englisch ist und Wörter enthält, die relevant für seine Klasse sind. Ein positiver Dateiinhalt der englisch ist und das Thema Intellectual-Property behandelt ist demnach nützlich. Ein nützlicher Dateiinhalt ist von Bedeutung, um ein Dokument richtig zu klassifizieren. Ein Dateiinhalte, der nicht nützlich ist, erhöht dagegen die Wahrscheinlichkeit, dass ein Dokument der falschen Klasse zugeordnet wird.

Trotz schlechtem Recall ist das Sprachprofil eine Bereicherung des Programms, da es hauptsächlich Dateien filtert, die nicht nützlich sind. Die kurzen Textdateien, mit einem Inhalt von höchstens 250 Zeichen, waren aufgrund ihres Inhaltes und dessen Länge nicht

von Bedeutung. Einige bestanden zwar ausschließlich aus englischem Text, dessen Inhalt war jedoch nicht nützlich. Kurze Dateien sind in der Regel auch nur ein kleiner Teil eines Dokumentes. Es wird daher davon ausgegangen, dass die sie einen vernachlässigbar kleinen Einfluss auf die Wahrscheinlichkeit haben, mit der ein Dokumente der richtigen Klasse zugeordnet wird. Bei dem Inhalt der kurzen Textdateien handelte es sich in der Regel um den Firmennamen oder um eine technische Benachrichtigung. Einige der Inhalte waren „Contact Support“, „Object moved Object moved to here.“, „Untitled Document“, „Login Check SSO Connection...“, „Clients <body> </body>“, „index“, „index_pais.htm Eesti“ und „FEEL 100%□ □ □ □ PART 1“. Die kurzen Textdateien wurden vor der Einführung des Sprachprofils „java_script“ alle als englisch klassifiziert. So wie die kurzen, waren auch die langen Textdateien nicht von Bedeutung. Ihr Inhalt war nicht ausschließlich englisch. Allerdings hatten 23 Dateien einen so hohen englisch Anteil, dass sie vielleicht doch nützlich gewesen wären. Sie bilden insgesamt einen Anteil von ca. 4,98 %. Von allen 462 Dateien wurden vor der Einführung des Sprachprofils „java_script“ insgesamt 178 Dateien als englisch klassifiziert. Im Hinblick auf diese Menge, bildeten die 23 Textdateien einen Anteil von ca. 12,92 %. Ca. 78,09 % bzw. 139 Dateien der ehemals als englisch klassifizierten Dateien waren dagegen nicht nützlich. Bei den restlichen 16 Dateien bzw. ca. 8,99 % der Dateien wurde davon ausgegangen, dass sie aufgrund ihres Inhaltes und dessen Kürze nicht nützlich gewesen wären. Es wurde aber auch davon ausgegangen, dass sie die Wahrscheinlichkeit zur falschen Klassifikation nicht bedeutend beeinflusst hätten. Das Sprachprofil ist, wie bereits erwähnt, eine Bereicherung für das Programm, im Hinblick auf die Filterung der Dateien. Mit einem Anteil von ca. 12,92 % an wahrscheinlich fälschlicherweise verworfenen Dateien, ist es jedoch keine optimale Lösung.

Dafür, dass Textdateien fälschlicherweise als „java_script“ erkannt wurden, gab es mehrere Ursachen. Eine davon war, dass mehrsprachige Texte von der Google Language Detection nicht unterstützt werden. Des Weiteren funktioniert sie nach eigenen Angaben nicht so gut für kurze Texte. [12] Eine weitere Ursache zeigte sich bei einer genaueren Betrachtung des Sprachprofils. Dieses ist im JSON-Format[1] gespeichert. Das Sprachprofil enthält u. a. eine Liste von Zeichen mit den dazugehörigen absoluten Häufigkeiten. Diese Werte entstammen der Textdatei, die beim Generieren des Sprachpro-

1 http://json.org/

fils übergeben wurde. Mit Hilfe der Interactive Ruby Shell wurde dem Sprachprofil „java_script“ eine Liste der zehn häufigsten Zeichen entnommen. Das Gleiche wurde mit dem Sprachprofil „none“ und dem englischen „en“ getan. Daraus ergaben sich die in Tabelle 5 dargestellten Listen.

Sprachprofil „en“		**Sprachprofil „java_script“**		**Sprachprofil „none“**	
Absolute Häufigkeit	**UTF8-Zeichen**	**Absolute Häufigkeit**	**UTF8-Zeichen**	**Absolute Häufigkeit**	**UTF8-Zeichen (ggf. mit Zeichenname u. Hex-Code)**
27.848.709	„e“	64.617	„t“	211.649.205	„“ REPLACEMENT CHARACTER (0xEF 0xBF 0xBD)
23.418.136	„a“	62.391	„e“	322.495	„“ [Unassigned Code Point] (0xDF 0xBD)
20.570.816	„i“	49.310	„n“	310.893	„ “ <Hangul Syllable, First>
19.713.516	„n“	46.092	„i“	17.908	„۶“ EXTENDED ARABIC-INDIC DIGIT SIX
19.408.712	„t“	37.868	„r“	14.146	„ş“ LATIN CAPITAL LETTER S WITH CEDILLA
18.673.929	„o“	35.675	„o“	13.779	„ٹ“ ARABIC LETTER THEH
16.873.084	„r“	32.814	„s“	13.683	„ţ“
16.004.058	„s“	30.996	„a“	13.515	„ٸ“ ARABIC LETTER YEH
10.669.664	„l“	24.797	„l“	11.121	„“ [Unassigned Code Point] (0xDF 0xBF)
10.134.148	„h“	24.529	„u“	9.916	„ϧ“ NKO LETTER LA

Tabelle 5: Übersicht der zehn häufigsten Zeichen aus den Sprachprofilen „java_script“, „none“ und „en“ (englisch). Die Hex-Codes entstammen [20] und die Zeichenname [21].

Wie in der Liste des Sprachprofils „java_script“ in Tabelle 5 zu sehen ist, fehlen die erwähnten Sonderzeichen, die signifikant häufiger vorkommen als im Englischen. Dazu gehören Klammerzeichen, wie „(“, „)“, „{“, „}“, „[“ und „]“, Semikolons und Gleichheitszeichen. Diese Zeichen sind alle nicht im Sprachprofil enthalten. Es wurde herausgefunden, dass beim Generieren des Sprachprofils nur Buchstaben gezählt werden. Die Erkennung basiert daher auf den verwendeten Variablennamen, Strings und Funktionsnamen und nicht auf den für viele Computersprachen typischen Sonderzeichen der Syntax. Ohne deren Entfernung wären der Anteil der plain Textdateien im Test wahrscheinlich deutlich unter den 55,22 %. Andererseits würden dann nicht die unnötigen, kurzen Textdateien erkannt werden. Es wird aber angenommen, dass diese, aufgrund ihrer Länge, keinen nennenswerten Einfluss auf ein gesamtes Dokument hätten. Auf eine Modifizierung der Language-Detection-Bibliothek wurde verzichtet, da es später das aktuell halten des Klassifikationsprogramms erschwert. Die Anpassungen müssten in jeder neuen Version der Bibliothek wiederholt vorgenommen werden. Die Veränderung von Bibliotheken ist vor allem in einem produktiven Umfeld, wie es am Fraunhofer MOEZ zu finden ist, keine gute Praxis, da viele verschiedene Bibliotheken benutzt werden.

Wie der Test zeigte, ist das Sprachprofil „java_script“ trotz seines schlechten Recalls eine Bereicherung für das Programm. Die Höhe des Recalls wurde auch hier nicht ermittelt, da dies mit einem zu hohen Aufwand verbunden wäre. Alternativ wurde wieder eine stichprobenartige Prüfung, der als englisch erkannten Dateiinhalte, vollzogen, um zu testen, ob sich unter ihnen immer noch Nicht-Textdateien befinden.

3.4.4 Testen des Programms

Nachdem die beiden Sprachprofile „none“ und „java_script“ eingeführt und ihre Recall-Werte ermittelt wurden, wurde ein abschließender Test durchgeführt. Er sollte Prüfen, ob sich immer noch Nicht-Textdateien unter den als englisch erkannten Dateiinhalten befanden. Gegebenenfalls sollten letzte Verbesserungen vorgenommen werden. Die Dateierweiterungs-Erkennung wurde zeitgleich mit getestet, um festzustellen ob sie eine eventuell nötige Verbesserung des Programms darstellt.

Das vorhandene Programm und die Dateierweiterungs-Erkennung wurden zunächst modifiziert. Dokumente wurden nun wieder aus englischen Dateiinhalten gebildet. Es blieb jedoch dabei, dass ein Dokument aus nur einem einzelnen Dateiinhalt bestand und nicht, wie im eigentlichen Sinne, aus allen Dateiinhalten, die zur selben URL gehören.

Neben der Spracherkennung wurde auch die Dateierweiterungs-Erkennung angewendet. Im Produktiveinsatz würde sie dazu dienen Nicht-Textdateien zu verwerfen. Im folgenden Test wurde sie lediglich dazu genutzt Textdateien und Nicht-Textdateien zu markieren. Die ursprünglichen drei Listen der Dateierweiterungen wurden entsprechend auf zwei reduziert. Die Liste der technischen Dateien und die Liste der Binärdateien wurden dazu zusammengefasst. Da Dateien ohne Dateierweiterung existieren, decken die beiden Listen im Produktiveinsatz nicht mehr alle vorhandenen Dateierweiterungen ab. Die Gründe für Letzteres wurden bereits in Kapitel 3.4.3.1 genannt. Alle Dateien, die keine Erweiterung aus den beiden Listen besitzen, befinden sich demnach in einer Grauzone. Um die Dateien der Grauzone klassifizieren zu können wurde der Dateierweiterungs-Erkennung ein Blacklist- und Whitelist-Modus hinzugefügt. Läuft die Dateierweiterungs-Erkennung im Blacklist-Modus werden die Dateien der Grauzone als Textdatei klassifiziert. Im Whitelist-Modus werden dagegen alle Dateien der Grauzone als Nicht-Textdatei klassifiziert.

Für den folgenden Test wurde die Dateierweiterungs-Erkennung im Blacklist-Modus ausgeführt. Das Programm traversierte erneut die positiven Dateien des Level-1-Downloads. Der Language-Detection standen die Standard-Sprachprofile sowie die Profile „none“ und „java_script“ zur Verfügung. Der Programmdurchlauf erkannte 32.311 Dateiinhalte als englisch. Von ihnen wurden ca. 3,57 % bzw. 1.152 Dateien von der Dateierweiterungs-Erkennung als Nicht-Textdatei klassifiziert. Ca. 96,43 % bzw. 31.159 Dateien wurden als Textdatei klassifiziert. Davon gehörten 55 Dateien zur Grauzone und 31.104 Dateien hatten eine Erweiterung aus der Liste für Textdateien. Letztere werden im folgenden als Menge der echten Textdateien bezeichnet.

Die Grauzone war mit 55 Dateien bzw. ca. 0,17 % aller als englisch klassifizierten Dateien sehr klein. Es ist jedoch unklar, wie groß dieser Anteil im Produktivbetrieb sein wird. Dies liegt daran, dass die Erstellung der Listen für Dateierweiterungs-Erkennung auf den selben Dateien basierte, wie der Testdurchlauf. Es konnten demnach keine unbekannten Dateierweiterungen auftreten. Die Grauzone enthielt hier demnach nur Dateien, die gar keine Erweiterung besaßen oder keine echte, wie z. B. „.pdf-2011-04-20“. Auch ist unklar, wie groß der Anteil der Dateien ohne Erweiterungen sein würde, würde beim Herunterladen der Websites auf die Wget-Option „--adjust-extension“ verzichtet werden, welche fehlende HTML- und CSS-Erweiterungen ergänzt. [16]

3.4.4.1 Testen der Spracherkennung:

Die drei Mengen der Nicht-Textdateien, der Grauzone und der echten Textdateien wurden händisch untersucht und ihre Dateien anhand ihrer Inhalte klassifiziert. Für die beiden Mengen der Nicht-Textdateien und der Grauzone wurde jede Datei untersucht. Für die Menge der echten Textdateien wurde eine Stichprobe von 1.000 Dateien angefertigt und untersucht. Die Ergebnisse der Stichprobe wurden auf die Gesamtmenge hochgerechnet. Die 1.000 Dateien der Stichprobe entstammten den Websites 420 unterschiedlicher URLs. Ein zusammengefasstes Ergebnis aller drei Untersuchungen ist in Tabelle 6 dargestellt.

Arten des Dateiinhaltes			**Absolute Häufigkeit**	**Relative Häufigkeit**
Nicht-Textdateien:	Binärdateien:		1,0	ca. 0,0031 %
	Technische Dateien:		1079,1	ca. 3,3397 %
Textdateien:	Englisch:	Mit Tags:	385,0	ca. 1,1917 %
		Ohne Tags:	30.779,6	ca. 95,2606 %
	Mehrsprachig mit englisch:	Ohne Tags:	62,2	ca. 0,1925 %
	Fremdsprachig:		4,0	ca. 0,0124 %

Tabelle 6: Ergebnis der manuellen Klassifikation der Dateiinhalte des positiven Level-1-Downloads. Es wurden die Dateien klassifiziert, die von der Spracherkennung als „en“ (englisch) erkannt wurden. Die Spracherkennung verwendete dabei die Sprachprofile „java_script“ und „none“ sowie die Standard-Profile. Für ca. 96,26 % der Dateien wurden nur eine Stichprobe von 1.000 klassifiziert und hochgerechnet.

Wie aus Tabelle 6 ersichtlich, weicht das Ergebnis des Testdurchlaufs mit ca. 95,26 % um ca. 4,74 % vom Idealfall ab. Es wurden fremdsprachige, mehrsprachige und Nicht-Textdateien fälschlicherweise als englisch klassifiziert. Des Weiteren enthielten noch ca. 1,2 % der Dateiinhalte Tags, obwohl diese durch die Java-Bibliothek jsoup hätten entfernt werden sollen.

Problem: Nicht-Textdateien, technische Dateien

Bei ca. 3,35 % der im Testdurchlauf als englisch erkannten Dateien handelte es sich um Nicht-Textdateien. Unter diesen befand sich eine einzige Binärdatei. Sie wurde fälschlicherweise als englisch klassifiziert, da ihr Inhalt nur 93 Zeichen lang war und davon 46 Zeichen englischen, schlichten Text darstellten. Die technischen Dateiinhalte enthielten, wie üblich, signifikant mehr Sonderzeichen als normaler Text. Ihr Sprachgebrauch äh-

nelte jedoch nicht den typischen Funktionswörtern und sie wurden somit als englisch erkannt, da die Sonderzeichen vom Sprachprofil nicht beachtet werden. Für das Beachten der Sonderzeichen lässt sich auf Basis der Google Language-Detection keine Lösung finden, ohne die Bibliothek zu modifizieren. Modifikation ist jedoch, wie bereits in Kapitel 3.4.3.4 erläutert, keine Option. Ob die Dateierweiterungs-Erkennung ein mögliche Lösung des Problems darstellt, wird unterhalb in Kapitel 3.4.4.2 in einer detaillierteren Untersuchung geklärt. Die Dateierweiterungs-Erkennung kann jedoch höchstens eine Lösung für bekannte Dateierweiterungen darstellen. Im Testdurchlauf wurden 55 Dateien von der Dateierweiterungs-Erkennung als Grauzone erkannt. Die händische Klassifizierung ergab, dass 5 Dateien davon Nicht-Textdateien waren. Dies entspricht einem Anteil von ca. 0,02 % an allen, als englisch klassifizierten, Dateien und $9.\overline{09}$ % der Grauzone. Eine Lösung durch Modifizieren der Bibliothek ist, wie erwähnt, ausgeschlossen. Eine bessere Spracherkennungs-Bibliothek, die auch Sonderzeichen beachtet, wurde nicht gefunden. Eine mögliche Lösung könnte eine weitere Klassifikation der bereits als englisch klassifizierten Dateiinhalte sein. Dabei könnte eine passende, zweite Spracherkennungs-Bibliothek oder ein selbst entwickelter Algorithmus zum Einsatz kommen. Die Lösung des Problems wurde, unter Berücksichtigung seines Ausmaßes und des Aufwands einer Lösung, als unwirtschaftlich erachtet.

Problem: mehrsprachige Dateien mit englisch

Der Anteil der mehrsprachigen Dateiinhalte, die auch englisch beinhalten, wurde auf ca. 0,1925 % hochgerechnet. Tatsächlich waren es nur 2 mehrsprachige Dateien in der Stichprobe. Diese wurden von der Google Language-Detection als englisch klassifiziert, weil sie überwiegend aus englischer Sprache bestanden. Aus diesen Gründen wird das Problem nicht als kritisch erachtet. Beim Lösen des Problems würde der Aufwand nicht den Nutzen rechtfertigen.

Problem: Fremdsprachige Dateien

Die fremdsprachigen Dateiinhalte wurde fälschlicherweise als englisch erkannt, weil sie zum Großteil aus englischsprachigen und englisch anmutenden URLs bestanden und teilweise HTML-Tags enthielten. Da nur ca. 0,0124 % der Dateien betroffen waren, wird dieses Problem nicht als kritisch erachtet. Ihm wäre aber durch die Entfernung der URLs und Tags entgegen gewirkt.

Problem: nicht entfernte Tags

Ca. 1,2 % der untersuchten Dateien enthielten noch Tags. Bei den Tags handelte es sich um XML- oder HTML-Tags. Die Java-Bibliothek jsoup konnte demnach nicht alle HTML-Tags entfernen. Die Ursache dafür liegt in der Art und Weise, wie sie es tut. Sie verwendet nicht die einfache und riskante Methode alle Zeichen zwischen einem „<“ und einem „>“ zu entfernen. Stattdessen versucht sie den übergebenen Dateiinhalt als HTML-Dokument zu parsen. Dabei ergänzt sie ggf. fehlende Tags und sorgt für valides HTML. [22] Da das Modifizieren einer Bibliothek, aus den bereits genannten Gründen, nicht in Frage kommt, wurde auf eine genauere Fehleranalyse verzichtet. Das Problem wurde gelöst, in dem das Entfernen der HTML-Tags mittels jsoup solange wiederholt wird, bis die Methode keine Änderungen am Inhalt mehr vornimmt. Stichproben der betroffenen Dateien zeigten, dass eine weitere Wiederholung in der Regel ausreichte, um alle HTML-Tags zu entfernen. Des Weiteren wurde deutlich, dass auf diese Weise auch alle XML-Tags entfernt werden. Dies eröffnete die Möglichkeit die RDF-, RSS- und XML-Dateien in der Dateierweiterungs-Erkennung auf die Liste der Textdateien zu verschieben. Ob ihre Dateiinhalte jedoch die beiden Dokumentklassen positiv und negativ widerspiegeln oder ob sie eher technischer Natur sind und somit doch auf der Liste der technischen Dateien bleiben sollten, zeigt sich erst in Kapitel 3.4.4.2.

Problem: URLs im Text

Wie bereits bei den fremdsprachigen Dateien erwähnt wurde, führte ein großer Anteil an URLs im Dateiinhalt zu einer falschen Klassifikation durch die Spracherkennung. Dies ist vor allem bei sehr kurzen Dateiinhalten problematisch. URLs traten jedoch auch allgemein in vielen Dateiinhalten auf. Eine der fremdsprachigen Textdateien befand sich unter den 55 Dateien der Grauzone. Dies entspricht einem Anteil von $1,\overline{81}$ % an der Grauzone. Der Anteil könnte im Produktivumfeld jedoch größer sein, da die Größe der Grauzone nicht repräsentativ für den Produktiveinsatz ist, wie oberhalb erläutert. Aus diesen Gründen wurde ein Mechanismus in das Programm eingebaut, der URLs aus den Dateiinhalten entfernt, bevor diese an die Spracherkennung weitergegeben werden. Eine URL wird dabei über den regulären Ausdruck „(ht|f)tp(s?)://[^\\s]*“ erkannt. Demnach wird im Text nach Vorkommen von „http“, „https“, „ftp“ oder „ftps“ gefolgt von „://“ gesucht. Darauf folgen wiederum null oder mehr Nicht-Whitespace-Charaktere. Anders

gesagt, die URLs wird als beendet interpretiert, sobald der erste, folgende Whitespace-Charakter vorkommt. Dieser wird nicht zur URL gezählt.

3.4.4.2 Testen der Dateierweiterungs-Erkennung:

Ein bis hierhin ungelöstes Problem waren die übrigen Nicht-Textdateien, die von der Spracherkennung als englisch klassifiziert wurden. Wie bereits erwähnt, wurde die Dateierweiterungs-Erkennung beim Programmdurchlauf benutzt, um den traversierten Dateien eine der drei Klassen zu zuweisen. Daraus entstanden die drei oberhalb vorgestellten Mengen der Nicht-Textdateien, der Grauzone und der echten Textdateien. Die Mengen wurden untersucht, um herauszufinden, ob die Dateierweiterungs-Erkennung das besagte Problem löst. Des Weiteren sollte festgestellt werden, ob im Fall des Einsatzes der Blacklist- oder der Whitelist-Modus zu bevorzugen ist.

Menge der Grauzone

Die Anzahl der Dateien, die von der Spracherkennung als englisch erkannt wurden und von der Dateierweiterungs-Erkennung keiner Liste zugeordnet werden konnten, betrug 55. Die Grauzone hatte demnach einen Anteil von ca. 0,17 % an allen als englisch klassifizierten Dateien. Für die Datenmenge des Testdurchlaufs wäre es demnach nicht von großer Bedeutung, ob die Dateierweiterungs-Erkennung im Blacklist- oder Whitelist-Modus ausgeführt wird. Der Anteil der Grauzone ist jedoch im Rahmen dieses Tests für den Produktiveinsatz nicht vorhersehbar. Die Grauzone enthält hier nur Dateien ohne echte Erweiterung. Die Gründe dafür wurden bereits in der Einleitung von Kapitel 3.4.4 beschrieben. Die Dateien der Grauzone wurden händisch anhand ihrer Inhalte klassifiziert. Die Ergebnisse sind in Tabelle 7 dargestellt.

Arten des Dateiinhaltes				**Absolute Häufigkeit**	**Relative Häufigkeit**
Nicht-Textdateien:	Technische Dateien:	JavaScript-Dateien:		1	1,$\overline{81}$ %
		Andere:		4	7, $\overline{27}$ %
Textdateien:	Englisch:		Mit Tags:	35	63, $\overline{63}$ %
			Ohne Tags:	14	25, $\overline{45}$ %
	Fremdsprachig:		Mit Tags:	1	1, $\overline{81}$ %

Tabelle 7: Ergebnis der manuellen Klassifikation von Dateiinhalten des positiven Level-1-Downloads. Es wurden die Dateien klassifiziert, die von der Dateierweiterungs-Erkennung im Blacklist-Modus als Grauzone erkannt wurden und

von der Spracherkennung als „en“ (englisch) erkannt wurden. Die Spracherkennung verwendete dabei die Sprachprofile „java_script“ und „none“ sowie die Standard-Profile.

Die Resultate aus Tabelle 7 beruhen auf nur 55 Dateien. Es wird daher davon ausgegangen, dass die Prozentsätze nicht als repräsentative Aufteilung für den Produktivbetrieb angesehen werden können. Falls überhaupt, lässt sich aus ihnen nur eine sehr grobe Tendenz erkennen. Die englischen Dateien bilden einen Anteil von $89,\overline{09}$ %. Dazu kommen ca. 1, $\overline{81}$ % fremdsprachige Dateien. Mit diesem Anteil an richtig als Textdatei erkannten Dateiinhalte geht die Tendenz dahin, dass der Blacklist-Modus gegenüber dem Whitelist-Modus zu bevorzugen ist.

Menge der Nicht-Textdateien

Es gab insgesamt 1.152 Dateien, die von der Spracherkennung als englisch erkannt wurden und eine Dateierweiterung aus der Liste für Nicht-Textdateien besaßen. Dies sind ca. 3,75 % aller als englisch klassifizierten Dateien. Diese Dateien würden durch das Anwenden der Dateierweiterungs-Erkennung verworfen werden bzw. von der Dokumentbildung ausgeschlossen werden. Daher wurden sie händisch untersucht und anhand ihres Inhaltes klassifiziert, um herauszufinden, ob die Dateierweiterungs-Erkennung fälschlicherweise zu viele Textdateien verwerfen würde. Die Resultate sind in Tabelle 8 dargestellt.

Arten des Dateiinhaltes			**Absolute Häufigkeit**	**Relative Häufigkeit**
Nicht-Textdateien:	Binärdateien:		1	ca. 0,09 %
	Technische Dateien:		1043	ca. 90,54 %
Textdateien:	Englisch:	Mit Tags:	39	ca. 3,39 %
		Ohne Tags:	66	ca. 5,73 %
	Fremdsprachig:	Mit Tags:	2	ca. 0,17 %
		Ohne Tags:	1	ca. 0,09 %

Tabelle 8: Ergebnis der manuellen Klassifikation von Dateiinhalten des positiven Level-1-Downloads. Es wurden die Dateien klassifiziert, die von der Dateierweiterungs-Erkennung als Nicht-Textdateien erkannt wurden und von der Spracherkennung als „en“ (englisch) erkannt wurden. Die Spracherkennung verwendete dabei die Sprachprofile „java_script“ und „none“ sowie die Standard-Profile.

Wie aus Tabelle 8 ersichtlich, wurden ca. 90,63 % der Dateien aus der Menge richtig als Nicht-Textdateien klassifiziert. Bei den restlichen ca. 9,38 % handelte es sich jedoch um Textdateien. Sie wurden somit falsch klassifiziert. Um herauszufinden, wie die Falsch-Klassifizierung verhindert werden kann, wurde die Menge detaillierter untersucht. In der Menge kamen insgesamt sieben verschiedene Dateierweiterungen vor. Die Menge wurde anhand dieser in sieben Untermengen aufgeteilt. Für jede Untermenge wurden die Dateiinhalte händisch klassifiziert.

ASPX-Dateien:

Es wurden 10 ASPX-Dateien erkannt. Alle Dateiinhalte waren Textdateien. Die Erweiterung „aspx“ wurde demnach fälschlicherweise auf die Liste der Nicht-Textdateien gesetzt.

ASHX-Dateien:

Für einige der Erweiterungen der Liste der technischen Dateien war von Beginn an nicht ganz sicher, ob sie vielleicht doch auf die Liste der Textdateien gehören. Dazu gehörten die Erweiterungen „ashx“, „aspx“, „php“, „rss“ und „xml“. Wie die Untersuchung der ASPX-Dateien zeigte, wurde die Erweiterung „aspx“ fälschlicherweise auf der Liste der Nicht-Textdateien gesetzt. Aufgrund dieser Erkenntnis wurde auch eine Untersuchung der verwandten ASHX-Dateien durchgeführt. Beide Dateierweiterungen gehören zu dem Web-Framework ASP.NET[1]. Die Untersuchung basierte auf den positiven Dateien des Level-2-Downloads, da die ASHX-Dateien nicht in der hier untersuchten Menge der Nicht-Textdateien vorkamen. Die Untersuchung ergab, dass auch die Dateierweiterung „ashx“ von der Liste der Nicht-Textdateien auf die Liste der Textdateien verschoben werden sollte.

AXD-Dateien:

Von der Dateierweiterungs-Erkennung wurde nur eine AXD-Datei erkannt. Diese enthielt, wie erwartet, JavaScript-Code. Es wurden weitere AXD-Dateien aus dem Level-2-Download inspiziert. Diese enthielten ebenfalls alle JavaScript-Code. An dieser Stelle ist demnach kein Korrektur oder Ausbesserung der Dateierweiterungs-Erkennung nötig.

1 http://www.asp.net

PHP-Dateien:

Arten des Dateiinhaltes			Absolute Häufigkeit	Relative Häufigkeit
Nicht-Textdateien:	Binärdateien:		1	ca. 0,32 %
	Technische Dateien:		285	ca. 91,64 %
Textdateien:	Englisch:	Mit Tags:	18	ca. 5,79 %
		Ohne Tags:	4	ca. 1,29 %
	Fremdsprachig:		3	ca. 0,96 %

Tabelle 9: Ergebnis der manuellen Klassifikation von Dateiinhalten des positiven Level-1-Downloads. Es wurden die Dateien klassifiziert, die von der Dateierweiterungs-Erkennung als PHP-Dateien erkannt wurden und von der Spracherkennung als „en“ (englisch) erkannt wurden. Die Spracherkennung verwendete dabei die Sprachprofile „java_script“ und „none“ sowie die Standard-Profile.

Wie die Tabelle 9 verdeutlicht, handelte es sich bei ca. 8,04 % der Dateiinhalte um Textdateien. Die restlichen ca. 91,96 % wurden von der Dateierweiterungs-Erkennung zurecht als Nicht-Textdateien klassifiziert. Eine nähere Betrachtung der Dateinamen ergab, dass unter den 25 Textdateien 16 Dateien einen Name besaßen, der mit „index.php“ begann. Weitere acht Dateien besaßen einen Name der mit „rss_view.php“ begann. Diese acht Dateien stammten jedoch alle von der selben Website.

Unter den Nicht-Textdateien befand sich eine Binärdatei. Sie trug den Namen „css.-php“ und ihr Inhalt war „GIF89a# # ï¿½ !ï¿½'Created with GIMP by donncha@o-caoimh.ie !ï¿½## # , # # ##L# ;“ Der Inhalt kann hier jedoch nicht richtig Darge-stellte werden. Hinter den als Doppelkreuze interpretierten UTF8-Zeichen verbergen sich sieben „START OF HEADING“-Zeichen mit dem Hex-Code 0x01, zwei „START OF TEXT“-Zeichen mit dem Hex-Code 0x02 und ein „END OF TRANSMISSION“-Zeichen mit dem Hex-Code 0x04.

285 der, als Nicht-Textdateien erkannten, Dateien, besaßen einen technischen Inhalt. Unter ihnen befanden sich zwei Dateien namens „server.php“. Die restlichen Dateien hießen „xmlrpc.php“. Die „xmlrpc.php“-Dateien hatten Inhalte, wie z. B. „WordPress http://wordpress.org/ http://www.ip21.co.uk“ oder auch „XML-RPC server accepts POST requests only.“ Diese Inhalte könnten zwar als Textdateien angesehen werden, da sie teilweise aus schlichtem Text bestehen, sie sind jedoch technische Meldungen und haben nichts mit den beiden Klassen positiv und negativ zu tun. Dazu kommt, dass alle

technischen Dateien nur einen sehr kurzen Inhalt hatten. Es wurde daher davon ausgegangen, dass nicht herausgefilterte, technische PHP-Dateien keinen bedeutenden Einfluss auf das Dokument und dessen Klassifikation haben. Basierend auf diesen Beobachtungen und Schlussfolgerungen, wurde entschieden, dass die Erweiterung „php“ auf die Liste der Textdateien verschoben werden sollte. Dabei werden aber die Dateinamen „css.php“, „server.php“ und „xmlrpc.php“ als Nicht-Textdateien klassifiziert.

RDF-Dateien:

Es wurden drei RDF-Dateien erkannt. Bei allen drei Dateiinhalten handelte es sich um englische Textdateien. Diese enthielten noch XML-Tags, aber diese werde, wie bereits erwähnt, in Zukunft durch jsoup entfernt. Die Erweiterung „rdf“ sollte daher auf die Liste der Textdateien verschoben werden.

RSS-Dateien:

Analog zu den RDF-Dateien wurde auch die Erweiterung „rss“ auf die Liste der Textdateien verschoben, da die Inhalte der RSS-Dateien nun auch keine XML-Tags mehr enthalten. Die hier getestete Menge enthielt keine RSS-Dateien. Daher wurde eine stichprobenartige Untersuchung der Dateien des positiven Level-2-Downloads vorgenommen. Es waren ebenfalls alles Textdateien.

SVG-Dateien:

Es wurden fünf SVG-Dateien erkannt. Ihre Inhalte wurden alle als Textdatei klassifiziert. Die Erweiterung „svg“ sollte trotzdem nicht auf die Liste der Textdateien verschoben werden, da alle Inhalte nur kurze Copyright- bzw. Herkunftsinformationen enthielten. Darunter Inhalte, wie z.B. „Copyright (C) 2012 by original authors @ fontello.com“ oder „This is a custom SVG webfont generated by Font Squirrel.“ Diese Inhalte haben mit den beiden Dokumentklassen positiv und negativ nichts zu tun.

TXT-Dateien:

Es wurden 700 TXT-Dateien entdeckt. Bei allen handelte es sich um Nicht-Textdateien, genauer gesagt, um Dateien namens „robots.txt“. Es würde demnach reichen, den Dateinamen „robots.txt“ auszuschließen. Es wurde davon ausgegangen, dass das auftreten einer TXT-Datei, die keine „robots.txt“ ist, sehr unwahrscheinlich ist. Desweiteren wurde davon ausgegangen, dass es sich ggf. wahrscheinlich um eine technische Datei handeln würde. An dieser Stelle wurde demnach kein Korrektur oder Ausbesserung an der Dateierweiterungs-Erkennung vorgenommen.

XML-Dateien:

Arten des Dateiinhaltes			Absolute Häufigkeit	Relative Häufigkeit
Nicht-Textdateien:	Technische Dateien:		57	ca. 46,72 %
Textdateien:	Englisch:	Mit Tags:	17	ca. 13,93 %
		Ohne Tags:	48	ca. 39,34 %

Tabelle 10: Ergebnis der manuellen Klassifikation von Dateiinhalten des positiven Level-1-Downloads. Es wurden die Dateien klassifiziert, die von der Dateierweiterungs-Erkennung als XML-Dateien erkannt wurden und von der Spracherkennung als „en" (englisch) erkannt wurden. Die Spracherkennung verwendete dabei die Sprachprofile „java_script" und „none" sowie die Standard-Profile.

Wie die Tabelle 10 verdeutlicht, handelte es sich bei ca. 53,29 % der Dateiinhalte um Textdateien. Die restlichen ca. 46,72 % wurden von der Dateierweiterungs-Erkennung zurecht als Nicht-Textdateien klassifiziert. Eine nähere Betrachtung der Dateinamen zeigte, dass von den 65 Textdateien acht die Suchfunktion einer Website bereitstellten. Fünf dieser Dateien hießen „osd.xml" und die restlichen drei „opensearch.xml" bzw. „OpenSearch.xml". Diese Dateien sollten in die Generierung der Dokumente einbezogen werden, da sie eine kurze Beschreibung enthalten, nach was auf der Website gesucht werden kann. Ihr Inhalt ist demnach, wie eine kurze Inhaltsangabe einer Website. Auf der Seite der Nicht-Textdateien kamen vier Dateien namens „sitemap.xml" vor. Diese enthielten lediglich zeitliche Angaben und Links. Ihr Inhalt wurde daher eher als technisch angesehen und sie wurden zu den Nicht-Textdateien gezählt. Die restlichen 53 Nicht-Textdateien hatten ebenfalls einen technischen Inhalt. Sie hießen alle „wlwmanifest.xml". Ihre Inhalte waren sich alle sehr ähnlich und sahen beispielsweise, wie folgt, aus: „WordPress Yes Yes WordPress images/wlw/wp-icon.png images/wlw/wp-watermark.png View site Dashboard {blog-postapi-url}/../wp-admin/ {blog-postapi-url}/../wp-admin/post.php?action=edit&post={post-id} 0 Manage Comments images/wlw/wp-comments.png {blog-postapi-url}/../wp-admin/edit-comments.php". Die Untersuchung zeigte, dass die knappe Mehrheit der XML-Dateien Textdateien waren und die Nicht-Textdateien standardisierte Namen zu besitzen schienen. Die Erweiterung „xml" wurde daher auf die Liste der Textdateien verschoben und es wurden nur noch die Dateinamen „sitemap.xml" und „wlwmanifest.xml" verworfen bzw. als Nicht-Textdateien klassifiziert.

Menge der echten Textdateien

Die Anzahl der Dateien, die von der Dateierweiterungs-Erkennung als echte Textdateien klassifiziert wurden, betrug 31.104. Dies sind ca. 96,26 % aller als englisch klassifizierten bzw. untersuchten Dateien. Diese Dateien würden von der Dateierweiterungs-Erkennung zur Generierung der Dokumente zugelassen werden.

Es wurde eine Stichprobe von 1.000 Dateiinhalten händisch untersucht und klassifiziert, um herauszufinden, ob alle Nicht-Textdateien verworfen wurden oder ob weitere Verbesserungen nötig sind. Diese 1.000 Dateien entstammten den Websites 420 unterschiedlicher URLs. Die Resultate der Klassifikation sind in Tabelle 11 dargestellt.

Arten des Dateiinhaltes				**Absolute Häufigkeit**	**Relative Häufigkeit**
Nicht-Textdateien:	Technische Dateien:	„robots.txt“-Dateien		1	0,1 %
Textdateien:	Englisch:		Mit Tags:	10	1,0 %
			Ohne Tags:	987	98,7 %
			Ohne Tags:	2	0,2 %

Tabelle 11: Ergebnis der manuellen Klassifikation von Dateiinhalten des positiven Level-1-Downloads. Es wurden die Dateien klassifiziert, die von der Dateierweiterungs-Erkennung als echte Textdateien erkannt wurden und von der Spracherkennung als „en“ (englisch) erkannt wurden. Die Spracherkennung verwendete dabei die Sprachprofile „java_script“ und „none“ sowie die Standard-Profile.

Wie in Tabelle 11 zu sehen, wurden 99,9 % bzw. 999 der hier untersuchten Dateien richtig als Textdateien erkannt. Es wurde nur eine Nicht-Textdatei fälschlicherweise als Textdatei erkannt. Es handelte sich dabei um den Dateiinhalt einer „robots.txt“. TXT-Dateien werden jedoch von der Dateierweiterungs-Erkennung als Nicht-Textdateien klassifiziert. Die Datei hätte demnach an dieser Stelle nicht vorkommen dürfen. Eine Betrachtung des Dateinamens zeigte, dass die Datei vorkam, weil sie „robots.txt.html“ hieß. Es wird vermutet, dass die HTML-Erweiterung durch die Wget-Option „--adjust-extension“ angehängt wurde. Diese Option ergänzt u. a. fehlende HTML-Erweiterungen für Dateien der MIME-Typen „application/xhtml+xml“ und „text/html“. [16]

Eine Suche in allen 31.104 als Textdatei erkannten Dateien ergab, dass insgesamt 31 Dateien bzw. ca. 0,1 % der Dateien den Namen „robots.txt.html“ trugen. Daraufhin wur-

de die Dateierweiterungs-Erkennung so modifiziert, dass alle Dateinamen, die mit „robots.txt“ beginnen als Nicht-Textdateien klassifiziert werden.

3.4.5 Verbesserung des Programms

Wie die Untersuchungen aus Kapitel 3.4.4 zeigten, sollten einige Veränderungen am Programm vorgenommen werden. Die beiden generierten Sprachprofile „none“ und „java_script“ erwiesen sich als nützlich und wurden daraufhin in die Liste der Sprachprofile der Google Language-Detection-Bibliothek übernommen.

Wie bereits in Kapitel 3.4.4.1 erläutert wurden Maßnahmen unternommen, um die Ergebnisse der Spracherkennung zu verbessern. Zum Einen wurde ein Mechanismus eingebaut, der URLs aus den Dateiinhalten entfernt, bevor diese an die Spracherkennung übergeben werden. Dazu wird der reguläre Ausdruck „(ht|f)tp(s?)://[^\\s]*“ benutzt. Zum Anderen wurde die HTML-Tag-Entfernung durch jsoup von einem einzelnen Durchlauf auf mehrere Durchläufe erhöht. Sie wird so oft wiederholt, bis jsoup keine Änderungen am Dateiinhalt mehr vornimmt.

Wie Kapitel 3.4.4.2 zeigte, stellt die Dateierweiterungserkennung eine gute Lösung für das Problem dar, dass einige Nicht-Textdateien von der Spracherkennung als englisch klassifiziert wurden. Als Reaktion auf die Ergebnisse aus Kapitel 3.4.4.2 wurden folgende Veränderungen vorgenommen:

1. Der Blacklist-Modus wurde als Standard übernommen. Es gibt jedoch die Möglichkeit beim Programmaufruf den Whitelist-Modus zu aktivieren. Dazu muss der Parameter „use_whitelist“ angegeben werden.
2. Die Dateierweiterungen „aspx“, „ashx“, „rdf“, „rss“ und „xml“ wurden auf die Liste für Textdateien verschoben.
3. Dateinamen mit den Präfixen „css.php“, „server.php“, „xmlrpc.php“, „sitemap.xml“, „wlwmanifest.xml“ und „robots.txt“ werden als Nicht-Textdateien klassifiziert.

Unabhängig vom Test der Dateierweiterungs-Erkennung wurden weitere Dateierweiterungen zu der Liste der Nicht-Textdateien hinzugefügt. Es handelte sich dabei um „jsf“, „jsp“, „odt“, „ogv“, „ott“, „pgp“, „pot“, „potx“, „pps“, „rft“, „rm“, „tif“, „thmx“, „vbs“ und „vcs“. Die finalen Listen der Dateierweiterung sind in Tabelle 12 dargestellt.

Art des Dateiinhaltes	Dateierweiterungen
Nicht-Textdateien:	7zip, gzip, rar, zip, mp3, wav, ogg, cfm, pdf, doc, docx, odt, ott, pot, potx, pps, ppt, pptx, rft, thmx, txt, vcf, vcs, xls, xlsx, eot, ttf, woff, bmp, gif, ico, jpeg, jpg, png, svg, svgz, swf, tif, pgp, rm, axd, cgi, css, js, jsf, jsp, php, vbs, flv, mp4, ogv, wmv
Textdateien:	aspx, ashx, html, htm, rdf, rss, xhtml, xml

Tabelle 12: Übersicht der Listen aus der Dateierweiterungs-Erkennung

3.4.6 Anwendung des Programms

Nach dem das Java-Programm zum Erzeugen der Dokumente erstellt, getestet und verbessert wurde, wurden die positiven und negativen Dokumente aus den heruntergeladenen Dateien des Level-0-, Level-1- und Level-2-Downloads generiert. In Tabelle 13 befindet sich ein Übersicht darüber, wie viele Dokumente erzeugt wurden. Der Korpus umfasst demnach, je nach gewählter Rekursionstiefe des Downloads, 1.969, 2.449 oder 2.491 Dokumente.

Klasse:	**positiv**	**negativ**
Anzahl der URLs:	1854	1220
Anzahl der heruntergeladenen Websites (gleich für alle 3 Level):	1778	1182
Anzahl der Dokumente (generiert aus Level-0-Dateien):	985	984
Anzahl der Dokumente (generiert aus Level-1-Dateien):	1411	1038
Anzahl der Dokumente (generiert aus Level-2-Dateien):	1445	1046

Tabelle 13: Vergleich der Anzahl der URLs, der Anzahl der heruntergeladenen Websites der URLs und der Anzahl der generierten Dokumente.

4 Testen der Klassifikationsverfahren und Implementierung des Besten

4.1 RapidMiner Prozessaufbau

Für das Testen der verschiedenen Klassifikationsverfahren wurde das Programm RapidMiner[1] in der 64-bit Variante der Version 5.3.015 verwendet. Des Weiteren wurden die RapidMiner Text-Mining-Extensions in der Version 5.3.2 und die Weka-Extension in der Version 5.3.1 verwendet. Zum Beschleunigen der Tests kam die Prallel-Processing-Extension in der Version 5.3.0 zum Einsatz. Das Programm und alle genannten Erweiterungen stehen auf Sourceforge zum kostenlosen Download zur Verfügung.[2] RapidMiner kann als Experimentierumgebung für maschinelles Lernen benutzt werden. Es bietet eine grafische Oberfläche mit der eine Verarbeitungsprozess aus diversen Operatoren zusammengestellt werden kann. Ein Übersicht der Operatoren und ihrer einstellbaren Parameter befindet sich in [23]. Sofern im Folgenden nicht anders angegeben, wurden die Parameter eines Operators auf ihrem Standardwert gelassen.

Für die zum Testen erstellten Hauptprozesse wurde das Encoding UTF-8 eingestellt, da die Dokumente des Korpus bei der Generierung in UTF-8 kodiert wurden. Den Anfang jedes Prozesses machte der Operator „Process Documents from Files“. Er dient dazu Dokumente bzw. Textdateien in einen Wordvektor umzuwandeln. Seine Ausgabe wird als Example-Set bezeichnet. Für das Laden der Textdateien können mehrere Ordnerpfade angegeben werden. Für jeden Pfad kann dabei eine Klasse definiert werden. Im Rahmen der Bachelorarbeit wurde für einen Test jeweils ein Ordner mit positiven und ein Ordner mit negativen Dokument angegeben. Beide Dokumentmengen basierten dabei auf dem selben Download-Level.

Ein weiterer Parameter ist die Art der Vector-Creation. Zur Auswahl stehen TF-IDF, Term-Frequency, Term-Occurrences und binary Term-Occurrences. Es wurde TF-IDF gewählt, da es die Relevanz eines Terms für ein Dokument nicht nur innerhalb des Dokumentes bestimmt, sondern den gesamten Korpus beachtet. [7, pp. 588-589]
Ein weiterer Parameter ist die Prune-Methode. Sie dient der Reduzierung des Vokabulars, welches durch den Operator erstellt wird. Für die durchgeführten Test wurde, wo möglich, kein Pruning verwendet und ein weiterer Test mit prozentualem Prune durch-

1 https://rapidminer.com/products/studio/
2 http://sourceforge.net/projects/rapidminer/

geführt. Die verwendeten Standardwerte sorgten dafür, dass Wörter die in weniger als 3 % der Dokumente vorkamen und Wörter die in mehr als 80 % der Dokumente vorkamen gelöscht wurden. Es wird davon ausgegangen, dass Wörter die in vielen Dokumente oder in nur ganz wenigen Dokumenten vorkommen kein typisches Merkmal einer Klasse sind.

In den Operator „Process Documents from Files“ können weitere Operatoren geschachtelt werden. Als erstes wurden die Dokumente an den Operator „Transform Cases“ übergeben, um alle Texte in den „lower case“ umzuwandeln. Auf diese Weise wird in der zu generierenden Wortliste nicht mehr zwischen Klein- und Großschreibung unterschieden. Dies ist nützlich, damit gleiche Wörter nicht mehrfach vorkommen, wenn sie z.B. an einem Satzanfang großgeschrieben wurden, aber sonst kleingeschrieben werden. Als nächstes wurden die Dokumente an den Operator „Tokenize“ übergeben. Dieser zerlegt den Text eines Dokuments in eine Sequenz aus Tokens. Dabei kann eingestellt werden, wie der Text zerlegt werden soll. RapidMiner bietet u. a. die einfache Zerlegung anhand von Nicht-Buchstaben an oder eine komplexere Zerlegung anhand von linguistischen Tokens. Es wurde der Modus „non letters“ gewählt, da er die besseren Resultate lieferte.

Als nächstes wurde der Operator „Filter Tokens (by Length)“ verwendet, um Tokens mit weniger als zwei Zeichen und Tokens mit mehr als 25 Zeichen zu entfernen. Es wurde davon ausgegangen, dass es sich bei diesen Tokens nie um Wörter handeln wird. Als nächstes wurden die Dokumente an den Operator „Filter Stopwords (English)“ übergeben. Er verkleinert das Vokabular erneut, in dem hochfrequente Wörter, sogenannte Stoppwörter, gefiltert werden. Zu den englischen Stoppwörtern gehören u. a. „I“, „the“, „a“ und „in“. Auf Stoppwörter kann verzichtet werden, da sie aufgrund ihrer Häufigkeit wahrscheinlich sehr oft und in allen Dokumenten vorkommen. Sie stellen somit kein Unterscheidungsmerkmal dar. Als nächster Operator wurde „Stem (Porter)“ gewählt. Er wendet den Porter-Stemmer-Algorithmus[1] auf die Tokens an. Stemming bezeichnet das Zurückführen eines Wortes auf seinen Wortstamm. Der Porter-Stemmer-Algorithmus setzt dies um, in dem er eine iterative und regelbasierte Ersetzung von Wortsuffixen vollzieht, bis ein Wort auf seine minimale Länge reduziert wurde. Durch den Einsatz von Stemming kann das Vokabular weiter reduziert werden, da die ver-

1 http://tartarus.org/~martin/PorterStemmer/

schiedenen Beugungsformen eines Wortes zu einem Token zusammengefasst werden. Neben dem Porter-Stemmer stehen in RapidMiner noch andere Stemming-Verfahren zur Auswahl. Es wurde der Porter-Stemmer gewählt, da er eine üblicher Stemmer für das Englische ist, dessen hohe Wirksamkeit schon wiederholt gezeigt wurde. [8, p. 31] Dies war der letzte Operator innerhalb des „Process Documents from Files“-Operators. Er sorgte anschließend dafür, dass aus den so erstellten Tokens ein Wortvektor für jedes Dokument generiert wird. Die Gesamtheit der Wortvektoren bilden das Example-Set, welches die Ausgabe des Operators darstellt.

Das ausgegebene Example-Set des „Process Documents from Files“-Operators wurde an den Trainings-Set-Eingang des Operators „X-Validation“ weitergegeben. Dieser führt eine Kreuzvalidierung durch, wie in Kapitel 2.5.2.2 vorgestellt. Für den Parameter „sampling type“ wurde der Standardwert „stratified sampling“ verwendet. Durch ihn erfolgt die Aufteilung des Example-Sets in gleichgroße Untermengen zufällig. Dabei wird allerdings beachtet, dass die Verteilung der Klassen innerhalb einer Untermenge so ist, wie im ganzen Example-Set. Das n der sogenannten n-fold Cross-Validation wurde mittels „number of validations“ auf zehn gesetzt.

Bei dem „X-Validation“-Operator handelte es sich erneut um einen geschachtelten Operator. Er enthält einen Trainings- und einen Test-Unterprozess. Im Trainings-Unterprozess wurde jeweils der Operator des getesteten Klassifikationsverfahrens eingefügt. An diese wurde stets das an den „X-Validation“-Operator übergebene Example-Set als Trainings-Set weitergeleitet. Die Ausgabe war stets das trainierte Modell. Dieses wurde an den Test-Unterprozess weitergereicht und diente dort als Eingabe für den Operator „Apply Model“. Dieser enthielt als weitere Eingabe das vom „X-Validation“-Operator erzeugte Trainings-Set. Der Operator „Apply Model“ wendet das eingegebene Model auf das eingegebene Example-Set bzw. Trainings-Set an. Dessen Klassenlabels werden ignoriert und es wird ein neu gelabeltes Example-Set als Ausgabe erzeugt. Das erzeugte Example-Set wurde anschließen an den Operator „Performance“ weitergeleitet. Dieser dient zum evaluieren der Performanz. Die Ergebnisse stehen als Performance-Vector an einem Ausgang des Operators bereit. Dieser Vektor enthält u. a. die Recall- und Precision-Werte. Der Ausgang des Performance-Vektors wurde an den Ausgang des Test-Unterprozesses und somit an den Ausgang des „X-Validation“-Operators weitergeleitet und von dort aus an den Ausgang des Hauptprozesses durchgereicht.

Wie bereits erwähnt, wurden innerhalb des Trainings-Unterprozesses die verschiedenen, getesteten Klassifikationsoperatoren eingefügt. Es wurden verschiedene Variationen von Naive Bayes, k-NN, logistischer Regression und SVMs verwendet. Auf eine Erklärung und Differenzierung der Funktionsweisen der einzelnen Operatoren wird verzichtet, da dies den Rahmen der Bachelorarbeit übersteigen würde. Welche Operatoren genau benutzt wurden und welche Paramter gewählt wurden, ist aus den Tabellen im Anhang ersichtlich.

4.2 Auswertung der Testergebnisse

Der im vorherigen Kapitel beschriebene RapidMiner-Prozess wurde genutzt, um verschiedene Klassifikationsverfahren zu Testen. Das beste Verfahren sollte anschließend in das Klassifikationsprogramm implementiert werden. In die Entscheidung für das beste Verfahren ging hauptsächlich ein, wie viele der relevanten Dokumente bzw. Websites als relevant klassifiziert wurden. Das Klassifikationsprogramm soll vorrangig relevante Websites erkennen. Ob eine nicht relevante Website als relevant klassifiziert wird, ist zwar auch wichtig, aber nicht in erster Linie. Das liegt daran, dass die als relevant klassifizierten Websites, im Gegensatz zu den nicht relevanten, noch einmal von einer Person geprüft werden. Dies geschieht, wenn diese das Firmenprofil in der IPIB anlegt. Da dieser Schritt auf lange Sicht auch automatisiert werden soll, ist eine hohe Erkennungsrate der nicht relevanten Websites jedoch auch wünschenswert. Ein weiteres Kriterium war ursprünglich, wie gut sich das Klassifikationsverfahren implementieren lässt bzw. ob es eine Bibliothek dafür gibt und wie populär und wie gut dokumentiert diese ist.
RapidMiner bietet jedoch die Möglichkeit in ein Java-Programm integriert zu werden. Auf diese Weise konnte der gesamte Prozess eins zu eins in das bereits entwickelte Java-Programm implementiert werden. Eine erneute Evaluation des Klassifikationsverfahrens innerhalb des Klassifikationsprogramms war somit hinfällig. Für die Auswahl des besten Verfahrens war somit nur noch der Recall der Klasse positiv und negativ sowie die Accuracy zu beachten.

Um bei der Auswertung der Tests die Übersichtlichkeit zu erhalten, wurden alle Tabellen in den Anhang ausgelagert. Für jeden Klassifikationsoperator und die getätigten Einstellungen an ihm, wurde eine Tabelle angelegt. Die Tabellen beinhalten die Daten des Performance-Vektors für sechs verschiedene Ausführungen. Dazu gehören die drei

Ausführungen auf dem Korpus der Level-0-, Level-1- und Level-2-Dokumente. Jede der drei Ausführungen wurde zweimal durchgeführt. Einmal ohne Prune und einmal mit prozentualem Prune von unter 3 % und über 80 %. Sofern in der Tabellenbeschreibung nicht anders angegeben, wurden die Parameter der Operatoren auf ihren Standardwerten gelassen. Überall da, wo ein „k. A." in den Tabellen steht, konnten keine Angaben gemacht werden, weil der Prozess mehr Arbeitsspeicher benötigte, als zur Verfügung stand. Dies geschah hauptsächlich bei den Varianten von k-NN und der logistischen Regression. Für das Ausführen von RapidMiner waren stets 12 bis 13 GB Arbeitsspeicher reserviert. Manche Prozesse wurden auch abgebrochen, weil sie zu hängen schienen.

Für die Tests auf dem Level-2-Korpus wurde ein Dokument aus dem Korpus entfernt. Es handelte sich dabei, um das größte, vorhandene Dokument. Die Datei benötigte ca. 700 MB Speicherplatz und sorgte dafür, dass die Ausführungen der Prozesse sehr viel Zeit benötigten. Zum Vergleich sei erwähnt, dass die durchschnittliche Dokumentengröße bei ca. 1,8 MB lag. Das zweit größte Dokument benötigte ca. 319 MB Speicherplatz, das dritt-, viert- und fünftgrößte nur noch zwischen 100 und 200 MB.

Bei dem Ausführen der Prozesse traten des Öfteren Fehler auf, wenn der X-Validation-Operator der Prallel-Processing-Extension in Verbindung mit Klassifikationsoperatoren der Weka-Extension genutzt wurde. Seine Verwendung führte Teilweise zu der Warnung: „WARNING: Error executing task asynchronously: com.rapidminer.operator.UserError: Input ExampleSet does not have a label attribute." Diese Prozessausführungen brachen dann mit einem Fehler über das Fehlen des Label-Attributes ab. In einigen Fällen erfolgte lediglich eine Warnung zur asynchronen Ausführung. Fehlende Label-Attribute wurden dabei nicht erwähnt. Diese Prozessausführungen brachen nicht ab, schienen aber, aufgrund sehr viel längerer Ausführungszeiten, in einer Endlosschleife zu hängen.

4.2.1 Variationen von Naive Bayes

Die Testergebnisse der Operatoren, für die verschiedenen Variationen von Naive Bayes, werden in Tabelle 18 bis 22 dargestellt. Die besten Ergebnisse lieferte im Schnitt der Operator „W-ComplementNaiveBayes" mit dem Parameter N = off. Für die Level-0-Dokumente war jedoch der Operator „W-NaiveBayesMultinomial" ein wenig besser. Die Ergebnisse sind in Tabelle 14 zusammengefasst.

RapidMiner-Operator:	W-NaiveBayes Multinomial	W-Complement NaiveBayes mit N = off	W-Complement NaiveBayes mit N = off
Verwendete Dokumente:	Level-0-Download	Level-1-Download	Level-2-Download
Prune unter 3 % und über 80 %:	Ja	Ja	Ja
Class-Recall negativ (in %):	91,16	89,88	87,48
Class-Recall positiv (in %):	91,88	94,97	95,71
Class-Precision negativ (in %):	91,81	92,93	93,65
Class-Precision positiv (in %):	91,23	92,73	91,34
Accuracy (in %):	91,52 ± 1,52	92,81 ± 1,17	92,25 ± 1,41

Tabelle 14: Übersicht der besten Naive Bayes Ergebnisse.

4.2.2 Variationen von K-NN

Die Ergebnisse der verschiedenen Ausführungen von k-NN sind in Tabelle 23 bis 29. Bei k-NN wurde die Parameter-Einstellung „weighted vote = on“ nicht aufgeführt. Sie lieferte stets etwas schlechtere Ergebnisse. K-NN wurde für den Level-1-Korpus mit Prune von k = 1 bis k = 20 getestet. Diverse k's sind nicht im Anhang aufgeführt. Sie lieferten stets ähnliche, aber etwas schlechtere Ergebnisse, wie ihre Vorgänger oder Nachfolger. Für das Level-1-Korpus wurden die besten Ergebnisse mit k = 10 ohne Prune erzielt. Aufgrund eines zu hohen Arbeitsspeicherbedarfs konnten die Test auf dem Level-1- und Level-2-Korpus nur mit Prune ausgeführt werden. Die besten Ergebnisse lieferte in beiden Fällen k = 16. Die besten Ergebnisse der drei Korpora sind in Tabelle 15 zusammengefasst.

RapidMiner-Operator:	k-NN mit k = 10	k-NN mit k = 16	k-NN mit k = 16
Verwendete Dokumente:	Level-0-Download	Level-1-Download	Level-2-Download
Prune unter 3 % und über 80 %:	Nein	Ja	Ja
Class-Recall negativ (in %):	84,76	89,50	66,73
Class-Recall positiv (in %):	95,74	93,76	97,92
Class-Precision negativ (in %):	95,21	91,35	95,88
Class-Precision positiv (in %):	86,28	92,39	80,25
Accuracy (in %):	90,25 ± 1,61	91,96 ± 1,86	84,82 ± 2,46

Tabelle 15: Übersicht der besten k-NN Ergebnisse.

4.2.3 Variationen der logistischen Regression

Die Ergebnisse der verschiedenen Ausführungen der logistischen Regression befinden sich in Tabelle 30 bis 37. Als bestes Ergebnis auf allen drei Korpora wurde der Operator „W-SimpleLogistic“ mit Prune ausgewählt. Für das Level-0-Korpus gab es dabei zwei weitere Kandidaten. Zum Einen hatte das Ergebnis ohne Prune zwar eine etwas höhere Accuracy, dafür aber die größere Standardabweichung. Zum Anderen hatte der Operator „W-BayesianLogisticRegression“ einen um 0,41 % besseren, positiven Class-Recall. Sein negativer Class-Recall war jedoch um 3,46 % schlechter. Die drei besten Ergebnisse sind in Tabelle 16 zusammengefasst.

RapidMiner-Operator:	W-Simple Logistic	W-Simple Logistic	W-Simple Logistic
Verwendete Dokumente:	Level-0-Download	Level-1-Download	Level-2-Download
Prune unter 3 % und über 80 %:	Ja	Ja	Ja
Class-Recall negativ (in %):	95,33	96,92	94,84
Class-Recall positiv (in %):	90,96	92,70	92,59
Class-Precision negativ (in %):	91,33	90,71	90,26
Class-Precision positiv (in %):	95,12	97,61	96,12
Accuracy (in %):	93,14 ± 2,25	94,49 ± 1,02	93,53 ± 1,99

Tabelle 16: Übersicht der besten Ergebnisse der logistischen Regression.

4.2.4 Variationen von SVMs

Die Ergebnisse der verschiedenen SVM-Varianten sind in Tabelle 38 bis 51 aufgeführt. Für den Operator „Support Vector Machine (LibSVM)“ wurden unter Verwendung des Level-1-Korpus mit Prune und unter Verwendung des SVM-Typs C-SVC wurden verschiedene polynomiale Kernel-Typen getestet. Die Parameter „degree“ wurde dabei von 1 bis auf 10 angehoben. Im Anhang befinden sich nur die Ergebnisse der ersten drei Tests. Für die höheren „degree“-Werte pendelte sich der positive Class-Recall bei ca. 100 % ein, während der negative bei ca. 2 % lag.

Als bestes Ergebnis für den Level-0-Korpus stellte sich die „Support Vector Machine (LibSVM)“ mit Prune und den Parametern „svm type = C-SVC“ und „kernel type = sigmoid“ heraus. Für das Level-1-Korpus wurden mehrfach Accuracys von ca. 95 % erreicht. Es wurde die Variante aus Tabelle 43 ohne Prune gewählt, weil sie die ausgegli-

chensten Recall-Werte lieferte. Das beste Ergebnis auf dem Level-2-Korpus teilen sich die C-SVC LibSVM aus Tabelle 42 und 46. Sie lieferten mit und ohne Prune identische Werte. Die Ergebnisse mit Prune waren allerdings besser. Die drei besten Ergebnisse sind nocheinmal in Tabelle 17 zusammengefasst.

RapidMiner-Operator:	LibSVM vom Typ C-SVC mit Sigmoid-Kernel	LibSVM vom Typ C-SVC mit polynomialem Kernel des Grades 2	LibSVM vom Typ C-SVC mit Sigmoid-Kernel oder linearem Kernel
Verwendete Dokumente:	Level-0-Download	Level-1-Download	Level-2-Download
Prune unter 3 % und über 80 %:	Ja	Nein	Ja
Class-Recall negativ (in %):	97,05	95,47	97,51
Class-Recall positiv (in %):	91,07	95,32	93,70
Class-Precision negativ (in %):	91,56	93,76	91,81
Class-Precision positiv (in %):	96,87	96,62	98,11
Accuracy (in %):	94,06 ± 1,09	95,39 ± 0,87	95,30 ± 1,37

Tabelle 17: Übersicht der besten SVM Ergebnisse.

4.3 Implementierung des Klassifikationsprogramms

Wie bereits im vorherigen Kapitel erwähnt, konnte der RapidMiner-Prozess eins zu eins im Java-Programm nachgebaut werden. Eine erneute Evaluierung des implementierten Klassifikationsverfahrens war somit unnötig.

Das Klassifikationsprogramm basiert auf dem, im Kapitel 3.4 vorgestellten, Java-Programm zur Erzeugung der Dokumente aus heruntergeladenen Websites. Er wurde so modifiziert, dass aus dem Quellpfad nur noch ein Dokument erzeugt wird. Des Weiteren wurde eine abgewandelte Form des RapidMiner-Prozesses aus Kapitel 4.1 hinzugefügt. Der Prozess wurde dahingehend modifiziert, dass er ein einzelnes Dokument klassifiziert und nicht mehr ein Modell trainiert. Bei dem dazu verwendeten Modell handelt es sich um das Modell, welches in Tabelle 43 unter Verwendung des Level-1-Korpus und Prune evaluiert wurde.

5 Kurzzusammenfassung

Die Aufgabe der Bachelorarbeit bestand darin ein Kommandozeilen-Programm für da Fraunhofer MOEZ zu implementieren, welches eine Firmen-Website automatisch klassifiziert. Die Websites sollen dabei als relevant oder nicht relevant für eine analytische Datenbank namens IPIB klassifiziert werden. Die IPIB wird ebenfalls am Fraunhofer MOEZ entwickelt und beherbergt u. a. Firmenprofile aus dem Bereich IP. Um ein Klassifikationsmodell für das Kommandozeilen-Programm zu trainieren wurde als erstes ein Korpus erstellt. Dazu wurden URLs zu relevanten und nicht relevanten Websites gesammelt. Die Websites wurden anschließen heruntergeladen. Dies erfolgte in Rekursionstiefen null, eins und zwei. Die heruntergeladenen Dateien wurden zu Dokumenten zusammengefügt. Dabei wurde für jede Website bzw. URL ein Dokument erzeugt. Das Generieren der Dokumente erledigte ein extra dafür entwickeltes Java-Programm. Dokumente sollten nur aus dem englischen Text einer Website bestehen. Daher wurde eine Spracherkennung in das Java-Programm eingebaut. Da diese zu Beginn teilweise Binärdateien und technische Dateien, wie z.B. Quellcode-Dateien, als englisch klassifizierte, wurden im Rahmen dieser Arbeit zwei zusätzliche Sprachprofile angefertigt. Zusätzlich dazu wurde noch eine Dateierweiterungs-Erkennung entwickelt, um weitere technische Dateien beim Generieren der Dokumente zu verwerfen, die sonst nicht durch die erweiterte Spracherkennung herausgefiltert werden würden. Des weiteren wurden HTML- und XML-Tags sowie URLs aus den Dateien entfernt. Aus den generierten Dokumenten der drei Rekursionstiefen wurden so drei Korpora erzeugt. Diese wurden schließlich genutzt, um im Programm RapidMiner diverse Klassifikationsverfahren zu testen. Als bestes stellte sich dabei die Support Vector Machine der LibSVM vom Typ C-SVC mit dem polynomialem Kernel des Grades 2 heraus, wenn es ohne Prune auf dem Korpus der Rekursionstiege eins trainiert wurde. Das Model erzielte eine Accuracy con 95,39 ± 0,87 % und hatte dabei einen positiven Class-Recall von 95,32 % und einen negativen von 95,47 %. Das zu entwickelnde Kommandozeilen-Programm basierte auf den Java-Programm, welches zum Generieren der Dokumente verwendet wurde. Der verwendete RapidMiner-Prozess wurde so modifiziert, dass er eine Datei klassifiziert und dabei das Modell aus dem Test verwendet. Da RapidMiner dies unterstützt, wurde der Prozess eins zu eins in das Kommandozeilen-Programm übernommen.

Literaturverzeichnis

[1] “Fraunhofer MOEZ - Fraunhofer MOEZ.” [Online]. Available: http://www.moez.fraunhofer.de/. [Accessed: 25-May-2014].
[2] “Leistungsangebot,” *Fraunhofer MOEZ*. [Online]. Available: http://www.moez.fraunhofer.de/de/leistungen.html. [Accessed: 25-May-2014].
[3] “Innovative Transfersysteme,” *Fraunhofer MOEZ*. [Online]. Available: http://www.moez.fraunhofer.de/de/gf/Transfersysteme.html. [Accessed: 25-May-2014].
[4] “Arbeitsgruppe Competitive Intelligence,” *Fraunhofer MOEZ*. [Online]. Available: http://www.moez.fraunhofer.de/de/gf/Transfersysteme/AG-Competitive-Intelligence.html. [Accessed: 25-May-2014].
[5] “IP Industry Base (IPIB) - eine analytische Datenbank zu den Akteuren der weltweiten IP-Service-Industrie,” *Fraunhofer MOEZ*. [Online]. Available: http://www.moez.fraunhofer.de/de/gf/Transfersysteme/projekte/laufendeProjekte/IPindustriebase.html. [Accessed: 25-May-2014].
[6] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A design science research methodology for information systems research,” *J. Manag. Inf. Syst.*, vol. 24, no. 3, pp. 45–78, 2007.
[7] K.-U. Carstensen, C. Ebert, C. Ebert, S. Jekat, H. Langer, and R. Klabunde, *Computerlinguistik und Sprachtechnologie*. Springer-Verlag, 2010.
[8] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*, 1 edition. Cambridge University Press, 2008.
[9] B. Liu, *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*, 1st ed. 2007. Corr. 2nd printing edition. Springer, 2009.
[10] D. Jurafsky, “Practical Issues in Text Classification.” [Online]. Available: https://class.coursera.org/nlp/lecture/29. [Accessed: 04-Feb-2015].
[11] A. Ng, “Supervised Learning.” [Online]. Available: https://class.coursera.org/ml-005/lecture/3. [Accessed: 04-Feb-2015].
[12] “Language Detection Library for Java.” [Online]. Available: http://www.slideshare.net/shuyo/language-detection-library-for-java. [Accessed: 11-Jan-2015].
[13] A. Ng, “Machine Learning.” [Online]. Available: https://class.coursera.org/ml-005/lecture. [Accessed: 08-Feb-2015].
[14] T. Berners-Lee, R. T. Fielding, and L. Masinter, “RFC 3986: Uniform Resource Identifier (URI): Generic Syntax.” The Internet Society, Jan-2005.
[15] T. Berners-Lee, L. Masinter, and M. McCahill, “RFC 1738: Uniform Resource Locators (URL).” Internet Engineering Task Force, Dec-1994.
[16] “GNU Wget 1.13.4 Manual.pdf - Google Drive.” [Online]. Available: https://docs.google.com/file/d/0B4rrYgKfVJpQeHhhRGZQQUdFV0U/edit?pli=1. [Accessed: 31-Dec-2014].
[17] “Umleitungen › Shell › Wiki › ubuntuusers.de,” 18-Jun-2014. [Online]. Available: http://wiki.ubuntuusers.de/Shell/Umleitungen. [Accessed: 09-Feb-2015].
[18] N. Freed and N. S. Borenstein, “RFC 2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types.” Internet Engineering Task Force, Nov-1996.
[19] “language-detection - Language Detection Library for Java - Google Project Hosting.” [Online]. Available: https://code.google.com/p/language-detection/. [Accessed: 11-Jan-2015].

[20] “UTF8 ⇔ Bin, Hex, & b64 - Nathan Lex on the Internet.” [Online]. Available: https://sites.google.com/site/nathanlexwww/tools/utf8-convert. [Accessed: 18-Jan-2015].
[21] “Complete Character List for UTF-8.” [Online]. Available: http://www.fileformat.info/info/charset/UTF-8/list.htm. [Accessed: 18-Jan-2015].
[22] “jsoup Java HTML Parser, with best of DOM, CSS, and jquery.” [Online]. Available: http://jsoup.org/. [Accessed: 11-Jan-2015].
[23] “Operator Manual - RapidMiner Documentation.” [Online]. Available: http://docs.rapidminer.com/studio/operators/. [Accessed: 02-Feb-2015].

Anhang

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	96,44	95,33	96,63	90,94	96,65	89,87
Class-Recall positiv (in %):	38,17	72,39	44,22	77,96	40,58	83,38
Class-Precision negativ (in %):	60,91	77,52	56,03	75,22	54,09	79,66
Class-Precision positiv (in %):	91,48	93,94	94,69	92,13	94,36	91,91
Accuracy (in %):	67,29 ± 2,32	83,85 ± 2,83	66,44 ± 2,46	83,46 ± 1,85	64,14 ± 2,65	86,10 ± 1,75

Tabelle 18: Übersicht der Performance in RapidMiner für:
Operator: Naive Bayes (Kernel)
Parameter: Laplace correction = on (selbe Performance, wie off)
Estimation mode = greedy (Standardwert)

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	74,49	82,83	81,79	79,67	82,12	77,44
Class-Recall positiv (in %):	86,29	93,81	85,97	87,53	85,11	87,53
Class-Precision negativ (in %):	84,45	93,04	81,09	82,45	79,98	81,82
Class-Precision positiv (in %):	77,20	84,54	86,52	85,41	86,79	84,27
Accuracy (in %):	80,40 ± 2,61	88,32 ± 1,43	84,20 ± 1,97	84,20 ± 2,08	83,86 ± 2,10	83,29 ± 1,68

Tabelle 19: Übersicht der Performance in RapidMiner für:
Operator: Naive Bayes
Parameter: Laplace correction = on (selbe Performance, wie off)

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	76,65	90,89	48,46	89,88	k. A.	87,48
Class-Recall positiv (in %):	97,46	91,45	98,94	94,97	k. A.	95,71
Class-Precision negativ (in %):	96,79	91,45	97,10	92,93	k. A.	93,65
Class-Precision positiv (in %):	80,67	90,89	72,29	92,73	k. A.	91,34
Accuracy (in %):	87,06 ± 2,28	91,17 ± 1,69	77,54 ± 1,41	92,81 ± 1,17	k. A.	92,25 ± 1,41

Tabelle 20: Übersicht der Performance in RapidMiner für:
Operator: W-ComplementNaiveBayes (aus der Weka-Extension)
Parameter: N = off (Standardwert)

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	72,73	87,59	45,09	76,01	k. A.	72,37
Class-Recall positiv (in %):	97,45	94,15	99,01	97,66	k. A.	97,85
Class-Precision negativ (in %):	96,64	93,77	97,10	95,99	k. A.	96,07
Class-Precision positiv (in %):	77,96	88,31	71,02	84,70	k. A.	83,02
Accuracy (in %):	85,03 ± 2,79	90,86 ± 1,29	76,15 ± 1,38	88,49 ± 1,78	k. A.	87,15 ± 1,92

Tabelle 21: Übersicht der Performance in RapidMiner für:
Operator: W-ComplementNaiveBayes (aus der Weka-Extension)
Parameter: N = on

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	80,71	91,16	28,61	86,71	k. A.	82,70
Class-Recall positiv (in %):	97,46	91,88	99,65	96,31	k. A.	97,44
Class-Precision negativ (in %):	96,95	91,81	98,34	94,54	k. A.	95,90
Class-Precision positiv (in %):	83,48	91,23	65,49	90,78	k. A.	88,60
Accuracy (in %):	89,09 ± 0,25	91,52 ± 1,52	69,54 ± 1,76	92,24 ± 1,26	k. A.	91,24 ± 1,76

Tabelle 22: Übersicht der Performance in RapidMiner für:
Operator: W-NaiveBayesMultinomial (aus der Weka-Extension)

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	78,66	97,26	k. A.	95,86	k. A.	88,34
Class-Recall positiv (in %):	90,36	32,89	k. A.	36,71	k. A.	44,11
Class-Precision negativ (in %):	89,07	59,15	k. A.	52,70	k. A.	53,38
Class-Precision positiv (in %):	80,91	92,31	k. A.	92,34	k. A.	83,93
Accuracy (in %):	84,51 ± 2,29	65,06 ± 2,98	k. A.	61,78 ± 1,90	k. A.	62,69 ± 1,60

Tabelle 23: Übersicht der Performance in RapidMiner für:
Operator: k-NN
Parameter: k = 1

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	78,66	97,26	k. A.	99,33	k. A.	88,34
Class-Recall positiv (in %):	90,36	32,89	k. A.	22,18	k. A.	44,11
Class-Precision negativ (in %):	89,07	59,15	k. A.	48,43	k. A.	53,38
Class-Precision positiv (in %):	80,91	92,31	k. A.	97,81	k. A.	83,93
Accuracy (in %):	84,51 ± 2,29	65,06 ± 2,98	k. A.	54,88 ± 2,37	k. A.	62,69 ± 1,60

Tabelle 24: Übersicht der Performance in RapidMiner für:
Operator: k-NN
Parameter: k = 2

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	81,91	99,09	k. A.	90,66	k. A.	13,00
Class-Recall positiv (in %):	93,30	21,22	k. A.	29,98	k. A.	97,92
Class-Precision negativ (in %):	92,43	55,68	k. A.	48,78	k. A.	81,93
Class-Precision positiv (in %):	83,77	95,87	k. A.	81,35	k. A.	60,84
Accuracy (in %):	87,61 ± 2,17	60,13 ± 2,75	k. A.	55,70 ± 3,79	k. A.	62,25 ± 1,48

Tabelle 25: Übersicht der Performance in RapidMiner für:
Operator: k-NN
Parameter: k = 3

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	83,43	91,46	k. A.	83,53	k. A.	12,62
Class-Recall positiv (in %):	94,82	28,53	k. A.	92,49	k. A.	99,03
Class-Precision negativ (in %):	94,15	56,11	k. A.	89,11	k. A.	90,41
Class-Precision positiv (in %):	85,14	76,99	k. A.	88,41	k. A.	61,01
Accuracy (in %):	89,13 ± 1,90	59,98 ± 2,54	k. A.	88,69 ± 2,47	k. A.	62,73 ± 6,64

Tabelle 26: Übersicht der Performance in RapidMiner für:
Operator: k-NN
Parameter: k = 6

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	84,76	44,21	k. A.	91,81	k. A.	48,66
Class-Recall positiv (in %):	95,74	94,52	k. A.	91,21	k. A.	98,34
Class-Precision negativ (in %):	95,21	88,96	k. A.	88,49	k. A.	95,50
Class-Precision positiv (in %):	86,28	62,91	k. A.	93,80	k. A.	72,56
Accuracy (in %):	90,25 ± 1,61	69,37 ± 10,38	k. A.	91,47 ± 1,89	k. A.	77,47 ± 4,70

Tabelle 27: Übersicht der Performance in RapidMiner für:
Operator: k-NN
Parameter: k = 10

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	84,35	72,05	k. A.	90,37	k. A.	58,70
Class-Recall positiv (in %):	95,63	95,33	k. A.	92,91	k. A.	98,41
Class-Precision negativ (in %):	95,07	93,91	k. A.	90,37	k. A.	96,39
Class-Precision positiv (in %):	85,95	77,35	k. A.	92,91	k. A.	76,69
Accuracy (in %):	90,00 ± 2,14	83,70 ± 2,97	k. A.	91,83 ± 2,33	k. A.	81,73 ± 3,24

Tabelle 28: Übersicht der Performance in RapidMiner für:
Operator: k-NN
Parameter: k = 12

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	84,45	81,00	k. A.	89,50	k. A.	66,73
Class-Recall positiv (in %):	95,74	95,03	k. A.	93,76	k. A.	97,92
Class-Precision negativ (in %):	95,19	94,21	k. A.	91,35	k. A.	95,88
Class-Precision positiv (in %):	86,04	83,35	k. A.	92,39	k. A.	80,25
Accuracy (in %):	90,10 ± 1,18	88,01 ± 1,98	k. A.	91,96 ± 1,86	k. A.	84,82 ± 2,46

Tabelle 29: Übersicht der Performance in RapidMiner für:
Operator: k-NN
Parameter: k = 16

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	94,21	92,38	89,02	95,47	k. A.	91,20
Class-Recall positiv (in %):	81,02	86,29	82,14	85,97	k. A.	89,40
Class-Precision negativ (in %):	83,21	87,07	78,57	83,35	k. A.	86,18
Class-Precision positiv (in %):	93,33	91,89	91,04	96,27	k. A.	93,35
Accuracy (in %):	87,61 ± 2,67	89,34 ± 2,81	85,06 ± 2,64	90,00 ± 2,08	k. A.	90,16 ± 1,61

Tabelle 30: Übersicht der Performance in RapidMiner für:
Operator: Logistic Regression
Parameter: kernel type = dot (Standardwert)

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	20,73	63,92	8,77	12,62	k. A.	10,61
Class-Recall positiv (in %):	80,30	7,21	94,47	66,12	k. A.	75,35
Class-Precision negativ (in %):	51,26	40,76	53,85	21,51	k. A.	23,77
Class-Precision positiv (in %):	50,35	16,67	58,46	50,71	k. A.	53,78
Accuracy (in %):	50,53 ± 2,61	35,56 ± 17,47	58,15 ± 1,57	43,45 ± 2,46	k. A.	48,15 ± 2,81

Tabelle 31: Übersicht der Performance in RapidMiner für:
Operator: Logistic Regression
Parameter: kernel type = multiquadric

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	73,48	67,78	54,14	77,65	k. A.	47,99
Class-Recall positiv (in %):	80,00	93,10	81,72	92,27	k. A.	94,46
Class-Precision negativ (in %):	78,59	90,75	68,54	88,09	k. A.	86,25
Class-Precision positiv (in %):	75,12	74,31	70,78	84,88	k. A.	71,46
Accuracy (in %):	76,74 ± 3,54	80,44 ± 9,50	70,03 ± 2,00	86,08 ± 2,14	k. A.	74,94 ± 6,21

Tabelle 32: Übersicht der Performance in RapidMiner für:
Operator: Logistic Regression
Parameter: kernel type = dot (Standardwert)

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	63,11	5,08	52,50	17,73	k. A.	30,78
Class-Recall positiv (in %):	68,53	99,39	76,05	95,96	k. A.	93,14
Class-Precision negativ (in %):	66,70	89,29	61,72	76,35	k. A.	76,48
Class-Precision positiv (in %):	65,03	51,28	68,52	61,32	k. A.	65,01
Accuracy (in %):	65,82 ± 3,66	52,26 ± 0,96	66,07 ± 2,18	62,80 ± 8,65	k. A.	66,95 ± 5,31

Tabelle 33: Übersicht der Performance in RapidMiner für:
Operator: Logistic Regression
Parameter: kernel type = polynomial mit degree = 2 (Standardwert)

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	63,11	5,08	52,50	17,73	k. A.	30,78
Class-Recall positiv (in %):	68,53	99,39	76,05	95,96	k. A.	93,14
Class-Precision negativ (in %):	66,70	89,29	61,72	76,35	k. A.	76,48
Class-Precision positiv (in %):	65,03	51,28	68,52	61,32	k. A.	65,01
Accuracy (in %):	65,82 ± 3,66	52,26 ± 0,96	66,07 ± 2,18	62,80 ± 8,65	k. A.	66,95 ± 5,31

Tabelle 34: Übersicht der Performance in RapidMiner für:
Operator: Logistic Regression
Parameter: kernel type = polynomial mit degree = 3

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	91,57	91,87	k. A.	89,69	k. A.	90,63
Class-Recall positiv (in %):	91,98	91,37	k. A.	95,25	k. A.	94,60
Class-Precision negativ (in %):	91,94	91,41	k. A.	93,29	k. A.	92,40
Class-Precision positiv (in %):	91,61	91,84	k. A.	92,63	k. A.	93,31
Accuracy (in %):	91,77 ± 2,66	91,62 ± 1,44	k. A.	92,89 ± 1,13	k. A.	92,93 ± 1,05

Tabelle 35: Übersicht der Performance in RapidMiner für:
Operator: W-BayesianLogisticRegression (aus der Weka-Extension)

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	k. A.	74,80	k. A.	84,48	k. A.	85,47
Class-Recall positiv (in %):	k. A.	78,38	k. A.	86,17	k. A.	87,67
Class-Precision negativ (in %):	k. A.	77,56	k. A.	81,82	k. A.	83,40
Class-Precision positiv (in %):	k. A.	75,69	k. A.	88,28	k. A.	89,28
Accuracy (in %):	k. A.	76,59 ± 1,75	k. A.	85,45 ± 1,22	k. A.	86,75 ± 1,76

Tabelle 36: Übersicht der Performance in RapidMiner für:
Operator: W-Logistic (aus der Weka-Extension)

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	k. A.	95,33	k. A.	96,92	k. A.	94,84
Class-Recall positiv (in %):	k. A.	90,96	k. A.	92,70	k. A.	92,59
Class-Precision negativ (in %):	k. A.	91,33	k. A.	90,71	k. A.	90,26
Class-Precision positiv (in %):	k. A.	95,12	k. A.	97,61	k. A.	96,12
Accuracy (in %):	k. A.	93,14 ± 2,25	k. A.	94,49 ± 1,02	k. A.	93,53 ± 1,99

Tabelle 37: Übersicht der Performance in RapidMiner für:
Operator: W-SimpleLogistic (aus der Weka-Extension)
oder W-LogistcBase (aus der Weka-Extension)

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	94,41	97,26	31,50	95,47	34,99	91,68
Class-Recall positiv (in %):	80,10	78,78	92,63	87,81	83,66	89,47
Class-Precision negativ (in %):	82,58	82,08	75,87	85,21	60,80	86,32
Class-Precision positiv (in %):	93,48	96,64	64,77	96,35	63,98	93,69
Accuracy (in %):	87,25 ± 2,34	88,02 ± 3,53	66,72 ± 1,91	91,06 ± 2,07	63,21 ± 2,36	90,40 ± 1,58

Tabelle 38: Übersicht der Performance in RapidMiner für:
Operator: Support Vector Machine
Parameter: kernel type = dot (Standardwert)

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	49,80	49,80	0,00	0,00	k. A.	0,00
Class-Recall positiv (in %):	49,75	49,75	100,00	100,00	k. A.	100,00
Class-Precision negativ (in %):	49,75	49,75	0,00	0,00	k. A.	0,00
Class-Precision positiv (in %):	49,80	49,80	57,62	57,62	k. A.	57,99
Accuracy (in %):	49,77 ± 0,08	49,77 ± 0,08	57,62 ± 0,18	57,62 ± 0,18	k. A.	57,99 ± 0,20

Tabelle 39: Übersicht der Performance in RapidMiner für:
Operator: Support Vector Machine
Parameter: kernel type = multiquadric

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	44,11	90,96	0,00	25,53	k. A.	19,31
Class-Recall positiv (in %):	58,58	87,51	100,00	97,73	k. A.	97,92
Class-Precision negativ (in %):	51,48	87,92	0,00	89,23	k. A.	87,07
Class-Precision positiv (in %):	51,15	90,64	57,62	64,08	k. A.	62,62
Accuracy (in %):	51,29 ± 2,56	89,24 ± 2,54	57,62 ± 0,18	67,13 ± 2,01	k. A.	64,90 ± 1,92

Tabelle 40: Übersicht der Performance in RapidMiner für:
Operator: Support Vector Machine
Parameter: kernel type = polynomial mit degree = 2

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	49,80	64,74	0,00	0,00	0,00	0,00
Class-Recall positiv (in %):	49,75	96,75	100,00	100,00	100,00	100,00
Class-Precision negativ (in %):	49,75	95,22	0,00	0,00	0,00	0,00
Class-Precision positiv (in %):	49,80	73,31	57,62	57,62	57,99	57,99
Accuracy (in %):	49,77 ± 0,08	80,75 ± 1,78	57,52 ± 0,18	57,52 ± 0,18	57,99 ± 0,20	57,99 ± 0,20

Tabelle 41: Übersicht der Performance in RapidMiner für:
Operator: Support Vector Machine
Parameter: kernel type = polynomial mit degree = 3

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	98,78	96,34	98,27	97,69	97,80	97,51
Class-Recall positiv (in %):	89,24	90,56	93,05	93,83	92,31	93,70
Class-Precision negativ (in %):	90,17	91,07	91,23	92,10	90,21	91,81
Class-Precision positiv (in %):	98,65	96,12	98,65	98,22	98,30	98,11
Accuracy (in %):	94,01 ± 1,88	93,45 ± 1,97	95,26 ± 0,66	95,47 ± 0,80	94,62 ± 1,67	95,30 ± 1,37

Tabelle 42: Übersicht der Performance in RapidMiner für:
Operator: Support Vector Machine (LibSVM)
Parameter: svm type = C-SVC
kernel type = linear

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	99,59	97,46	95,47	97,78	95,22	98,37
Class-Recall positiv (in %):	70,15	86,19	95,32	90,01	94,25	88,78
Class-Precision negativ (in %):	76,92	87,58	93,76	87,80	92,31	86,40
Class-Precision positiv (in %):	99,42	97,14	96,62	98,22	96,46	98,69
Accuracy (in %):	84,87 ± 2,51	91,82 ± 2,02	95,39 ± 0,87	93,30 ± 0,69	94,66 ± 1,75	92,81 ± 1,52

Tabelle 43: Übersicht der Performance in RapidMiner für:
Operator: Support Vector Machine (LibSVM)
Parameter: svm type = C-SVC
kernel type = polynomial mit degree = 2

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	99,29	98,17	3,37	97,59	3,06	97,51
Class-Recall positiv (in %):	50,56	78,98	99,72	86,39	99,79	84,97
Class-Precision negativ (in %):	66,73	82,35	89,74	84,07	91,43	82,46
Class-Precision positiv (in %):	98,61	97,74	58,38	97,99	58,70	97,92
Accuracy (in %):	74,91 ± 2,52	88,57 ± 2,05	58,88 ± 0,75	91,14 ± 1,00	59,16 ± 0,70	90,24 ± 1,82

Tabelle 44: Übersicht der Performance in RapidMiner für:
Operator: Support Vector Machine (LibSVM)
Parameter: svm type = C-SVC
kernel type = polynomial mit degree = 3

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	49,80	49,80	0,00	0,00	0,00	0,00
Class-Recall positiv (in %):	61,62	79,19	100,00	100,00	100,00	100,00
Class-Precision negativ (in %):	56,45	70,50	0,00	0,00	0,00	0,00
Class-Precision positiv (in %):	55,13	61,22	57,62	57,62	57,99	57,99
Accuracy (in %):	55,71 ± 6,03	64,49 ± 14,92	57,62 ± 0,18	57,62 ± 0,18	57,99 ± 0,20	57,99 ± 0,20

Tabelle 45: Übersicht der Performance in RapidMiner für:
Operator: Support Vector Machine (LibSVM)
Parameter: svm type = C-SVC
kernel type = rbf

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	98,78	97,05	98,07	97,69	97,80	97,51
Class-Recall positiv (in %):	89,24	91,07	93,20	93,83	92,31	93,70
Class-Precision negativ (in %):	90,17	91,56	91,38	92,10	90,21	91,81
Class-Precision positiv (in %):	98,65	96,87	98,50	98,22	98,30	98,11
Accuracy (in %):	94,01 ± 1,88	94,06 ± 1,09	95,26 ± 1,24	95,47 ± 0,80	94,62 ± 1,67	95,30 ± 1,37

Tabelle 46: Übersicht der Performance in RapidMiner für:
Operator: Support Vector Machine (LibSVM)
Parameter: svm type = C-SVC
kernel type = sigmoid

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	99,49	98,98	98,75	98,75	98,57	99,14
Class-Recall positiv (in %):	84,97	84,77	90,50	89,23	89,27	88,50
Class-Precision negativ (in %):	86,87	86,65	88,44	87,09	86,93	86,20
Class-Precision positiv (in %):	99,41	98,82	98,99	98,98	98,85	99,30
Accuracy (in %):	92,23 ± 2,64	91,87 ± 2,12	94,00 ± 0,66	93,26 ± 0,95	93,17 ± 1,64	92,97 ± 2,13

Tabelle 47: Übersicht der Performance in RapidMiner für:
Operator: Support Vector Machine (LibSVM)
Parameter: svm type = nu-SVC
kernel type = linear

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	99,90	100,00	99,52	99,90	k. A.	94,26
Class-Recall positiv (in %):	27,31	66,40	18,43	55,14	k. A.	81,37
Class-Precision negativ (in %):	57,86	74,83	47,30	62,10	k. A.	78,57
Class-Precision positiv (in %):	99,63	100,00	98,11	99,87	k. A.	95,14
Accuracy (in %):	63,59 ± 4,59	83,19 ± 3,75	52,81 ±12,68	74,12 ± 5,99	k. A.	86,79 ± 7,17

Tabelle 48: Übersicht der Performance in RapidMiner für:
Operator: Support Vector Machine (LibSVM)
Parameter: svm type = nu-SVC
kernel type = polynomial mit degree = 2

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	99,49	99,80	98,11	99,90	k. A.	94,93
Class-Recall positiv (in %):	29,14	48,83	46,92	42,88	k. A.	80,19
Class-Precision negativ (in %):	58,38	66,08	55,26	56,27	k. A.	77,64
Class-Precision positiv (in %):	98,29	99,59	85,42	99,83	k. A.	95,62
Accuracy (in %):	64,30 ± 7,06	74,31 ± 6,45	64,81 ±11,40	67,05 ± 6,08	k. A.	86,39 ± 7,52

Tabelle 49: Übersicht der Performance in RapidMiner für:
Operator: Support Vector Machine (LibSVM)
Parameter: svm type = nu-SVC
kernel type = polynomial mit degree = 3

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	99,59	99,09	98,84	98,75	98,95	99,14
Class-Recall positiv (in %):	83,96	84,57	89,23	89,30	87,74	88,43
Class-Precision negativ (in %):	86,12	86,51	87,10	87,16	85,40	86,13
Class-Precision positiv (in %):	99,52	98,93	99,06	98,98	99,14	99,30
Accuracy (in %):	91,77 ± 2,76	91,82 ± 1,94	93,30 ± 0,62	93,30 ± 1,05	92,45 ± 1,69	92,93 ± 2,08

Tabelle 50: Übersicht der Performance in RapidMiner für:
Operator: Support Vector Machine (LibSVM)
Parameter: svm type = nu-SVC
kernel type = rbf

Verwendete Dokumente:	**Level-0-Download**		**Level-1-Download**		**Level-2-Download**	
Prune unter 3 % und über 80 %:	Nein	Ja	Nein	Ja	Nein	Ja
Class-Recall negativ (in %):	99,49	98,98	98,94	98,94	97,51	98,95
Class-Recall positiv (in %):	84,57	85,08	88,38	88,66	85,46	88,64
Class-Precision negativ (in %):	86,56	86,89	86,23	86,52	82,93	86,32
Class-Precision positiv (in %):	99,40	98,82	99,13	99,13	97,94	99,15
Accuracy (in %):	92,03 ± 2,60	92,03 ± 2,25	92,85 ± 3,31	93,02 ± 1,31	90,52 ± 4,00	92,97 ± 2,13

Tabelle 51: Übersicht der Performance in RapidMiner für:
Operator: Support Vector Machine (LibSVM)
Parameter: svm type = nu-SVC
kernel type = sigmoid

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Leipzig, 10. Februar 2015 _____