Kategorisierung von Beiträgen in Onlineforen

Martin Öhlinger

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Januar 2020

Betreuung:

FH-Prof. Mag. DI Dr. Andreas Stöckl

Erklärung

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

Hagenberg, am 24. Januar 2020

Martin Öhlinger

Inhaltsverzeichnis

Kurzfassung

Diese Masterarbeit befasst sich mit dem Einsatz von Text-Klassifikation zur Kategorisierung von Beiträgen in einem Forum. Als Datenbasis wurden dabei Beiträge aus 10 unterschiedlichen Themengebieten von der Plattform *Reddit* gewählt. Im Speziellen liegt das Ziel dieser Arbeit, bei der Verbesserung klassischer Text-Modelle wie *Bag-of-Words*, durch die Kombination mit Features, welche sich aus der Struktur der Beiträge ableiten lassen. Um den tatsächlichen Nutzen von diesen szenario-spezifischen Features zu messen, werden diese auch im direkten Vergleich mit gängigen Techniken wie *Part-of-Speech Tagging* getestet. Die Arbeit geht dabei speziell auf die unterschiedlichen Arten des *Feature Engineerings* ein und erläutert auch die Ansätze hinter Text-Klassifikation.

Das Training zu den Beiträgen erfolgt nach zwei Varianten. Bei der Ersten werden alle Kommentare eines Beitrags zusammengeführt und als ein Textdokument klassifiziert. Bei der zweiten Variante werden die Kommentare einzeln klassifiziert. Das Endergebnis wird dabei durch *Soft Voting* bestimmt. Diese zweite Variante liefert bei der Auswertung aber sehr schlechte Ergebnisse mit dem besten F_1-Wert bei 13%.

Beim Training mit zusammengeführten Kommentaren werden jedoch sehr gute Ergebnisse erzielt. Diese liegen im Durchschnitt alleine mit Text-Modellen bei einem F_1-Wert von 78%. Der höchste F_1-Wert von 99% wird dabei durch eine *Support Vector Machine* mit dem Text-Modell *TF-IDF* in Kombination mit 10 Features, welche sich aus dem Benutzerverhalten ableiten lassen, erreicht. Dieses Ergebnis unterstützt zwar die Theorie, dass szenario-spezifischen Features Text-Modelle verbessern können, jedoch sollten in diesem Fall weitere Daten gesammelt werden, um eine Überanpassung der Methode ausschließen zu können.

Abstract

The main focus of this thesis lies on the classification of forum posts by combining the usage of standard text classification techniques like bag of words and forum specific features. As our training data, forum posts from 10 different topics across reddit have been chosen. To evaluate the impact of forum specific features we also use techniques like part-of-speech tagging to test against them. Throughout this thesis we will also take a closer look at the workflow regarding feature engineering and commonly used methods for classification like naïve bayes.

Two different approaches for classifying the forum posts will be tested. For the first one, all comments belonging to one post will be combined into one text document and classified as such. For the second approach the comments will be classified separately. In this case we are using soft voting to evaluate the topic of the original post. After the implementation and training however, it turns out that this approach does not work at all. The best F_1 score we can get is around 13%.

Fortunately, with the first approach, where we combine the comments into one document, our results are much better. The overall F_1 score for using only text models is already around 78%. By using a support vector machine and combining the TF-IDF model with 10 features related to the user activity in the forum, we can get an F_1 score of 99%. On the one hand this proves that forum specific features can increase the performance of standard text classification techniques but on the other hand, more data should be collected to prove that this is not just a case of overfitting our model.

Kapitel 1

Einleitung

„Sie haben gewonnen!“ ist ein Satz den wahrscheinlich jeder schon einmal gelesen hat, der eine E-Mail-Adresse besitzt. Leider handelt es sich dabei aber eher um Spam als den erhofften Geldsegen. Mussten diese und ähnliche Spam-E-Mails früher händisch aus dem Postfach entfernt werden, so wird diese Aufgabe heutzutage automatisch ausgeführt. Text-Klassifikation spielt dabei eine wesentliche Rolle. Diese wurde bereits 1998 eingesetzt, um Spam-E-Mails zu erkennen [29] und findet auch heute noch Anwendung.

Der Einsatz von Text-Klassifikation hat sich in den letzten Jahren noch um ein Vielfaches gesteigert und ist aus vielen Bereichen der Softwareindustrie nicht mehr wegzudenken. Oft ist dieser so unscheinbar, dass er den meisten Nutzern gar nicht bewusst ist. Text-Klassifikation wird zur Stimmungsanalyse in sozialen Medien, für Chatbots, zur Erkennung von Sprache oder Kategorisierung von Dokumenten eingesetzt, um nur eine Handvoll Beispiele zu nennen.

Diese Arbeit bezieht sich speziell auf den Einsatz von Text-Klassifikation in einem Forum. Obwohl die Bedeutung von Foren in den letzten Jahren aufgrund von anderen sozialen Medien wie Facebook stark abgenommen hat, so sind sie immer noch ein wichtiger Bestandteil der Internetlandschaft. Dieser Umstand beruht darauf, dass Foren gerade von Personengruppen mit sehr ähnlichen Interessen genutzt werden, um sich mit Gleichgesinnten auszutauschen. Die Gemeinschaft besteht meist schon über Jahre und würde vom Wechsel zu anderen Plattformen nicht profitieren. So verbleiben die Nutzer in den Foren und es wird über aktuelle Entwicklungen der Automobilbranche oder die besten Angelplätze diskutiert.

Damit hat sich in den letzten Jahrzehnten sehr viel Wissen in diesen Foren angesammelt. Es würde sich hier also anbieten Text-Klassifikation einzusetzen um diese Informationen besser zugänglich zu machen. Dabei steht man jedoch vor dem Problem, dass Text-Klassifikation in der Regel auch viel Text benötigt um zufriedenstellende Ergebnisse zu liefern. Da der Titel eines Beitrags meist sehr kurzgehalten ist, soll eine Mischung von Eigenschaften des Forums und Kommentaren zum Beitrag zu besseren Ergebnissen führen.

1.1 Zielsetzung

Ziel dieser Arbeit ist es somit, Beiträge in einem Forum anhand ihrer Kommentare und Eigenschaften des Forums einem Themengebiet zuzuordnen bzw. zu klassifizieren. Als Datenbasis wurden dabei Beiträge aus 10 unterschiedlichen Themengebieten von der Plattform *Reddit*[1] gewählt. Die Themengebiete auf Reddit können dabei mit einem klassischen Forum verglichen werden.

Um einen Beitrag zu klassifizieren werden zwei Varianten getestet. In der ersten Variante werden die Kommentare eines Beitrags zu einem Textdokument zusammengeführt. Somit wird der Beitrag, wie auch bei der Klassifizierung von E-Mails, als ganzes kategorisiert. Bei der zweiten Variante werden die Kommentare nicht zusammengeführt sondern einzeln ausgewertet. Es bestimmt somit die Summe der Einzelergebnisse die Kategorie des Beitrags. Es wird nun die Annahme getroffen, dass durch das Miteinbeziehen von Eigenschaften des Forums, das Ergebnis dieser beiden Varianten verbessert wird. Um den Einfluss von Forum Eigenschaften zu messen, werden deswegen beide Varianten zu Beginn nur mit klassischen Text-Modellen wie *Bag-of-Words* und unterschiedlichen Methoden zur Text-Klassifizierung getestet.

Zusätzlich werden noch weitere, gängigere Techniken eingesetzt um die Klassifizierung der Beiträge zu verbessern. Dadurch soll ersichtlich werden ob sich die Eigenschaften des Forums auch im Vergleich mit diesen durchsetzen.

1.2 Herausforderungen

Eine Herausforderung bei der Umsetzung von Projekten im Bereich Text-Klassifikation oder maschinellem Lernen im Allgemeinen, sind häufig die Menge an der zur Verfügung stehenden Daten. Dieses Problem wurde durch die Auswahl von Reddit als Datenbasis entschärft, da es hier in gewisser Weise kein Limit an Daten gibt. Die Menge an Beiträgen für dieses Projekt konnte somit frei gewählt werden.

Aber nicht nur die Menge an Daten kann eine Herausforderung darstellen. Ebenso ausschlaggebend für den Erfolg ist die gewählte Methode zur Klassifizierung. Diese hat nicht nur Einfluss auf das Ergebnis, sondern auch auf die Laufzeit. Eine lange Laufzeit spielt für dieses Projekt zwar keine allzu große Rolle und kann auch im Zusammenhang mit der Menge an eingesetzten *Features* stehen, sie sollte jedoch immer im Auge behalten werden. Da es sich vorab nicht sagen lässt welche Methode am besten für diese Arbeit geeignet ist, werden die Ergebnisse von fünf Methoden verglichen.

1.3 Struktur

Um einen guten Überblick über die allgemeine Thematik zu bekommen werden im nächsten Kapitel Begrifflichkeiten im Zusammenhang mit Text-Klassifikation genauer erläutert. Da das gesamte Thema sehr von englischen Begriffen dominiert wird, macht es in Ausnahmefällen mehr Sinn den englischen Begriff (z. B. *Feature*) zu verwenden. Wird die deutsche Übersetzung verwendet, findet sich der englische Begriff in Klammern daneben, da dieser meist geläufiger ist.

[1] https://reddit.com

In Kapitel 3 wird ein kurzer Überblick über die Errungenschaften der letzten Jahre im Bereich Text-Klassifikation gegeben. Es wird dann speziell auf den Einsatz von Text-Klassifikation in sozialen Medien und damit auch Foren eingegangen.

Kapitel 4 beschäftigt sich im Detail mit dem *Feature Engineering* Prozess. Dabei werden die wichtigsten Bereiche von der Textaufbereitung über unterschiedliche *Feature Engineering* Techniken angesprochen, welche in dieser Arbeit auch Einsatz finden. Zusätzlich wird auch ein Blick auf die Techniken, welche bei der Auswahl von *Features* angewendet werden, geworfen. In Kapitel 5 werden die fünf Methoden zur Text-Klassifizierung, welche für das Training mit den Beiträgen genutzt werden, im Detail beschrieben.

In Kapitel 6 wird der gesamte Implementierungsprozess mit Auszügen aus der Umsetzung erklärt. Dabei wird besonders Wert auf die Features und der Implementierung der zwei Varianten zur Klassifikation von Beiträgen gelegt. Die Resultate aus der Implementierung werden im darauffolgenden Kapitel 7 präsentiert.

Am Ende der Arbeit findet sich noch einmal eine Zusammenfassung über die Erkenntnisse, zu denen die Ergebnisse der Implementierung geführt haben und welche Probleme dabei aufgetreten sind. Außerdem wird auf mögliche Verbesserungen im Bezug auf die Implementierung eingegangen. Im Anhang A finden sich zusätzliche Ergebnisse, welche während der Implementierung entstanden sind aber aus Gründen der besseren Übersicht nicht in Kapitel 7 gezeigt wurden.

Kapitel 2

Technischer Hintergrund

Als Einleitung zum Thema maschinelles Lernen und damit auch Text-Klassifikation, werden zu Beginn dieser Arbeit häufig verwendete Begrifflichkeiten in diesem Bereich erklärt. Im ersten Abschnitt wird auf den Unterschied zwischen überwachtem und unüberwachtem Lernen eingegangen und welcher dieser beiden für diese Arbeit zum Einsatz kommt. Danach werden *Features* anhand eines Beispiels und die Bedeutung von *Feature-Engineering* erklärt. In den darauffolgenden Abschnitten wird Computerlinguistik und Text-Klassifikation selbst erläutert. Zum Schluss wird noch eine Übersicht über die Evaluierungsmethoden, für den späteren Vergleich der Ergebnisse, gegeben.

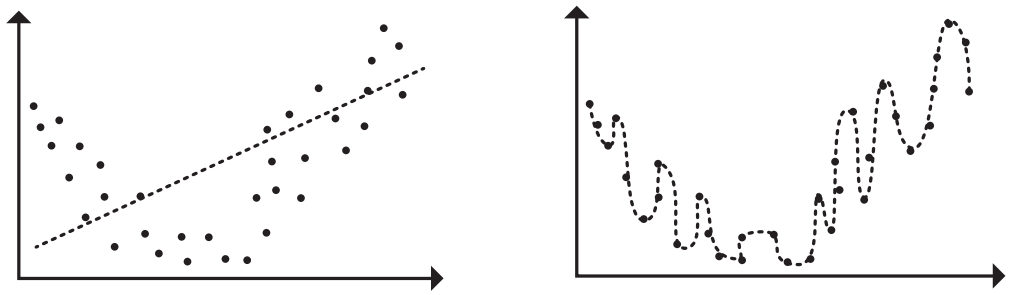
2.1 Maschinelles Lernen

Maschinelles Lernen (engl. *machine learning*) ist eine Form der künstlichen Intelligenz, welche das Extrahieren von Informationen aus Daten zur Erkennung von Mustern beschreibt. Ein Algorithmus *lernt* dabei diese Muster anhand von Beispieldaten und kann diese im nächsten Schritt verallgemeinern [43]. Mit diesem Wissen können *Prognosen* zu neuen unbekannten Daten erstellt werden.

Es gibt zwei Arten des maschinellen Lernens, welche in der Praxis am häufigsten zum Einsatz kommen. Überwachtes oder unüberwachtes Lernen. Die Verwendung einer dieser Beiden ist immer von der Art der Anwendung oder den vorhandenen Daten abhängig.

2.1.1 Überwachtes Lernen

Bei überwachtem Lernen (engl. *supervised learning*) stehen bereits Daten mit dem gewünschten Endergebnis zur Verfügung. Diese Daten werden im ersten Schritt in *Trainings-* und *Testdaten* unterteilt, wobei die Trainingsdaten zum Trainieren eines Algorithmus und die Testdaten zur Überprüfung des Trainings genutzt werden. In der Regel besteht das Trainingsset aus 80% der Ausgangsdaten. Die restlichen 20% werden in das Testset aufgenommen. Beide Datensets sollten eine ähnliche Verteilung der Klassen besitzen um optimale Ergebnisse zu erzielen. Gibt es bereits in den Ausgangsdaten einen starken Unterschied in der Verteilung von Klassen spricht man von *unausgeglichenen* Daten [49].



(a) (b)

Abbildung 2.1: (a) Unter- bzw. (b) Überanpassung.

Das Hauptziel bei überwachtem Lernen ist, anhand der vorgegebenen Trainingsdaten möglichst gut zu *generalisieren* um genaue Prognosen mit den Testdaten und in späterer Folge, unbekannten Daten zu erstellen. Wird dieses Ziel nicht erreicht spricht man entweder von Unteranpassung (engl. *underfitting*) oder Überanpassung (engl. *overfitting*). Diese Probleme werden in Abbildung 2.1 visualisiert.

Grund für eine Unteranpassung sind meist zu wenige Daten. Der Algorithmus kann weder mit dem Trainingsset noch mit den Testdaten genaue Prognosen erstellen. Bei der Überanpassung wird der Algorithmus zu stark auf das Trainingsset abgestimmt. Prognosen für das Trainingsset sind sehr gut, jedoch für die Testdaten unbrauchbar. Anwendungsfälle für überwachtes Lernen sind Klassifikation und Regression.

Klassifikation

Bei der Klassifikation wird ein Algorithmus anhand einer vorgegebenen Zuordnung von Objekten zu Klassen trainiert. Ein Objekt wäre im Zusammenhang mit dieser Arbeit ein Textdokument und seine Klasse „Sportbeitrag“. Die Eigenschaften eines Objekts werden *Features* genannt. Mit Features werden Methoden zur Klassifizierung trainiert um einem Textdokument eine Klasse zuzuordnen. Mit nur einem Feature wird das Ergebnis jedoch nicht sehr gut ausfallen, weshalb in der Regel mehrere Features für ein Textdokument entwickelt werden müssen. Techniken dafür finden sich im nächsten Kapitel.

Bei der Klassifikation gibt es auch eine Unterscheidung zwischen *Binärer* und *Multiklassen* Klassifikation. Bei Binärer Klassifikation stehen nur zwei Klassen zur Auswahl. Darunter würde auch das Beispiel in der Einleitung zur Einteilung von Emails in „Spam“ oder „kein Spam“ fallen. Für diese Arbeit wird versucht Forumsbeiträge in eine von 10 verschiedenen Kategorien einzuteilen, weshalb es sich hier um *Multiklassen* Klassifikation handelt.

Regression

Bei Regression werden im Gegensatz zur Klassifikation keine vordefinierten Klassen sondern ein Wert, also eine Zahl bestimmt. Ein Anwendungsbeispiel für Regression wäre das Abschätzen des Einkommens von Personen anhand ihres Alters und Bildungsgrads.

2.1.2 Unüberwachtes Lernen

Im Gegensatz zu überwachtem Lernen sind bei unüberwachtem Lernen (engl. *unsupervised learning*) keine Daten zum gewünschten Endergebnis vorhanden. Es gibt also keine Trainingsdaten mit welchen ein Algorithmus trainiert werden kann oder Testdaten zur Überprüfung der Performance. Diese Art des Lernens wird vor allem zum Aufzeigen versteckter Themen in Textdokumenten oder zur Clusteranalyse eingesetzt. Obwohl diese Techniken keine Anwendung in dieser Arbeit finden, werden sie aufgrund der Vollständigkeit kurz erklärt.

Themenfindung

Bei der Themenfindung (engl. *topic modeling*) werden große Mengen an Textdokumenten in Themen gegliedert, um Informationen über deren Inhalte zu bekommen. Die Themen können dabei vorab durch eine Gruppierung von Wörtern oder nur durch ihre Anzahl für den Algorithmus definiert werden [51]. Diese Technik ist besonders hilfreich, wenn ein manuelles eruieren von Themen aufgrund der Menge an Textdokumenten unmöglich wird.

Clusteranalyse

Die Clusteranalyse (engl. *clustering*) versucht in einer Menge von Daten, Gruppen bzw. Cluster mit ähnlichen Eigenschaften zu finden. Dabei kann dem Algorithmus angegeben werden wie viele Gruppen gebildet werden sollen. Diese Technik kann, abgesehen von der Gruppierung, dafür eingesetzt werden, um redundante Daten zu bündeln oder um Ausreißer der Gruppen in den Daten zu identifizieren [40].

2.2 Corpus

In den vorherigen Abschnitten wird immer von „Daten“ in keinem bestimmten Kontext gesprochen. In Bezug auf Text-Klassifikation spricht man hier von einem *Corpus*. Ein Corpus ist eine Sammlung von verwandten Dokumenten, welche in einer bestimmten Sprache verfasst wurden [3, S. 19]. Die im Corpus enthaltenen Textdokumente sind je nach Anwendungsfall schon in Kategorien eingeteilt oder nicht.

In dieser Arbeit besteht der Corpus aus rund 860.000 Kommentaren, wobei jeder Kommentar einem von 10 Themen zugeteilt ist. Der Corpus kann damit für überwachtes Lernen eingesetzt werden. Bevor jedoch mit der Klassifizierung begonnen werden kann, benötigt es noch Features.

2.3 Features

Ein Feature repräsentiert eine Eigenschaft bzw. ein Attribut eines Objekts in Form einer Zahl [18]. „Max ist 14 Jahre alt.“ wäre ein Beispiel für ein Feature einer Person. Das Attribut ist „Alter“, der Wert „14“. Mehrere Features zu einem Objekt werden als *Feature Vector* bezeichnet.

Der Corpus dieser Arbeit besteht jedoch aus Textdaten d.h. Buchstaben bzw. Symbolen und kann somit in seiner Ursprungsform nicht zum Trainieren eines Algorithmus eingesetzt werden. Es müssen zuerst Features extrahiert werden, um mit dem Training starten zu können. Dieser Schritt wird *Feature-Engineering* genannt.

1. Max liebt Katzen.
2. Lisa hat 2 Hunde.

Diese Sätze können nun durch mehrere Features beschrieben werden. In Tabelle 2.1 würde eine Zeile einen Feature Vector darstellen, welcher die Features eines Satzes repräsentiert. Dieses Beispiel dient nur als Veranschaulichung wie die Umwandlung von einem Satz zu mehreren Features aussehen könnte. In der Praxis gibt es dafür bessere Techniken.

Tabelle 2.1: Features für Sätze

ID	*Wörter*	*Buchstaben*	*Zahlen*
1	3	14	0
2	3	12	1

2.4 Feature-Engineering

Feature-Engineering beschreibt, wie im vorherigen Abschnitt gezeigtem Beispiel, den Prozess der Umwandlung von rohen Daten eines Textes oder eine andere Form von Daten in Zahlen bzw. Features. Es ist damit der wichtigste Arbeitsschritt und kommt in praktisch jedem Bereich des maschinellen Lernens zum Einsatz. Laut einer Umfrage [46] wenden Datenwissenschaftler 60% ihrer Arbeitszeit für Feature-Engineering auf. Die daraus resultierenden Features bilden die Basis zur Verarbeitung für Algorithmen der jeweiligen Einsatzbereiche. Genaue Techniken des Feature-Engineerings im Bereich von Textdaten werden im Kapitel 4 beschrieben.

Bei der Auswahl von Features muss auch der Algorithmus bzw. die Methode zur Klassifikation in Betracht gezogen werden, da sich auch hier manche Features besser oder schlechter eignen. Die Anzahl der eingesetzten Features spielt eine wichtige Rolle. Zu wenige oder zu viele Features können zu einem schlechteren Ergebnis, verursacht durch Über- oder Unteranpassung, führen [38].

2.5 Computerlinguistik

Computerlinguistik (engl. *natural language processing*) umfasst alle Bereiche bei der sich Computerwissenschaft und natürliche Sprache überschneiden. Einige der wichtigsten Anwendungsgebiete für Computerlinguistik sind [8]

- die Erkennung von Sprache,
- Echtzeitübersetzungen und,
- Text-Klassifikation.

Außerdem beinhaltet dieser Bereich auch Techniken, welche für den Feature-Engineering Prozess für Texte hilfreich sind. Darunter fallen *Bag-of-Words*, *Part-of-Speech Tagging* und auch das automatische Korrigieren von Rechtschreibfehlern.

Computerlinguistik ist bis heute ein sehr aktives Gebiet der Forschung, da es immer noch viele Herausforderungen in diesem Bereich gibt. Sprache ist nicht nur durch seine Regeln bzw. Grammatik definiert, sondern auch in welcher Form sie eingesetzt wird. So kann ein Satz oder ein Wort mehrere Bedeutungen je nach Kontext haben [3, S. 35]. Dieser ergibt sich meist aus dem Umfeld, in dem die Unterhaltung stattfindet. Ist der Kontext unbekannt, so wird es fast unmöglich eine korrekte Interpretation zu finden. Ein weiteres Beispiel, wo auch Text Klassifikation an seine Grenzen kommt, wäre das Erkennen von Sarkasmus.

2.6 Text-Klassifikation

Bei Text-Klassifikation handelt es sich um ein Anwendungsgebiet der Computerlinguistik, welches sich mit der Zuordnung von Texten zu Klassen[1] beschäftigt. Diese Klassen sind dabei vordefinierte Themen, in welche ein Text eingeteilt werden soll. Es ist somit eine Technik, um Informationen zu kategorisieren bzw. zu filtern. Die Art in welcher Text-Klassifikation eingesetzt wird, ist vom gegebenen Szenario abhängig. Unabhängig davon ist der Prozess von den Ausgangsdaten bis zum gewünschten Ergebnis für viele Anwendungsgebiete der Text-Klassifikation sehr ähnlich.

Text-Klassifikation wird in zwei Arbeitsschritte unterteilt. Eine Aufbau- und Durchführungsphase. In der Aufbauphase werden aus dem im Corpus enthaltenen Dokumenten Features extrahiert, welche gemeinsam mit den zugeordneten Klassen zum Trainieren einer Methode zur Klassifizierung genutzt werden.

Der Schritt des Feature-Engineering und trainieren des Algorithmus wird meist öfters wiederholt, um bestmögliche Resultate zu erzielen. Für das optimale Training werden Techniken wie das *Kreuzvalidierungsverfahren* eingesetzt. Bei dieser Technik wird der Corpus bzw. dessen Features in k *Folds* geteilt. Das Training wird nun k mal durchlaufen wobei jedes Mal einer der *Folds* als Testdaten und die restlichen als Trainingsdaten genutzt werden [41]. Damit treten Probleme wie Über- oder Unteranpassung seltener auf. Sind die Ergebnisse zufriedenstellend, kann in die Durchführungsphase übergegangen werden. Hier bekommt der Algorithmus Textdokumente zur Klassifizierung, welche nicht Teil des Corpus sind.

[1] In dieser Arbeit wird dafür auch öfter das Wort „Kategorie“ verwendet.

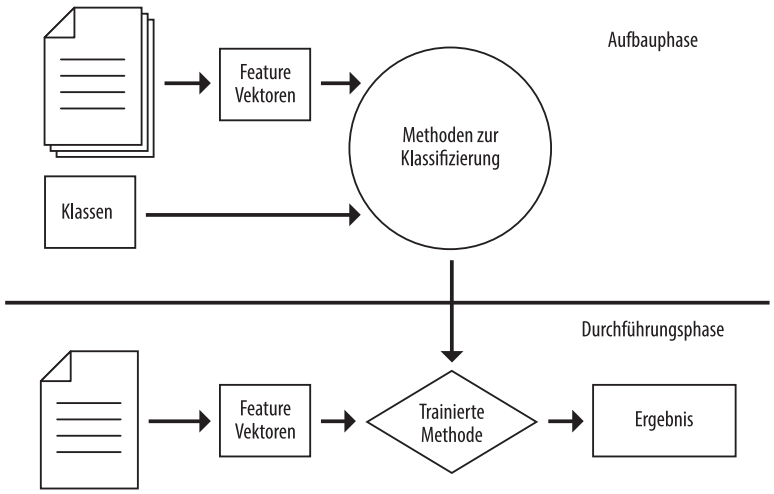


Abbildung 2.2: Ablauf einer Text-Klassifikation. Bildquelle [3]

2.7 Evaluierungsmethoden

Um Features und deren Ergebnisse durch zwei Methoden wie *Naïve Bayes* oder *Gradient Boosting* vergleichen zu können, benötigt es Evaluierungsmethoden. Bei einer Klassifikation lässt sich dies noch relativ einfach umsetzen. Hier werden die Metriken Genauigkeit (engl. *precision*), Treffsicherheit (engl. *recall*) und eine Kombination dieser, der F_1-Wert, als Vergleich herangezogen. Pauschal kann nicht gesagt werden, welcher dieser Metriken die größte Bedeutung hat, da dies vom jeweiligen Einsatzbereich der Klassifikation abhängig ist. Als Berechnungsbasis dieser Werte dient eine Wahrheitsmatrix.

2.7.1 Wahrheitsmatrix

Eine Wahrheitsmatrix (Abbildung 2.2) zeigt die Gegenüberstellung von tatsächlichen zu vorhergesagten Klassen eines Modells. Diese Matrix hat immer eine Größe von $l \times l$, wobei l für die Anzahl der Klassen steht [18]. Es wird aufgrund der Einfachheit hier von zwei Klassen ausgegangen. In der Implementierung findet dasselbe Prinzip nur mit mehreren Klassen Anwendung. *True-Positives* bzw. *Negatives* ist die Anzahl an korrekt getroffenen Vorhersagen des Modells. *False-Negatives* bzw. *Positives* spiegelt die Anzahl an falsch klassifizierten Objekten wieder.

Tabelle 2.2: Wahrheitsmatrix

		Vorhersage	
		Negativ	Positiv
Tatsächlich	Negativ	*True-Negatives*	*False-Positives*
	Positiv	*False-Negatives*	*True-Positives*

2.7.2 Precision

Die *precision* P wird mit der Formel

$$P = \frac{True\text{-}Positives}{True\text{-}Positives + False\text{-}Positives} \tag{2.1}$$

berechnet. Dieser Wert gibt an wie viele Vorhersagen eines Modells auch wirklich korrekt waren. Damit wird ein besonderes Augenmerk auf Klassifizierungen, welche *True-Positive* sind, gelegt. Bei Anwendungen wie Spam-Erkennung haben diese eine besondere Bedeutung. Der Wert sollte in diesem Bereich deshalb besonders hoch sein.

2.7.3 Recall

Im Gegensatz zur *precision* gibt der *recall* R an, wie viele der gesamten Objekte einer Klasse überhaupt korrekt erfasst wurden. Er wird mit der Formel

$$R = \frac{True\text{-}Positives}{True\text{-}Postives + False\text{-}Negatives} \tag{2.2}$$

berechnet. Der *recall* sollte in Anwendungsbereichen, für welche *False-Negative* Klassifizierungen großen Einfluss haben, besonders hoch sein.

2.7.4 F_1-Wert

Der F_1-*Wert* ergibt sich durch die *precision* P und *recall* R eines Modells und wird folgendermaßen berechnet:

$$F_1 = 2 \times \frac{P \times R}{P + R}. \tag{2.3}$$

Er ist damit ein Wert zum Messen der Balance zwischen *precision* und *recall*. Dieser Wert wird deshalb auch in Kapitel 6 und Kapitel 7 als Vergleich von Ergebnissen verwendet.

Kapitel 3

Stand der Technik

Basierend auf den zuvor vorgestellten Begriffen und Verfahren gibt das folgende Kapitel einen Überblick über den aktuellen Stand der Technik, sowie einen kurzen Rückblick über die Geschichte zur Text-Klassifikation. Abschnitt 3.2 geht dann speziell auf die Anwendungsfälle im Bereich sozialer Medien ein.

3.1 Entwicklungen im Bereich Text-Klassifikation

Text-Klassifikation erlebte seinen großen Aufschwung erst mit der Erfindung des Internets. Davor gab es zwar Versuche mit der Klassifizierung von Dokumenten [4, 22], aber zu dieser Zeit steckte die Technologie noch in ihren Kinderschuhen. Die dafür nötige Rechenleistung und die Menge an Daten, welche für solch eine Aufgabe benötigt werden, stand zu diesem Zeitpunkt noch nicht zur Verfügung. Mit dem Internet und den wesentlichen Steigerungen bei der Rechenleistung von Computern, änderte sich dieser Umstand jedoch rasant. Einer der bekanntesten und ersten Anwendungsfälle, welcher durch das Internet entstand, ist die Klassifikation von Spam-E-Mails [29]. Aber auch bei der Einteilung von Nachrichtenbeiträgen in ihre jeweilige Kategorie [21], wurden schnell die Möglichkeiten von Text-Klassifikation ersichtlich. Für diese ersten Anwendungsfälle kamen häufig *Naïve Bayes*, *Nächste-Nachbarn-Klassifikation* oder *Entscheidungsbäume* als Methoden zur Klassifizierung zum Einsatz. Aufgrund dieser neuen Möglichkeiten für Text-Klassifikation wurde viel in diesem Bereich geforscht, um bestehende Methoden zu verbessern oder neue zu finden. Ein gutes Beispiel dafür ist der Einsatz von *Support Vector Machines* (SVMs) [16], mit welchen bessere Ergebnisse, als mit zu diesem Zeitpunkt gängigen Methoden, erzielt werden konnten. *Naïve Bayes* hingegen wurde wiederum verbessert, um mit den Ergebnissen von SVMs mithalten zu können [30]. Aber auch mit der Kombination von mehreren Methoden zur Klassifizierung, genannt *Ensemble*, konnten gute Ergebnisse erreicht werden [36].

Lange Zeit galt der Einsatz von *Support Vector Machines* als beste Methode im Bereich Text-Klassifikation [2]. Mit dem Durchbruch bei der Entwicklung *künstlicher neuronaler Netze* [14] wurden diese jedoch verdrängt. Obwohl *künstliche neuronale Netze* bessere Ergebnisse als herkömmliche Methoden wie SVMs oder *Naïve Bayes* erzielen [19], werden diese immer noch, aufgrund ihrer Einfachheit bei der Umsetzung [32], eingesetzt.

Gleichzeitig zu diesen Entwicklungen bei Methoden zur Text-Klassifikation, gab es auch Weiterentwicklungen im Bereich des Feature Engineerings. Erste Anwendungen wie zum Beispiel das Erkennen von Spam-E-Mails oder die Kategorisierung von Nachrichtenbeiträgen nutzten ein *Bag-of-Words* oder *Bag-of-N-Words* Modell um Features aus den Texten zu generieren. Mit *TF-IDF*, eine weitere Variation zum *Bag-of-Words* Modell, wurden häufig in Kombination mit SVMs sehr gute Ergebnisse [9] erzielt. Mit dem Einsatz von Techniken aus der Computerlinguistik wie *Part-of-Speech Tagging* wurden unterschiedliche Ergebnisse erzielt, welche die Klassifikation in manchen Anwendungsfällen verschlechtern bzw. nicht verändern [25], aber auch speziell bei Stimmungsanalysen, zu Verbesserungen [26] führen können.

Aktuelle Errungenschaften im Bereich Text-Klassifikation haben häufig ihren Ursprung in großen Unternehmen wie Google, Facebook oder Amazon. Diese investieren sehr viel Zeit und Geld in die Entwicklung neuer Technologien, da diese einen wichtigen Faktor in der Wettbewerbsfähigkeit dieser Unternehmen ausmacht. Trotz des unternehmerischen Vorteils werden auch teilweise neue Methoden oder Hilfsmittel, wie zum Beispiel *word2vec* [23] oder *fastText*[1], veröffentlicht. Über Technologien, welche ihren Weg nicht an die Öffentlichkeit finden, kann nur gemutmaßt werden.

Generell sind jedoch die Probleme bei der Text-Klassifikation über die Jahre sehr ähnlich geblieben. Zu wenige Daten um nützliche Features zu finden [20] oder ein Ungleichgewicht in den Kategorien [6] sind Probleme, welche noch heute aktuell sind. Aber auch wenn diese nicht vorhanden sind, kann im Vorhinein nie zu 100% gesagt werden welche Features oder Methoden zur Klassifizierung in welchem Anwendungsfall am besten abschneiden werden. Die Ergebnisse bei der Kombination einer Methode und Features sind immer vom jeweiligen Anwendungsgebiet abhängig. Deswegen werden auch in den meisten Fällen mehrere Varianten getestet und verglichen um die bestmögliche Kombination zu finden.

3.2 Text-Klassifikation in sozialen Medien

Soziale Medien wie Twitter oder Reddit sind ein wichtiger Teil des Internets geworden und ermöglichen es ihren Benutzern mit Personen auf der ganzen Welt vernetzt zu sein und Meinungen auszutauschen. Text-Klassifikation wird hier in erster Linie von den Unternehmen selbst eingesetzt um auf die Benutzer abgestimmte Inhalte oder Werbungen zu zeigen. Aufgrund der Menge an Textdaten, welche in diesem Umfeld generiert werden, sind soziale Medien auch eine beliebte Datenbasis für Forschungszwecke geworden. Dabei wird häufig etwas mehr Zeit in die Aufbereitung der Beiträge investiert, da Benutzer typischerweise nicht sehr viel Wert auf eine korrekte Rechtschreibung legen und der Einsatz von Akronymen weit verbreitet ist. Die allgemeine Annahme, dass dies zu einer Verbesserung bei der Klassifizierung führt, ist jedoch nicht gegeben [24]. Beliebte Forschungsarbeiten im Umfeld von sozialen Medien sind

- Stimmungsanalysen,
- das Erkennen von *Fake News* oder
- das Entfernen von Hasskommentaren.

[1] https://github.com/facebookresearch/fastText

Wie schnell sich neue Anwendungsgebiete im Bereich Text-Klassifikation ergeben, wird gut am Beispiel von *Fake News* ersichtlich. Zu diesem Thema wird erst seit den letzten fünf Jahren intensiv geforscht [1, 13]. Das Konzept von *Fake News* ist auch relativ neu. In einer Publikation von 1970 [7] wurde noch davon ausgegangen, dass in einem Gespräch zum Thema *News* die Frage „Ist diese echt?“ nicht gestellt werden kann.

Bei der Umsetzung von Text-Klassifikation in sozialen Medien erfordert jede Plattform eine eigene Vorgehensweise. Bei Twitter wird häufig eine Form des unüberwachten Lernens eingesetzt. Hier sind Beiträge, auch genannt *Tweets*, von sich aus keiner konkreten Kategorie zugewiesen. Projekte welche Twitter für überwachtes Lernen einsetzen, nutzen entweder angegebene *Hashtags* oder definieren eigene Kategorien. Aus diesem Grund wird meist nur eine geringe Menge an *Tweets* für das Trainieren der Methoden genutzt, da die manuelle Zuordnung von Kategorien sehr zeitaufwendig ist. Außerdem können hier Modelle wie *Bag-of-Words* nur eingeschränkt eingesetzt werden. Dies beruht auf den Umstand, dass bis 2017 die Wörter eines *Tweets* auf nur 140[2] beschränkt waren. Mit dieser geringen Wortanzahl können keine zufriedenstellende Ergebnisse mit *Bag-of-Words* erzielt werden. Aus diesem Grund werden Ansätze wie *Part-of-Speech Tagging* [12] oder szenario-spezifische Features wie das Benutzerverhalten [33] eingesetzt.

Reddit bzw. Foren hingegen eignen sich besonders gut für überwachtes Lernen. Ein Forum baut sich meist durch mehrere Kategorien auf, in welchen ein Benutzer einen Beitrag erstellen kann. Bei Reddit werden diese Kategorien auch *Subreddit* genannt. Damit übernimmt der Benutzer die Zuordnung von einem Beitrag zu seiner Kategorie [37]. Dies kann aber auch zu einem Nachteil werden, wenn Benutzer ihren Beitrag in eine falsche Kategorie einteilen und damit das Trainingsergebnis verfälschen. Im Folgenden Abschnitt wird im Detail auf Arbeiten im Bereich Text-Klassifikation mit Reddit eingegangen. Hierbei werden die eingesetzten Techniken, Ergebnisse aber auch die dabei aufgetretenen Probleme genau beschrieben.

3.2.1 Reddit

Um einen Überblick über den Stand der Technik im Bezug auf Text-Klassifikation im Umfeld von Reddit zu bekommen wurden mehrere Arbeiten untersucht. Dabei lag der Fokus entweder auf der Klassifikation von Beiträgen [11, 15] oder Kommentaren [17, 34]. Im Folgenden wird nun konkret auf die Arbeiten [11, 15] eingegangen, da diese sehr ähnliche Ziele und Vorgehensweisen aufweisen.

In Beiden wurden Beiträge aus 5 - 12 Kategorien bzw. Subreddits gewählt und *Naïve Bayes* mit *Bag-of-N-Words* als Basis für weitere Ergebnisse eingesetzt. Außerdem wurden alle Beiträge, welche aus einem Bild, Video oder Link bestehen ignoriert, da diese nicht für eine Text-Klassifikation in Frage gekommen sind. Für das Feature-Engineering setzten beide Arbeiten auf unterschiedliche Techniken. Nennenswert sind dabei die einfache Wortanzahl und *word2vec*. Szenario-spezifische Features wurden hingegen nicht eingesetzt. *Bag-of-N-Words* konnte in Kombination mit der einfachen Wortanzahl und *Logistic Regression* einen F_1-*Wert* von 75% erreichen. Eine Besonderheit bei diesem Ergebnis ist, dass vom Titel und *Körper* eines Beitrags Features getrennt extrahiert und erst in einem weiteren Schritt kombiniert wurden. Damit konnten bessere Ergebnisse erzielt werden, als mit Titel und *Körper* von Beginn an zusammengeführt. Die

[2] https://blog.twitter.com/official/en_us/topics/product/2017/tweetingmadeeasier.html

zweite Arbeit erreichte mit einer SVM und *Bag-of-N-Words* einen F_1-*Wert* von 77% und schnitt damit besser als *word2vec* ab. Obwohl diese Arbeiten aufgrund ihrer unterschiedlichen Kategorien nicht direkt miteinander verglichen werden können, geben diese gut den aktuellen Stand bei der Klassifizierung von Beiträgen wieder.

Im Kontrast dazu haben untersuchte Arbeiten zur Klassifikation von Kommentaren in den meisten Fällen nichts mit der Zuordnung zu einem Subreddit zu tun. Viel mehr liegt der Fokus hier auf der Zuweisung von vordefinierten Kategorien wie *negativ*, *neutral* oder *positiv*, wie sie bei Stimmungsanalysen üblich sind. In diese Kategorie fällt auch [17], welche Kommentare anhand ihrer Nützlichkeit zu einem Thema bewertet. Eine Ausnahme macht dabei [34]. Hier werden *künstliche neuronale Netze* eingesetzt um Kommentare ihrem Subreddit zuzuweisen. Bei *künstlichen neuronalen Netzen* fällt jedoch der Feature-Engineering Prozess zum Großteil weg, was einen Einsatz in dieser Arbeit unmöglich macht und deswegen nicht näher auf diese Forschungsarbeit eingegangen wird.

Bei aktuellen Arbeiten zu Text-Klassifikation mit Reddit konnte somit sowohl die Klassifikation von Beiträgen als auch von Kommentaren festgestellt werden, aber nicht eine Kombination von beidem. Genau an diesem Punkt möchte diese Arbeit ansetzen und Beiträge nur mithilfe ihrer Kommentare klassifizieren. Dies setzt natürlich eine entsprechende Anzahl an Kommentaren pro Beitrag voraus, kann dafür aber auch für Beiträge, welche keinen Text beinhalten eingesetzt werden.

Kapitel 4

Feature Engineering Prozess

Dieses Kapitel widmet sich speziell dem Thema Feature-Engineering, da dieser Bereich einen wichtigen Teil im Text-Klassifikation Prozess darstellt. Den Anfang macht dabei die Verarbeitung von Textdaten. Es wird der gesamte Prozess von der Textaufbereitung bis zu den Feature Vektoren beschrieben. Dabei werden in Abschnitt 4.2 Text-Modelle angesprochen, welche auch als Ausgangsbasis für die Klassifikation der Forumsbeiträgen dienen. Da Text-Modelle häufig in Kombination mit Text-Features zu besseren Ergebnissen [33] führen, werden im Zuge der Arbeit auch eigenständige Text-Features erarbeitet.

In Abschnitt 4.4 wird dann speziell auf Feature-Engineering außerhalb von Texten eingegangen. Damit sind die in der Einleitung erwähnten Eigenschaften eines Forums gemeint. Im Bezug auf Beiträge wären das Features, welche sich von der Funktionalität, Struktur oder auch dem Autor herleiten lassen. Ziel ist es mit diesen Features in Kombination mit Text-Modellen bessere Ergebnisse zu erzielen, als dieselben Text-Modelle in Kombination mit Text-Features.

Zum Abschluss wird auf unterschiedlichen Möglichkeiten zur Auswahl der entwickelten Features eingegangen.

4.1 Textaufbereitung

Bevor Features aus einem Corpus extrahiert werden können, gilt es diesen aufzubereiten. Inwiefern ein Corpus aufbereitet bzw. gesäubert werden muss, hängt vom Zustand der Textdokumente und dem jeweiligen Einsatzbereich ab. In manchen Situationen kann es auch von Vorteil sein, nicht alle hier gelisteten Schritte durchzuführen bzw. den Prozess der Textaufbereitung mit dem Feature-Engineering abzustimmen. Als Beispiel könnte die Anzahl an Rechtschreibfehlern pro Dokument gezählt und in weiterer Folge als Feature eingesetzt werden. Diese Information würde durch zu starkes Aufbereiten des Textes verloren gehen. Die Rechtschreibkorrektur kann dennoch durchgeführt werden, um das Ergebnis mit Techniken wie Bag-of-Words zu verbessern. Bevor jedoch mit einer Rechtschreibkorrektur begonnen werden kann, sollten alle Textdokumente normalisiert und gesäubert werden.

4.1.1 Normalisierung

Bei der Normalisierung von Text wird dieser in bestimmten Bereichen verändert, ohne jedoch seine Bedeutung zu verlieren [50]. So werden meist alle Großbuchstaben durch Kleinbuchstaben ersetzt und Abkürzungen, welche häufig in englischen Texten vorkommen, ausgeschrieben (*don't* wird zu *do not*). Dadurch werden alle Textdokumente im Corpus standardisiert. In manchen Fällen können auch Akzentbuchstaben ersetzt werden (*á* wird zu *a*) solange dabei nicht Wörter mit unterschiedlichen Bedeutungen verschmelzen.

4.1.2 Säuberung

In einem nächsten Schritt sollte der Text von speziellen Symbolen wie HTML-Tags oder Sonderzeichen gesäubert werden. Das Entfernen von HTML-Tags ist meist nur dann notwendig, sollten die Texte von einer Webseite „abgegriffen“[1] worden sein. Zusätzlich zum Entfernen von speziellen Symbolen, können auch gleich Wörter ohne besonderer Bedeutung, im englischen *Stop-Words* (z. B. der, die, das) genannt, gelöscht werden. Das Entfernen von *Stop-Words* kann aber auch erst nach der Rechtschreibkorrektur durchgeführt werden, da davor eventuell nicht alle *Stop-Words* erkannt werden.

Das Entfernen von Daten ist immer mit Vorsicht zu genießen. Auf der einen Seite soll der Text von unnötigem Rauschen gesäubert werden, jedoch können damit auch wichtige Information verloren gehen. Für eine Gefühlsanalyse sind gerade Sonderzeichen oder Emoticons von besonderer Bedeutung und sollten bei solch einem Szenario nicht entfernt sondern als Feature genutzt werden.

4.1.3 Rechtschreibung

Die Rechtschreibkorrektur wird in dieser Arbeit, wie eingangs erwähnt, nicht nur als Mittel zur Aufbereitung des Textes, sondern auch als Feature Engineering Tool eingesetzt. Aber auch wenn keine Features extrahiert werden sollen, macht es Sinn die Rechtschreibkorrektur durchzuführen, da falsch geschriebene Wörter aufgrund ihres seltenen Auftretens von Techniken wie *Bag-of-Words* ignoriert werden könnten.

4.1.4 Umgangssprache

Ein Thema das eng mit der Rechtschreibung verbunden ist, ist die Umgangssprache. Gerade in Foren oder sozialen Medien wird häufig eine Form der Umgangssprache oder *Internet-Slang* verwendet. Dabei gibt es viele Formen des Internet-Slangs, welche einfache Abkürzungen wie BRB oder Symbole wie Emoticons beinhalten. Wie auch bei der Rechtschreibüberprüfung können diese Merkmale gezählt und als Features eingesetzt werden. Diese Art der Textaufbereitung wurde jedoch nicht in dieser Arbeit durchgeführt, da der Fokus mehr bei Features außerhalb des Textes gelegen ist.

4.1.5 Normalformenreduktion

Eine weitere Form der Textaufbereitung ist die Normalformenreduktion (engl. *Stemming*). Dabei werden verschiedene Variationen eines Wortes zu ihrem Wortstamm zu-

[1] Das Abgreifen von Inhalten einer Webseite wird auch *Web Scraping* genannt.

rückgeführt. Wörter wie *ging* und *gegangen* werden zu *gehen* [31]. Diese Technik kann eingesetzt werden, um die Anzahl an Features in Text-Modellen zu reduzieren, findet jedoch in dieser Arbeit keine Anwendung,

4.2 Text-Modelle

Nach der Textaufbereitung kann mit dem eigentlichen Feature-Engineering begonnen werden. Gestartet wird dabei häufig mit Text-Modellen, welche zu jedem Dokument im Corpus einen Feature-Vektor liefern. Für diese Arbeit dienen Text-Modelle auch als Ausgangsbasis für die Text-Klassifikation. Diese werden im späteren Verlauf der Arbeit mit weiteren Features kombiniert, um deren Einfluss aufeinander zu messen.

4.2.1 Bag-of-Words

Bei Bag-of-Words handelt es sich um das einfachste Konzept, wenn es darum geht Wörter in Features umzuwandeln. Es wird häufig in der Anfangsphase verwendet, um schnell Ergebnisse erzielen zu können. Der Grundgedanke bei Bag-of-Words ist es, die Häufigkeit der in einem Textdokument enthaltenen Wörter zu repräsentieren. Hier wird auf die Reihenfolge oder Grammatik keine Rücksicht genommen. Die am Ende erhaltenen Kombinationen mit Wörtern und deren Häufigkeit können nun als Features für eine Methode zur Klassifizierung eingesetzt werden [5]. Im folgenden Beispiel wird die Technik anhand eines Corpus mit drei Sätzen erläutert.

1. Max hat eine Maus und eine Katze.
2. Lisa hat eine Katze.
3. Max und Lisa haben eine Katze.

Diese Sätze werden nun mit Bag-of-Words in die Feature Vektoren umgewandelt. Diese sind in Tabelle 4.1 aufgelistet. Jede Spalte repräsentiert dabei ein Wort welches im Corpus vorkommt. Die Zahl repräsentiert die Häufigkeit dieser Wörter im jeweiligen Textdokument bzw. Satz. Damit kann eine Zeile als Feature Vektor einem Eintrag im Corpus zugeordnet werden. Vorsicht ist bei einem großen Corpus mit vielen Wörtern geboten. Da jedes eindeutige Wort im Corpus zu einem Feature wird, kann es leicht passieren, dass sehr viele Features entstehen, welche sich negativ auf die Performance auswirken. Um diesem entgegen zu wirken kann entweder eine Normalformreduktion durchgeführt oder extrem häufig bzw. selten vorkommende Wörter ignoriert werden.

Tabelle 4.1: Bag-of-Words

ID	*Max*	*Lisa*	*hat*	*haben*	*eine*	*und*	*Katze*	*Maus*
1	1	0	1	0	2	1	1	1
2	0	1	1	0	1	0	1	0
3	1	1	0	1	1	1	1	0

N-Gramms

N-Gramms ist eine einfache Erweiterung des Bag-of-Words Konzept. Bei dieser Technik werden im Gegenzug zu seinem Vorgänger nicht einzelne Wörter, sondern N Wortkombinationen gezählt. Die Anzahl der vorhandenen Wörter in einer Kombination kann selbst gewählt werden, ist jedoch im Normalfall auf zwei Wörter beschränkt. Mit dieser Technik würden für den ersten Satz, im oben angeführten Beispiel, nun die Wortkombinationen „Max hat“, „hat eine“,„eine Maus“, usw. als Features ausgewählt werden.

4.2.2 Term Frequency-Inverse Document Frequency (TF-IDF)

Bei TF-IDF wird, im Gegensatz zu Bag-of-Words, nicht die Häufigkeit, sondern die Relevanz eines Wortes gemessen. Diese ergibt sich im Zusammenhang mit der Häufigkeit eines Wortes verteilt über dem gesamten Corpus. Kommt ein Wort öfter vor, ist es weniger relevant oder umgekehrt, wenn es nur selten vorkommt [35]. Der damit erhaltene Wert kann nun wieder als Feature eingesetzt werden. Die Relevanz eines Wortes wird bei TF-IDF mit der Formel

$$w_{i,j} = t_{i,j} \times \log \frac{N}{d_i} \tag{4.1}$$

berechnet, wobei $t_{i,j}$ für die Häufigkeit eines Wortes i in einem Dokument j steht und damit den Wert des Bag-of-Words Modells widerspiegelt. N steht für die gesamte Anzahl an Dokumenten im Corpus und d_i für die Anzahl an Dokumenten welche dieses Wort i enthalten. Angewendet auf das Beispiel aus dem vorherigen Abschnitt ergeben sich damit folgende Werte für unseren Corpus.

In Tabelle 4.2 ist gut zu sehen, dass Wörter wie „eine“, und „Katze“ keine Relevanz, d.h. einen Wert von 0, besitzen, da sie in jedem Dokument des Corpus vorkommen. „Maus“ und „haben“ besitzen dabei ein größere Relevanz gegenüber den restlichen Wörter, da sie jeweils nur einmal im gesamten Corpus vorkommen.

Tabelle 4.2: TF-IDF

ID	*Max*	*Lisa*	*hat*	*haben*	*eine*	*und*	*Katze*	*Maus*
1	0.18	0	0.18	0	0	0.18	0	0.48
2	0	0.18	0.18	0	0	0	0	0
3	0.18	0.18	0	0.48	0	0.18	0	0

4.3 Text-Features

Die eben gezeigten Feature Vektoren in den vorherigen Beispielen können eng definiert auch als Text-Features bezeichnet werden. Im Bezug auf diese Arbeit sind damit aber explizit Features gemeint, welche sich aus einem Text ableiten lassen, jedoch nicht durch Text-Modelle erstellt werden. Darunter fällt das Zählen von Wörtern, Sonderzeichen oder auch Rechtschreibfehlern. Am Beispiel von Forumsbeiträgen zählen dazu auch die

Häufigkeit von externen Links oder das Erwähnen eines anderen Nutzers. Diese lassen sich durch das Erkennen bestimmter Passagen im Text feststellen. Um weitere Text-Features zu generieren wurde zusätzlich *Part-of-Speech Tagging* eingesetzt.

4.3.1 Part-of-Speech Tagging

Part-of-Speech Tagging ist ein Verfahren, welches Wörter eines Textdokuments ihrer Wortart zuweist. Hier können die Wortarten und ihre Häufigkeit pro Textdokument als Features für die Klassifikation genutzt werden. Bei diesem Verfahren handelt es sich aufgrund der Mehrdeutigkeit von Wörtern um ein besonders Komplexes. Die genaue Kategorie eines Wortes wird stark vom aktuellen Kontext beeinflusst. So ist zum Beispiel das Wort „Angeln“ ein Nomen oder ein Verb, je nachdem ob es sich um die Mehrzahl des Wortes „Angel“ oder um die Tätigkeit handelt.

4.4 Szenario-spezifische Features

Diese Features sind, wie der Name verrät, vom jeweiligen Szenario abhängig. Im Bezug auf diese Arbeit sind damit Features gemeint, welche sich aus den Forum Eigenschaften bzw. der Struktur von Beiträgen ableiten lassen. Hier gibt es keine vordefinierten Techniken, welche eingesetzt werden können wie im Fall von Text-Features. Vieles hängt von der Datenbasis und den zur Verfügung gestellten Daten ab.

Obwohl sich Beiträge innerhalb Reddits durch ihre Themen stark unterscheiden können, so ist deren Aufbau immer gleich. Ein Beitrag ist immer einer Kategorie zugeordnet und kann eine Vielzahl an Kommentaren besitzen. Innerhalb eines Beitrags können auf einen Kommentar auch mehrere Kommentare folgen, wodurch sich oft längere Kommentarstränge ergeben. Diese Struktur wird in der Abbildung 4.1 beispielhaft dargestellt. Hier ist gut zu erkennen, dass ein Kommentar nicht nur aus seinem Text besteht, sondern viele weitere kleine Eigenschaften wie Punkte, ein Erstellungsdatum oder eine Auszeichnung besitzt. In Tabelle 4.3 wird gezeigt wie diese vier Kommentare in der Datenbank gespeichert werden.

Felder wie „Editiert“ und „Ausz.“ können ohne weitere Änderungen sofort als Feature genutzt werden. Auf den ersten Blick trifft das auch für das Feld „Punkte“ zu. Der letzte Kommentar hat jedoch eine negative Punkteanzahl, was zu Problemen mit dem Klassifizierer *Naïve Bayes* führt, welcher keine negativen Werte erlaubt. Um die Punkte trotzdem als Feature nutzen zu können, kann die Funktionalität des *MinMaxScaler* eingesetzt werden. Dieser wird in der Python Bibliothek *scikit-learn* zur Verfügung gestellt und transformiert eine Reihe an Werten in einen benutzerdefinierten Bereich [45].

Um Features wie die Anzahl an Kindern oder Geschwister pro Kommentar nützen zu können, bedarf es etwas mehr Aufwand. Hier muss eine eigene Methode entwickelt werden, um diese Features aus den Daten zu extrahieren. Wie eine Implementierung dazu aussehen kann wird in Abschnitt 6.4.3 gezeigt.

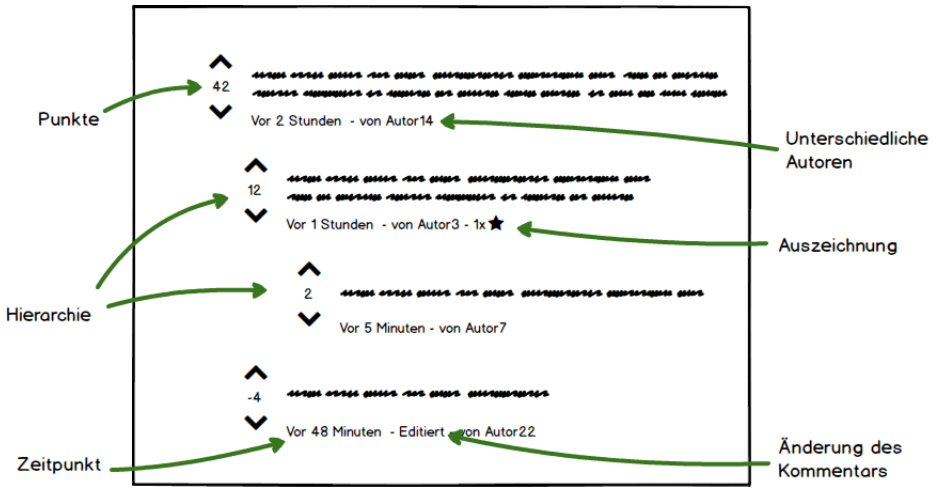


Abbildung 4.1: Struktur der Kommentare zu einem Beitrag

Tabelle 4.3: Daten zu den vier Kommentaren

ID	*Punkte*	*Datum*	*Eltern_ID*	*Autor*	*Editiert*	*Ausz.*
1	42	03.08.2019 09:12	Null	Autor14	0	0
2	12	03.08.2019 10:08	1	Autor3	0	1
3	2	03.08.2019 11:15	2	Autor7	0	0
4	-4	03.08.2019 10:32	1	Autor22	1	0

4.5 Feature Auswahl

Nach dem Feature Engineering sollten alle Features evaluiert und bei schlechtem Abscheiden auch wieder verworfen werden. Das mag vielleicht im ersten Moment kontraproduktiv wirken, da sehr viel Zeit in das Feature Engineering geflossen ist, doch macht diese Vorauswahl von Features in vielerlei Hinsicht auch Sinn. Es werden dabei, bevor überhaupt mit der Klassifikation begonnen wird, redundante aber auch irrelevante Features aus dem bestehend Feature-Set entfernt. Dadurch kann die Klassifikation im Endeffekt schneller durchgeführt und im besten Fall auch noch genauer werden [28]. Oft ist es aber einfach auch technisch unmöglich alle Features für die Klassifikation miteinzubeziehen.

Für Text-Modelle findet diese Feature Auswahl in einem Schritt mit dem Feature Engineering statt. Hier können Restriktionen gesetzt werden, welche das Aufnehmen von Wörtern als Features limitiert. In Abschnitt 6.4.1 wird gezeigt, wie diese Restriktionen in der Praxis eingesetzt werden können. Für einzelne Features wie sie in den Abschnitten 4.3 und 4.4 beschrieben werden gibt es mehrere Möglichkeiten zur Feature Auswahl, welche sich in die drei Ansätze *Filter*, *Wrapper* und *Embedded* einteilen lassen.

4.5.1 Filter-Ansatz

Bei dem Filter-Ansatz werden alle Features untereinander und deren Beziehung mit der zu klassifizierenden Kategorien verglichen, d.h. die Methode zur Klassifizierung spielt bei diesem Auswahlverfahren keine Rolle. Als Vergleich kann dabei die *Pearson-Korrelation* genutzt werden. Dies gibt eine Korrelation zwischen Eigenschaften mit einem Wert von -1 bis 1 an. Eine hohe Korrelation und damit ein Wert nahe 1 bedeutet, dass sich diese Eigenschaften gleich verhalten. Das ist besonders für die Beziehung zwischen einem Feature und der Kategorie erstrebenswert, da sich dieses Feature somit sehr wahrscheinlich positiv auf das Ergebnis der Klassifizierung auswirken wird.

Im Gegensatz dazu ist eine hohe Korrelation zwischen zwei Features ein Indikator für redundante Information. Darunter fällt auch ein Wert der nahe -1 liegt, da dieser auch eine hohe Korrelation aber in die entgegengesetzte Richtung angibt. Eine geringe Korrelation und damit einen Wert nahe null zwischen einem der beiden Features und der Kategorie kann dabei helfen zu entscheiden, welches Feature verworfen wird.

4.5.2 Wrapper-Ansatz

Hier wird die Methode zur Klassifizierung als Auswahlverfahren herangezogen. Dabei werden die Features entweder einzeln oder in Kombination miteinander eingesetzt, um mit der Methode einen Teil des Corpus zu klassifizieren. Features, welche dabei besonders schlecht abschneiden, werden verworfen. Bei der Implementierung wurde eine Kombination von Filter- und Wrapper-Ansatz zur Auswahl von Features genutzt.

4.5.3 Embedded-Ansatz

Bei dem Embedded-Ansatz wird die Feature Auswahl während der Klassifikation durchgeführt, d.h. die Methode zur Klassifizierung entscheidet selbst, welche Features besser bzw. schlechter geeignet sind. Bestes Beispiel dafür sind die im nächsten Kapitel beschriebenen Entscheidungsbäume.

Kapitel 5

Methoden zur Text-Klassifizierung

Wurden im vorherigen Kapitel Features und damit sozusagen der Treibstoff für Text-Klassifikation beschrieben, geht es in diesem Kapitel um die Methoden und Algorithmen, welche Features verarbeiten und schlussendlich zu einem Ergebnis führen. Wie bei den Features sollten auch diese getestet und miteinander verglichen werden, da es auch hier, abhängig vom Szenario, zu Schwankungen bei den Ergebnissen kommen kann. Auch bei den Methoden selbst können oft Einstellungen angepasst werden. Beim sogenannten *Hyperparameter Tuning* geht es darum die möglichen Einstellungen für einen Algorithmus zu optimieren und damit das Ergebnis zu verbessern. Wie diese Technik in der Praxis aussieht, wird in Abschnitt 6.5.1 erläutert.

Die folgende Auflistung an Methoden zur Text-Klassifizierung dient als letztes Kapitel, bei dem wichtige Bestandteile für die Implementierung erklärt werden. Den Anfang macht Naïve Bayes, welcher sich immer noch einer hohen Beliebtheit erfreut, und gerade während dem *Prototyping* häufig Einsatz findet.

5.1 Naïve Bayes

Eine Klassifikation mit Naïve Bayes kann schnell und einfach umgesetzt werden und bietet zugleich einen guten Ausgangswert zum Vergleich mit anderen Modellen. Bewährte Anwendungsgebiete für diese Methode ist das Erkennen von Spam-Emails oder die Zuordnung von Kategorien zu Nachrichtenbeiträgen. Im Kapitel Implementierung 6 wird Naïve Bayes auch als Ausgangswert zum Vergleich mit Modellen wie AdaBoost und GradientBoost verwendet, welche in den weiteren Abschnitten beschrieben werden. Naïve Bayes basiert auf den Satz von Bayes. Damit kann die Wahrscheinlichkeit eines Ereignis A unter der Bedingung, dass ein Ereignis B bereits eingetreten ist, mit

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} \tag{5.1}$$

berechnet werden, wobei $P(B \mid A)$ die Wahrscheinlichkeit von B unter der Bedingung, dass ein Ereignis A bereits eingetreten ist und $P(A)$ die Wahrscheinlichkeit von A bzw. $P(B)$ die Wahrscheinlichkeit von B ist. Um diesen Ansatz nun für Text-Klassifikation nützen zu können muss eine Abwandlung dieser Formel namens *Multinomial Naïve Bayes* eingesetzt werden. Die Bezeichnung Multinominal bezieht sich dabei auf die

Wahrscheinlichkeitsverteilung von Wörtern, oder in diesem Fall Features, in einem Textdokument. Sie ist auch der Grund warum negative Werte wie in Abschnitt 4.4 nicht genutzt werden können, da diese einen positiven Wert voraussetzt. Die Wahrscheinlichkeit einer Kategorie c gegeben einem Textdokument d ergibt sich damit durch

$$P(c \mid d) = \frac{P(c) \prod_{w \in d} P(w \mid c)^{n_{wd}}}{P(d)} \tag{5.2}$$

wobei n_{wd} die Häufigkeit eines Features w in einem Textdokument d ist. $P(w \mid c)$ ist dabei die Wahrscheinlichkeit, dass ein Feature w in der Kategorie c vorkommt [10].

5.2 Nächste-Nachbarn-Klassifikation

Nächste-Nachbarn-Klassifikation (engl. *k-Neighbour*) ist ähnlich wie Naïve Bayes eine der Einstiegsmethoden zur Klassifikation von Daten. Hier wird die Kategorie eines Objekts anhand der Kategorie seiner k nächsten Nachbarn bestimmt. Dabei ist k frei wählbar. Im einfachsten Fall ist $k = 1$ und bezieht damit nur einen Nachbarn in Betracht um die Kategorie eines neuen Objekts zu bestimmen. Um nicht durch Ausreißer beeinflusst zu werden wird meist ein $k > 1$ gewählt. In diesem Fall wird die am häufigsten vorkommende Kategorie der Nachbarn dem neuen Objekt zugewiesen [27].

In Abbildung 5.1 wird die Methode anhand eines Beispiels für Textdokumente veranschaulicht. Hier befinden sich bereits klassifizierte Dokumente verteilt in ihren Gruppen. Es soll nun ein neues Dokument anhand seiner Ähnlichkeit zu drei anderen Textdokumenten klassifiziert werden. Die Ähnlichkeit wird dabei anhand der Features bestimmt. In diesem Fall wird dem neuen Dokument die Kategorie A zugeordnet. Um ein „Unentschieden“ bei den Kategorien zu vermeiden sollte k keine gerade Zahl annehmen. In der Praxis wird ein optimales k durch *Hyperparameter Tuning* ermittelt.

Bei dieser Methode zur Klassifizierung von Objekten kann es bei einer zu großen Anzahl von Features bzw. Corpus eine zu lange Laufzeit bekommen, weshalb sie in dieser Arbeit nicht eingesetzt wurde.

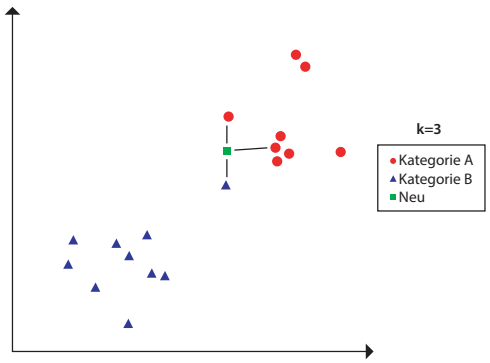


Abbildung 5.1: Veranschaulichung des Nächste-Nachbarn-Klassifikators. Bildquelle [44].

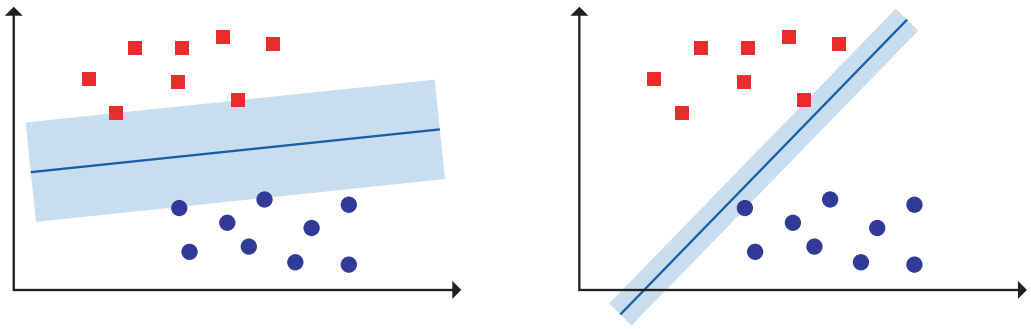
5.3 Support Vector Maschine

Eine Support Vector Maschine (SVM) wird nicht nur zur Klassifikation von Objekten, sondern auch für viele weitere Problemstellungen im Bereich des maschinellen Lernens, eingesetzt. Zur Berechnung von Ergebnissen arbeitet eine SVM mit einer Reihe an mathematischen Funktionen. Auf diese wird jedoch nicht weiter in dieser Arbeit eingegangen. Ziel dieses Abschnittes ist es, die Funktionsweise von SVMs, im Bereich der Klassifikation, anhand eines einfachen Beispiels zu erklären und auf den Einsatz in dieser Arbeit einzugehen.

Um Objekte in eine von zwei Kategorien einzuteilen, werden diese von der SVM in einem ersten Schritt als Vektoren auf einer Fläche dargestellt. Danach wird zwischen beide Gruppen eine lineare Trennlinie gezeichnet. Diese Technik wird in Abbildung 5.2(a) dargestellt. Dabei möchte man den Abstand, auch genannt Stützvektoren, zwischen Trennlinie und Objekten so groß wie möglich halten, um Raum für neue Objekte zu lassen. In der Praxis wird jedoch öfters manuell in die Berechnung der Stützvektoren eingegriffen, da in manchen Situationen dieser Abstand unerwünscht ist. Gezeigt wird das in Abbildung 5.2(b).

In der Praxis kommt es fast immer vor, dass eine lineare Trennung Aufgrund der Position der Vektoren unmöglich wird. Hier kommt der sogenannte *Kernel-Trick* zum Einsatz. Dabei werden die Vektoren solange in einen höherdimensionalen Raum überführt, bis es möglich ist eine Trennlinie bzw. Fläche zu zeichnen [48].

Gibt es mehr als zwei Kategorien, wie im Falle diese Arbeit, wird die Klassifikation in mehrere binäre Klassifizierungen aufgeteilt und zum Schluss evaluiert. Dabei wird zwischen den Evaluierungsmethoden *One-vs.-one* und *One-vs.-rest* unterschieden. Für diese Arbeit wird die `LinearSVC` Methode eingesetzt, welche nach dem *One-vs.-rest* Prinzip evaluiert.



(a) (b)

Abbildung 5.2: (a) Standardmäßige Stützvektoren, (b) angepasste Stützvektoren.

5.4 Entscheidungsbäume

Entscheidungsbäume (engl. *Decision Trees*) sind ein Tool zur Abbildung von aufeinanderfolgenden Regeln. Ein Entscheidungsbaum besteht immer aus einem *Wurzelknoten*, von welchem jeder Entscheidungsprozess gestartet wird. Von hier verläuft eine Entscheidung über beliebig viele *Knoten*, und damit durch Features definierte Regeln. Der Prozess ist abgeschlossen, sobald das Ende und damit die *Blätter* erreicht wurden. Ein Blatt repräsentiert dabei das Ergebnis dieser Entscheidung.

In Abbildung 5.3 wird ein einfacher Entscheidungsbaum dargestellt, welcher Dokumente anhand ihrer Anzahl von Wörtern und Adjektiven in eine von drei Kategorien unterteilt. In der Praxis sind Entscheidungsbäume um einiges komplexer und besitzen eine Vielzahl von Knoten und Blättern.

Für Klassifikationen wird selten ein Entscheidungsbaum genutzt, da er stark zu Überanpassung neigt. Er kann gut mit Trainingsdaten umgehen, scheitert aber bei der Klassifizierung von Testdaten. Verfahren wie *Random Forest*, *AdaBoost* oder *Gradient-Boosting* bauen auf der Idee von Entscheidungsbäumen auf und versuchen das Problem der Überanpassung mit unterschiedlichen Vorgehensweisen zu mindern.

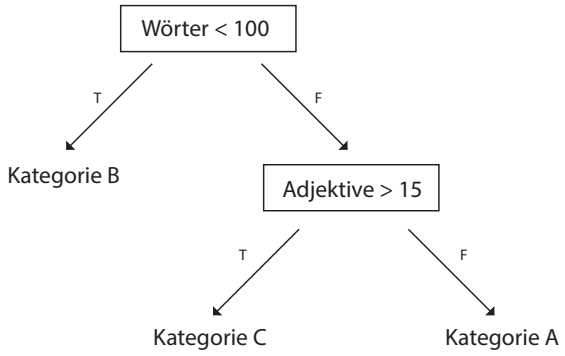


Abbildung 5.3: Ein Entscheidungsbaum mit zwei Knoten und drei Blättern.

5.4.1 Aufbau

Ist die Klassifikation mit einem fertigen Entscheidungsbaum relativ simpel, so ist deren Erstellung um einiges komplexer. Die wichtigste Frage bei der Erstellung ist die Platzierung der Regeln, welche sich aus den Features ableiten. Dabei werden alle Feature einzeln bewertet, wie gut diese von sich aus die Trainingsdaten klassifizieren können. Zur Bewertung der Features bei diesem Prozess können die Methoden

- *Gini Verunreinigung*,
- *Informationsgewinn* oder
- *Varianz Reduzierung*

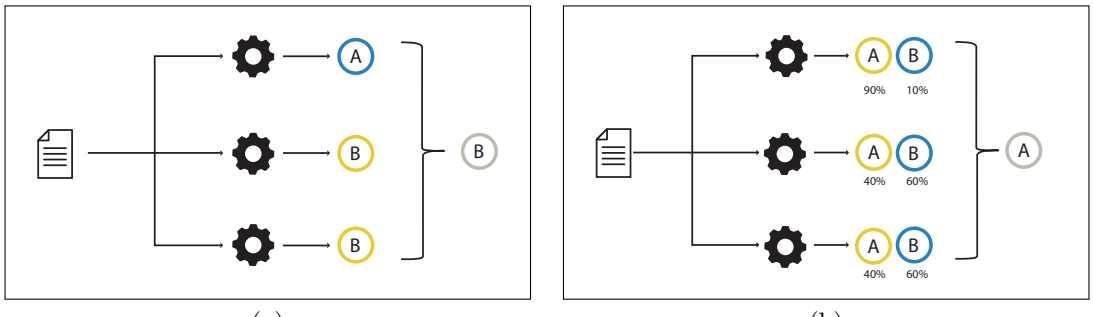
eingesetzt werden [42]. Jenes Feature, welche die Objekte am Besten bewertet, kommt als Wurzelknoten zum Einsatz. Dieser Prozess wird nun auf die Ergebnisse des Wurzelknotens bzw. eines inneren Knotens mit den restlichen Features angewandt. Dabei kann es auch vorkommen das ein Feature nicht als Regel eingesetzt wird, weil es zu schlechte Ergebnisse liefert.

5.5 Ensemblemethode

Bei der Ensemblemethode handelt es sich nicht um eine Methode an sich, sondern um ein allgemeines Konzept, bei dem die Resultate verschiedener Methoden vereint und nach einem Bewertungsverfahren ein gemeinsames Ergebnis liefern. Die dabei verwendeten Methoden werden meist als „schlechte Klassifizierer“ bezeichnet, da sie im Einzelnen keine guten Ergebnisse liefern. Durch den gebündelten Einsatz kann das Ergebnis jedoch signifikant verbessert werden.

Um aus dem Einzelergebnissen ein Gemeinsames zu ermitteln, können zwei Verfahren zum Einsatz kommen. Bei der ersten Variante, *Hard Voting* (siehe Abbildung 5.4(a)), wird die Kategorie, welche am häufigsten in den Einzelergebnissen vorkommt, als Gesamtergebnis gewählt. Es gewinnt somit die einfache Mehrheit. Bei der zweiten Variante, *Soft Voting* (siehe Abbildung 5.4(b)), wird ein genauer Blick auf die Wahrscheinlichkeit der Kategorien in den Einzelergebnissen geworfen. Dabei wird die Kategorie mit der höchsten Wahrscheinlichkeit gewählt. In dieser Arbeit wird dieses Konzept auch verwendet, um bei der Klassifizierung von mehreren Kommentaren auf ein gemeinsames Ergebnis zu kommen.

Aber nicht nur bei der Ermittlung des gemeinsamen Ergebnisses gibt es unterschiedliche Vorgehensweisen. Auch beim Trainieren der Methoden wird zwischen den Techniken *Bagging* und *Boosting* unterschieden. Beim *Bagging* werden alle Methoden zur Klassifizierung parallel mit zufällig ausgewählten Einträgen aus dem Trainingsset trainiert. Beim *Boosting* werden die Methoden sequentiell mit, nicht komplett zufällig, ausgewählten Einträgen trainiert. Die Einträge, mit denen eine Methode trainiert, ergeben sich dabei teilweise durch Einträge, welche von der vorherigen Methode beim Trainieren fehlerhaft klassifiziert wurden. Dadurch lernen die Methoden sozusagen von den Fehlern ihrer Vorgänger [39].



(a) (b)

Abbildung 5.4: Vergleich von Hard (a) mit Soft Voting (b).

5.5.1 Random Forest

Bei der Methode *Random Forest* werden N Entscheidungsbäume nach der *Bagging* Technik trainiert und mittels *Hard Voting* über ein endgültiges Ergebnis entschieden. Dabei wird auch in den Prozess, wie ein Entscheidungsbaum aufgebaut wird, eingegriffen. Es werden in diesem Fall für jeden Knoten im Baum nicht alle, sondern zufällig ausgewählte Features bewertet, um für noch mehr Diversität der Bäume zu sorgen. Dadurch wird dem Problem der Überanpassung entgegengewirkt. Diese Methode ist relativ simpel und wird deshalb, ähnlich wie *Naïve Bayes*, oft als Ausgangswert zum Vergleich mit anderen Methoden genutzt.

5.5.2 AdaBoost

Bei *AdaBoost* werden die Entscheidungsbäume, im Gegensatz zu *Random Forest* nach der *Boosting* Technik trainiert. *AdaBoost* beschränkt sich dabei nicht auf den Einsatz von Entscheidungsbäumen, wird jedoch in dieser Form am Häufigsten genutzt. Eine wichtige Anmerkung für diese Methode ist, dass jedem Eintrag im Trainingsset ein Wert, welcher seine Priorität gegenüber anderen Einträgen darstellt, zugewiesen wird. Dieser beeinflusst die *zufällige* Auswahl an Einträgen zum Trainieren eines Baums und ist zu Beginn für jeden Eintrag gleich.

Auch die Bäume, die während des Trainings durch *AdaBoost* entstehen, haben eine Besonderheit, denn sie besitzen nur einen Wurzelknoten mit zwei Blättern. Ein Baum mit diesen Eigenschaften wird auch *Stumpf* genannt. Jedem Baum wird nach seinem Training eine Gewichtung w zugewiesen, welche mit der Formel

$$w = l \times \log \frac{1-r}{r} \tag{5.3}$$

berechnet wird, wobei l für die Lernrate und r für die Fehlerrate steht. Die Fehlerrate ergibt sich dabei aus der Summe über die Prioritäten der Einträge, welche von diesem Baum falsch klassifiziert wurden. Für diese Einträge steigt die Priorität, was die Wahrscheinlichkeit, Teil des nächsten Trainingssets zu werden erhöht. Alle anderen Einträge verlieren dabei an Priorität, sodass die Summe über alle Einträge immer 1 bleibt. Die Lernrate l ist ein fixer Wert und wird meist durch *Hyperparameter Tuning* ermittelt. Das Endergebnis einer Klassifizierung mit *AdBoost* ergibt sich, durch die gewichteten Einzelergebnisse der Bäume und ist somit eine spezielle Form des *Soft Votings*.

5.5.3 Gradient Boosting

Gradient Boosting funktioniert, wie der Name bereits verrät, auch nach dem *Boosting* Prinzip. Wie bei *AdaBoost* beeinflusst das Training eines Baums seinen Nachfolger. Dabei sind diese Bäume aber nicht auf die Größe eines Stumpfs beschränkt. Es wird stattdessen ein Limit für die Anzahl an Blättern für jeden Baum definiert. Die Funktionsweise von *Gradient Boosting* ist auf binäre Klassifikation, also zwei Kategorien limitiert. Für mehrere Kategorien wird dieser Prozess in mehrere binäre Klassifizierungen aufgeteilt.

Bevor das Trainieren des ersten Baums startet, wird eine initiale Prognose p_{i} für

beide Kategorien K_a und K_b mit der Formel

$$p_\mathrm{i} = \frac{\exp(\log \frac{\sum K_\mathrm{a}}{\sum K_\mathrm{b}})}{1 + \exp(\log \frac{\sum K_\mathrm{a}}{\sum K_\mathrm{b}})} \tag{5.4}$$

berechnet. Mit dieser Prognose p_i wird für jeden Eintrag die Abweichung (*Residuum*) zur tatsächlichen Kategorie, wobei $K_\mathrm{a} = 1$ und $K_\mathrm{b} = 0$ ist, erstellt. Erst jetzt wird ein Baum erzeugt, welcher nicht die Kategorie, sondern diese Abweichung prognostizieren soll. Durch das Training eines Baumes, werden die Abweichungen aktualisiert. Die Berechnung der Abweichung wird dabei auch wieder von einer Lernrate l beeinflusst. Der gesamte Prozess wird nun so oft wiederholt, bis ein definiertes Maximum an Bäumen erreicht oder keine Änderungen der Abweichungen mehr auftreten. Eine Klassifizierung berechnet sich dabei aus den Einzelergebnissen aller erstellten Bäume.

Kapitel 6

Implementierung

Bei der Implementierung werden die in den vorherigen Kapiteln beschriebenen Techniken zu Feature Engineering und Text Klassifizierung in die Praxis umgesetzt. Ziel ist es nach dem Prinzip des überwachten Lernens Forumsbeiträge anhand ihrer Kommentare in 10 verschiedene Kategorien zu unterteilen. Als Datenbasis dienen dabei Beiträge der Internetplattform Reddit.

Zur Klassifizierung der Beiträge werden zwei Ansätze verfolgt. Zum ersten werden alle Kommentare eines Beitrags vereint und somit als ein Textdokument klassifiziert. Beim zweiten Ansatz werden die Kommentare einzeln klassifiziert und die Kategorie des Beitrags wird dabei, wie bei Ensemblemethoden, mithilfe von *Hard* bzw. *Soft Voting* ermittelt. Für beide Ansätze werden die Methoden Naïve Bayes, Support Vector Machine, Random Forest, AdaBoost und Gradient Boosting zur Klassifizierung eingesetzt. Dabei werden für jede dieser Methoden zuerst die Performance Metriken für die drei Text-Modelle Bag-of-Words, Bag-of-N-Words und TF-IDF ermittelt. Damit wird eine sogenannte Grundlinie (engl. *Baseline*) geschaffen, um im späteren Verlauf Performance Verbesserungen oder Einbußen durch die Kombination von Text-Modellen mit anderen Features erkennen zu können.

Für die Umsetzung wurde Python und unter anderem folgende Bibliotheken genutzt. *Sqlite3* in Kombination mit *pandas* zur Abfrage und Transformierung der Daten. *Matplotlib.pyplot* zur Erstellung der Grafiken für Kapitel 7 und die wohl wichtigste Bibliothek *scikit-learn*, zur Umsetzung aller Aufgaben im Bereich Text-Klassifikation.

6.1 Datenbasis

Die als Datenbasis genutzte Plattform Reddit ist eine Nachrichten- und Diskussionswebseite, welche sich durch eine Vielzahl an Foren, genannt *Subreddits*, organisiert. Diese dienen als Kategorie für Beiträge, wobei ein Beitrag immer nur einem Subreddit zugeordnet werden kann. Beiträge werden von Nutzern erstellt, um Inhalte wie Text, Bilder, Videos oder externe Links zu teilen. Diese Beiträge können wiederum von anderen Nutzern kommentiert und bewertet werden.

Im Grunde genommen gibt es also keine großen Unterschiede zwischen einem Subreddit und einem klassischen Forum, wie sie zu tausenden im Internet existieren. Wodurch sich Reddit jedoch von anderen Foren abhebt, ist dessen Dimension. Mit Stand 2019

gab es rund 1,5 Millionen solcher Subreddits [47]. In jeden dieser Subreddits werden mit unterschiedlicher Intensität Beiträge und Kommentare erstellt, wodurch eine große Anzahl an Textdaten zu fast jedem erdenklichen Themenbereich generiert wird. Reddit eignet sich daher perfekt als Datenbasis für Projekte im Bereich der Text-Klassifikation. Als Auswahlkriterium für die 10 Kategorien wurde nur eine aktive Gemeinschaft und unterschiedliche Themengebiete vorausgesetzt.

6.2 Datenbeschaffung

Ein weiterer Grund der für Reddit als Datenbasis spricht ist, seine API[1], welche den Zugriff auf Beiträge und Kommentare erleichtert. Für das Abgreifen der Textdaten wurde die Python Bibliothek *Python Reddit API Wrapper*, kurz *PRAW*, genutzt. Diese Art der Datenabfrage funktionierte sehr gut in der Anfangsphase des Projektes. Im späteren Verlauf kam es jedoch zu Problemen da es aufgrund eines fehlenden API Endpoints[2] nicht mehr möglich war, mehr als 100 Beiträge eines Subbreddits abzufragen.

Da diese Anzahl an Beiträgen zu gering für eine aussagekräftige Klassifikation wäre, wurde nach einer Alternative gesucht. Die Lösung für dieses Problem konnte mit Pushshift[3] gefunden werden, welche die Daten von Reddit online zur Verfügung stellt.

```
def getPushshiftData(subreddit, after, before):
  url = 'https://api.pushshift.io/reddit/search/submission?&size=1000&after=' +
  str(after) + '&before=' + str(before) + '&subreddit=' + str(subreddit)
  r = requests.get(url)
  return json.loads(r.text)['data']
```

PRAW wurde jedoch weiterhin eingesetzt um die Kommentare der einzelnen Beiträge abzufragen. Insgesamt wurden Beiträge, welche im Zeitraum Juli 2018 bis Dezember 2018 erstellt wurden für die Analyse herangezogen. Dadurch ergaben sich vor der Aufbereitung der Daten rund 1 Millionen Beiträge mit ca. 3 Millionen Kommentaren. Die Verteilung der Beiträge zu den Subreddits war zu diesem Zeitpunkt leider nicht optimal. Fast 40% der Beiträge stammten aus dem Politikforum, wobei Beiträge zu Büchern oder Essen nur jeweils 2% aller Beiträge ausmachten. Aus diesem Grund wurden für das Training nur 2000 Beiträge pro Subreddit und deren Kommentare genutzt, um für eine gewisse Balance innerhalb der Kategorien zu sorgen. Eine Statistik über die schlussendlich herangezogenen Kommentare pro Subreddit wird in Tabelle 6.1 gezeigt.

Im späteren Verlauf des Projekts wurden weitere Daten zu den Autoren der Kommentare gesammelt. Diese beinhalten Informationen zur aktiven Nutzung der 10 gewählten Subreddits. Mit diesen Daten wurden Ansätze wie in Abschnitt 3.2 erwähnt und im Bezug auf Reddit getestet.

[1] https://www.reddit.com/dev/api/

[2] https://www.reddit.com/r/changelog/comments/6pi0kk/improving_search/dkpx8nu/?context=4

[3] https://pushshift.io/

Tabelle 6.1: Statistik über jeweils 2000 Beiträge pro Subreddit

Subreddit	$\sum$ *Kommentare*	$\varnothing$ *Kommentare pro Beitrag*
Music	56605	28,3
Books	80249	40,1
Food	39448	19,7
Gaming	54457	27,2
Movies	103645	51,8
News	183235	91,6
Personalfinance	35982	18,0
Politics	134074	67,0
Relationships	49846	24,9
Sports	125684	62,8

6.3 Aufbereitung der Daten

Nachdem der Corpus mit 2000 Beiträgen pro Kategorie ausgewählt wurde, konnte mit der Aufbereitung der Daten begonnen werden. Wie bereits in Abschnitt 4.1 angesprochen, ist es ratsam diese im Einklang mit dem Feature Engineering durchzuführen. In diesem Fall wurden unter anderem bestimmte Textpassagen wie das Erwähnen von Autoren oder externe Links erst nachdem sie gezählt wurden entfernt. Spezielle Symbole oder Sonderzeichen wurden jedoch sofort aus dem Text entfernt. Für diesen Schritt kamen hauptsächlich reguläre Ausdrücke (engl. *regular expressions*) zum Einsatz.

```
#Entfernen von Links
text = re.sub('https?(?<=http).*?(?=(\t| |\$))', '', text)
#Entfernen von speziellen Zeichen wie  
text = re.sub('&(?<=&)[^ ]*?(?=;);', '', text)
#Entfernen von Autoren im Text wie /u/Rogocraft
text = re.sub('\/u\/(?<=\/u\/)[^ ]*?(?=\ )', '', text)
#Entfernen von Subreddits im Text wie /r/politics
text = re.sub('\/r\/(?<=\/r\/)[^ ]*?(?=\ )', '', text)
```

Da es sich bei den Kommentaren um englische Texte handelt, wurden auch Abkürzungen wie „*don't*“ zu ihrer ausgeschriebenen Form „*do not*“ umgewandelt. Zusätzlich wurden noch weitere kleine Schritte, wie das Entfernen von unnötigen Leerzeichen zur Aufbereitung der Daten vorgenommen. Zu guter Letzt wurden noch alle Kommentare auf Rechtschreibfehler überprüft und gegebenenfalls ausgebessert.

Das Aufbereiten der Kommentare wirkte sich zwar positiv auf die eingesetzten Text-Modelle aus und konnte weitere Text-Features hervorbringen, jedoch hatte es keinen Einfluss auf textunabhängige Features.

6.4 Features

Um zusätzlich zu den Text-Modellen Features zu finden wurden Kommentare bzw. die Funktionsweise und der Aufbau von Beiträgen innerhalb Reddits genau analysiert. Beiträge und Kommentare können zum Beispiel von Benutzern als positiv oder negativ bewertet und zusätzlich auch eine Auszeichnung verliehen bekommen. Ebenso wie diese kleinen Funktionen spielte auch die Struktur der Beiträge eine wichtige Rolle. Da es sich bei diesem Schritt um wohl einen der wichtigsten der Implementierung handelte, wurde hier sehr viel Zeit investiert. Am Ende wurden insgesamt 30 Features entwickelt, welche in die Kategorien

- Text (10),
- Forum (9) und
- Autor $(10 + 1)$

eingeteilt wurden. Bevor die Features nun gelistet werden ist zu betonen, dass nicht jedes Feature auch im Endeffekt eingesetzt wurde. In Abschnitt 6.5.2 wird das Auswahlverfahren beschrieben, welches bestimmt ob ein Feature nun mit einbezogen wird oder nicht. Die Liste an Features startet mit den Text-Modellen, da es sich bei ihnen um eine Sammlung von vielen Features handelt.

6.4.1 Text-Modelle

Die Text-Modelle dienen in erster Linie als Ausgangspunkt für die Methoden zur Text-Klassifikation, welche in der Implementierung Anwendung finden. Da diese bereits in Abschnitt 2.4 ausführlich beschrieben wurden, wird hier nur auf deren Konfiguration eingegangen.

Wie im Programm 6.1 zu erkennen, wurden für alle Modelle die gleichen Einstellungen für `min_df` und `max_df` gewählt. Diese geben an, wie oft ein Wort mindestens oder maximal vorkommen darf, um als Feature gewertet zu werden. In diesem Fall muss ein Wort mindestens drei Mal und darf maximal in 80% aller Kommentare vorkommen.

Programm 6.1: Konfiguration der Text-Modelle

```
#Bag-of-Words
CountVectorizer(stop_words="english",max_df=0.8,min_df=3)
#Bag-of-N-Words
CountVectorizer(ngram_range=(2,2),stop_words="english",max_df=0.8,min_df=3)
#TF-IDF
TfidfVectorizer(stop_words="english",max_df=0.8,min_df=3)
```

6.4.2 Text-Features

Abgesehen von Text-Modellen wurden auch andere gängige Techniken eingesetzt, um 10 Features aus dem vorhandenen Text zu generieren. Part-of-Speech Tagging und eine Rechtschreibüberprüfung zählen dabei zu den bekanntesten.

Nomen, Pronomen, Verb, Adverb und Adjektiv

Um ein Wort in eine dieser 5 Wortarten zu unterteilen wurde die Python Bibliothek *Textblob*[4] eingesetzt. Da dieser Schritt sehr zeitaufwendig ist, wurde er im Zuge der Aufbereitung durchgeführt. Die Ergebnisse wurden zu den jeweiligen Kommentaren gespeichert. Wie *Textblob* für diese Arbeit eingesetzt wurde, ist in Programm 6.2 zu sehen.

Programm 6.2: Part-of-Speech Tagging mit *Textblob* am Beispiel von Nomen

```
taggedWords = textblob.TextBlob(comment)
noun_count = 0
for tup in taggedWords.tags:
  if tup[1] in ['NN','NNS','NNP','NNPS']:
    noun_count += 1
cursor.execute('''UPDATE Comment SET Noun_Count = (?),WHERE ID  = (?)''',
  (noun_count, ID))
```

Fehlerhaft

Die in Abschnitt 6.3 erwähnte Rechtschreibüberprüfung korrigierte nicht nur den Text, sondern speicherte zusätzlich zu jedem Kommentar ob es fehlerhaft war oder nicht. Zur Rechtschreibüberprüfung können in Python verschiedenste Tools zum Einsatz kommen. Für diese Arbeit wurde *Symspellpy*[5] eingesetzt.

Editiert

Bei der Abfrage von Kommentaren über die Reddit API wird angegeben ob ein Kommentar nach seiner Erstellung editiert wurde. Diese Information konnte 1:1 als Feature übernommen werden. Leider kann dabei aber nicht zwischen einmaligem oder mehrmaligem Editieren differenziert werden.

Benutzer-, Url- und Subreddit-Anzahl

Auch diese Features wurden bereits im Zuge der Textaufbereitung gespeichert. Links zu externen Webseiten, einen anderen Subreddit oder einem Benutzer wurden vor ihrer Entfernung gezählt und zum jeweiligen Kommentar gespeichert.

6.4.3 Forum Features

Aus der im Abschnitt 6.1 angesprochenen Struktur von Beiträgen, ergeben sich für ein Kommentar folgende 9 Features:

Punkte

Ein Kommentar kann von anderen Nutzern positiv oder negativ bewertet werden. Daraus ergibt sich ein Punktestand, welcher dem Kommentar zugewiesen wird. Da dieser

[4] https://textblob.readthedocs.io/en/dev/quickstart.html#part-of-speech-tagging
[5] https://github.com/mammothb/symspellpy

Punktestand auch negativ sein kann, wird hier der absolute Wert als Feature verwendet. Damit die Information, ob der Punktestand positiv oder negativ war, nicht verloren geht, wird dies als zusätzliches Feature verwendet. Tabelle 6.2 zeigt wie die Punkte für drei Kommentare in der Datenbank gespeichert würden. Als Features werden nun *Punkte_Abs* und *Punkte_Positiv* eingesetzt.

Tabelle 6.2: Features für Punkte als Beispiel

Punkte	*Punkte_Abs*	*Punkte_Positiv*
24	24	1
-3	3	0
0	0	1

Auszeichnung

Zusätzlich zu einer Punktebewertung kann ein Kommentar auch von Nutzern ausgezeichnet werden. Laut API würden drei Auszeichnungen (Silber, Gold, Platinum) zur Verfügung stehen. Zum Zeitpunkt der Datenbeschaffung hatte aber kein einziger Kommentar Silber oder Platinum erhalten, weshalb diese Werte ignoriert wurden.

Ebene und Ist_Top

Jeder Kommentar in einem Beitrag befindet sich in einer gewissen Ebene. Wurde das Kommentar direkt als Antwort auf den Beitrag gegeben befindet es sich in der Ebene 0 und damit auch als „Top-Kommentar“ gekennzeichnet. Ist es eine Antwort auf ein Kommentar der Ebene 0, befindet sich dieses in Ebene 1. Es zeigt sich somit das jede Antwort auf einen Kommentar der Ebene n in die Ebene $n + 1$ gereiht wird. Auf einer Ebene sind mehrere Kommentare möglich.

Kinder, direkte Kinder und Geschwister

Durch die Einteilung von Kommentaren in Ebenen, ergibt sich eine gewisse Hierarchie. Ein Kommentar kann mehrere direkte und indirekte untergeordnete Kommentare besitzen. Zusätzlich zu untergeordneten Kommentaren besitzt ein Kommentar auch mehrere Geschwisterkommentare, welche ebenfalls gezählt und als Feature eingesetzt wurden. Zur Ermittlung von direkt und indirekt untergeordneten Kommentaren wurde eine rekursive Funktion eingesetzt.

Die rekursive Funktion, zu sehen in Programm 6.3, startet mit einem Kommentar auf der Ebene 0. Dabei wird die Funktion wieder von allen direkt (*Direct_C*) und indirekt (*Sum_C*) untergeordneten Kommentaren aufgerufen. Alle Informationen zu diesem Kommentarstrang werden dabei in eine globale Variable gespeichert, welche diese nach erfolgreichem Abschluss in die Datenbank speichert. Wichtig für die Performance der rekursiven Funktion war es, einen Index für *Parent_Id* zu erstellen.

Programm 6.3: Rekursive Funktion zur Ermittlung der (direkten) Kinder

```
def count_children(parentId):
  global Comment_Children_Dict
  cursor.execute('''SELECT ID from Comment WHERE Parent_Id like (?)''',
    ('\%' + parentId,))
  children = cursor.fetchall()
  children_count = len(children)

  Comment_Children_Dict[parentId] = {}
  Comment_Children_Dict[parentId]['Direct_C'] = children_count

  if children_count == 0:
    Comment_Children_Dict[parentId]['Sum_C'] = 0
    return 0

  for child in children:
    children_count += count_children(child[0])

  Comment_Children_Dict[parentId]['Sum_C'] = children_count
  return children_count
```

Intervall

Das Intervall gibt die Zeit in Sekunden an, welche zwischen dem Erstellen eines Beitrags und dem Erstellen des Kommentars vergangen ist. Da für jeden Beitrag und Kommentar ein Erstellungsdatum vorliegt, konnte dieses Feature relativ leicht berechnet werden.

6.4.4 Autor Features

Wie bereits in Abschnitt 3.2 erwähnt kann auch das Nutzerverhalten eines Autors als Feature für Text-Klassifikation eingesetzt werden. Deshalb wurde für jeden Autor eine Tabelle angelegt welche die Anzahl der Kommentare per Subreddit beinhaltet. Die Abfrage dieser Daten wurde wieder über die *pushshift.io* Schnittstelle gelöst da hier bereits eine Aggregationsfunktion für Autoren zur Verfügung stand.

```
def getAuthorDataSubreddit(author):
  url='https://api.pushshift.io/reddit/search/comment/?author=' + str(author) + \
  '&subreddit=' + '&subreddit='.join(subredditList) + '&aggs=subreddit'
  r = requests.get(url)
  data = json.loads(r.text)
  return data['aggs']['subreddit']
```

Die erhaltene Verteilung an Kommentaren pro Kategorie durch diesen Autor, wurde zu jedem Kommentar, welches von ihm verfasst wurde, hinzugefügt. Für zusammengefasste Kommentare wurde im weiteren Verlauf noch das zusätzliche „Eindeutige Autoren“ hinzugefügt. Dieses Feature steht für einzelne Kommentare nicht zur Verfügung.

6.5 Trainieren der Methoden

Das Trainieren der Methoden, Naïve Bayes, SVM, Random Forest, AdaBoost und Gradient Boosting kann in drei Etappen

1. *Hyperparameter Tuning*,
2. Feature Auswahl und
3. Training

unterteilt werden. Jede dieser Etappen wird im Detail in den folgenden Abschnitten beschrieben. Dabei werden auch Programmausschnitte in leicht vereinfachter Form genutzt, um zu zeigen wie die Implementierung mit Python umgesetzt wurde. Alle Trainingsläufe wurden nach dem *Kreuzvalidierungsverfahren* durchgeführt. Dabei kam `cross_validate` mit vier *Folds* bzw. `StratifiedKFold` zum Einsatz. `StratifiedKFold` wurde genutzt, um eine gleiche Verteilung der Klassen pro *Fold* zu garantieren.

Für die ersten beiden Etappen wurden Trainingsläufe nur mit einem Teil der Beiträge, nämlich 250 anstatt 2000, durchgeführt um die Laufzeit der einzelnen Methoden gering zu halten. 250 Beiträge wurden als ausreichend angesehen, da Ergebnisse mit dieser Anzahl an Beiträgen nicht stark von den Ergebnissen mit allen Trainingsdaten abweichen. Zum Vergleich der Ergebnisse für die Trainingsläufe beim *Hyperparameter Tuning* oder der Feature Auswahl wurde immer der F_1-Wert genutzt. In den ersten beiden Etappen wird außerdem nicht auf die Unterschiede zwischen einzelnen und zusammengeführten Kommentaren eingegangen, da diese äußerst gering und erst in den letzten beiden Abschnitten wirklich von Bedeutung sind.

6.5.1 Hyperparameter Tuning

Beim *Hyperparameter Tuning* wurden alle fünf Methoden mit den drei Text-Modellen Bag-of-Words, Bag-of-N-Words und TF-IDF trainiert. Die getesteten Werte für die Parameter jeder Methode liegen dabei unter bzw. über den von *scikit-learn* vorgegebenen Standartwerten. Der Einfluss dieser Parameter auf das Ergebnis variiert dabei sehr stark von Methode zu Methode.

```
naive_grid = [{'alpha': [0.01,0.1, 0.5, 1.0]}]
svm_grid = [{'C': [0.1,0.5,1.0], 'max_iter': [1000,2500], 'penalty': ['l1','l2']}]
rand_grid = [{'n_estimators': [50,100,250], 'max_depth': [5,10,25]}]
ada_grid = [{'n_estimators': [50,100,250], 'learning_rate': [0.5,1,1.5]}]
gra_grid = [{'n_estimators': [10,25,50],'learning_rate': [0.05,0.1,0.2]}]
```

Die Einstellungen wurden auch gezielt nach der Funktionsweise jeder Methode, wie sie in Kapitel 5 beschrieben werden, ausgewählt. Für SVMs wurden unterschiedliche Werte für `C` getestet, welcher die Position der Trennlinie bzw.-fläche beeinflusst. Für Random Forest wird eine unterschiedliche Anzahl an Bäumen und eine maximale Tiefe pro Baum getestet. Die Anzahl der Bäume wird auch bei AdaBoost und GradientBoost variiert, ebenso wie der Lernfaktor.

Für die Umsetzung des eigentlichen *Hyperparameter Tuning* wurde `ParameterGrid` genutzt um über alle möglichen Einstellungen zu iterieren. In Abbildung 6.1 wird das Ergebnis für den `alpha` Wert für Naïve Bayes gezeigt. Obwohl die Unterschiede nur

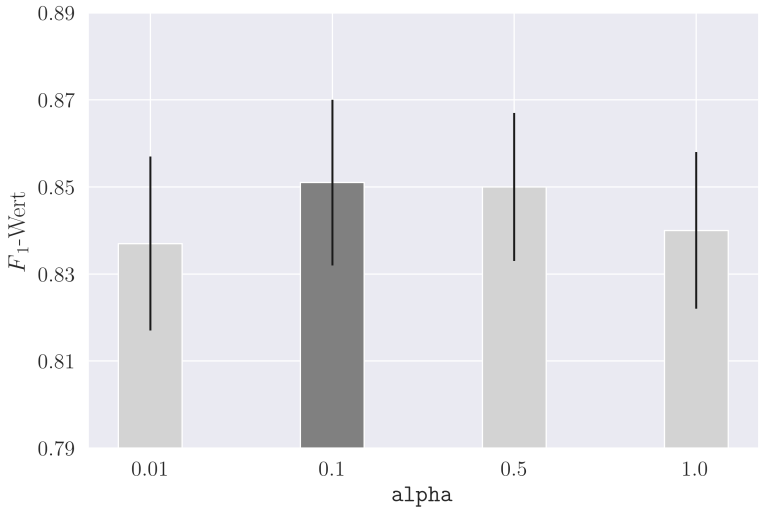


Abbildung 6.1: F_1-Wert von Naïve Bayes mit unterschiedlichen `alpha` Werten

minimal sind und aufgrund der Abweichung kein definitives Ergebnis ermittelt werden kann, wurde hier trotzdem 0.1 als fixer Wert gewählt. Alle weiteren ausgewählten Parameter pro Einstellung und Methode werden im nächsten Kapitel aufgelistet.

6.5.2 Feature Auswahl

Die Feature Auswahl kann noch einmal in zwei Schritte unterteilt werden. Im ersten Schritt wurden die Features nach dem Filter-Ansatz evaluiert. Die dabei selektierten Features wurden in einem weiteren Schritt mit einem Wrapper-Ansatz evaluiert, bevor eine endgültige Auswahl getroffen wurde. Auch hier werden die Ergebnisse zu beiden Verfahren erst im nächsten Kapitel aufgelistet.

Für den Filter-Ansatz wurde jeweils für Text, Forum und Autor Features eine Korrelationsmatrix erstellt. Zu sehen sind diese Matrizen im Anhang A.1. In Abbildung 6.2 ist jedoch ein Teil der Korrelationsmatrix für Forum Features zur Veranschaulichung des Auswahlverfahrens zu sehen. Da die Korrelation zwischen Kategorie und Features über alle Matrizen meist sehr gering war, wurde dies nicht als Ausscheidungsverfahren genutzt. Es wurde nur die Korrelation zwischen Features betrachtet. Eine Korrelation von > 0.5 oder < -0.5 wurde dabei als Limit festgelegt. In Abbildung 6.2 fällt die Korrelation zwischen den Features *Direkte_Kinder* und *Kinder* bzw. *Ebene* und *Ist_Top* in dieses festgelegte Limit. Da *Ist_Top* und *Direkte_Kinder* eine schlechtere Korrelation mit der Kategorie aufweisen, werden diese Features verworfen.

Die aus diesem Verfahren resultierenden Features wurden im nächsten Schritt mit dem Wrapper-Ansatz weiter reduziert. Dabei wurden mehrere, aufeinander aufbauende Tests durchgeführt. Bei der folgenden Beschreibung dieser Tests wird mit „Kategorie“ immer in Bezug auf Features und nicht Beiträgen gesprochen.

Als erstes wurden alle möglichen Feature-Kombinationen, auch *Feature-Sets* genannt, innerhalb der jeweiligen Kategorien Text, Forum und Autor mit allen Methoden und 250 Beiträgen getestet. Davon wurden die besten zwei Feature-Sets pro Kategorie

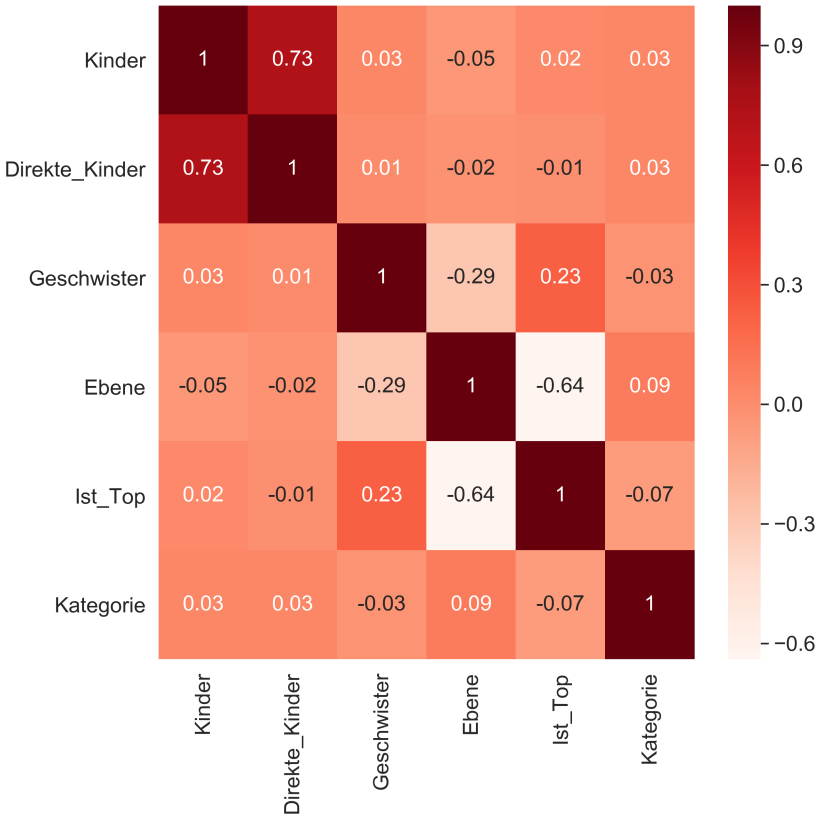


Abbildung 6.2: Teil der Korrelationsmatrix für Forum Features

und pro Methode ausgewählt, Duplikate entfernt, und in einem weiteren Test mit den drei Text-Modellen kombiniert. Dieser letzte Test wurde wieder mit limitierten Beiträgen und allen Methoden durchgeführt. Bei der Evaluierung wurden dabei die besten zwei Feature-Sets pro Kategorie aber über alle Methoden ausgewählt. Dadurch ergeben sich pro Kategorie jeweils zwei Feature-Sets. In Kombination mit den Text-Modellen wurden damit insgesamt 18 unterschiedliche Feature-Kombinationen zum Trainieren der Beiträge eingesetzt.

6.5.3 Kommentare zusammengeführt

Mit dem *Hyperparameter Tuning* und der Feature Auswahl abgeschlossen, kann nun endlich mit den Trainingsläufen gestartet werden. Für die Variante der zusammengeführten Kommentaren war dies relativ simpel umzusetzen. Viele Funktionen werden hier bereits von *scikit-learn* zur Verfügung gestellt. Das wohl beste Beispiel zum Unterschied in der Implementierung mit einzelnen Kommentaren, ist die Funktion `cross_validate`. Diese bietet alle wichtigen Funktionen wie *Kreuzvalidierung* mit korrekt aufgeteilten Gruppen und den drei Metriken, Genauigkeit, Trefferquote und dem F_1-Wert, an. Für das Speichern der Ergebnisse reicht ein einfacher Zugriff auf das von der Funktion zurückgegebene Array `score`. Im Programm 6.4 ist auch gut zu erkennen, dass zusätzlich die Laufzeit jedes Trainings gemessen wurde.

Programm 6.4: Klassifizierung der Kommentare zusammengeführt

```
start_time = time.time()
#Trainieren, Prognosen und Ergebnisse in einem
score = cross_validate(classifier, feature_set, categories, cv=4,
scoring=['f1_weighted'])

end_time = round(time.time() - start_time, 3)
#Ausgabe des F1-Werts für diesen Durchlauf
score['test_f1_weighted'].mean()
```

6.5.4 Kommentare einzeln

Die fünf Methoden mit einzelnen Kommentaren zu trainieren, war um einiges komplexer als bei den Zusammengeführten, da hier fast keine vordefinierten Funktionen genutzt werden konnten. Das wird auch am Vergleich der beiden Programme 6.4 und 6.5 ersichtlich.

Programm 6.5: Klassifizierung der Kommentare einzeln

```
start_time = time.time()
kf = StratifiedKFold(n_splits=4,shuffle=True)
for train_set, test_set in kf.split(threads,labels_all):
  #Trainieren mit allen Kommentaren
  classifier.fit(feature_set[train_set],categories[train_set])

  #Prognose der einzelnen Kommentaren
  for test_index in test_set:
    labels_thread.append(labels[test_index])
    results_soft_voting.append(predict_soft_voting(classifier,
    feature_set[text_index]))

  #Berechnung des F1-Werts für diesen Fold
  score['f1'].append(f1_score(labels_thread,results_soft_voting,average='weighted'))

end_time = round(time.time() - start_time, 3)
#Ausgabe des F1-Werts für diesen Durchlauf
scores['f1'].mean()
```

Hier musste die gesamte Funktionalität von `cross_validate` praktisch nachgebaut werden. Zusätzlich musste auch eine geeignete Evaluierungstechnik implementiert werden. Die Umsetzung dazu findet sich in Programm 6.6, welche sich von *Hard* und *Soft Voting* ableiten lies. Diese Techniken beziehen sich zwar in Abschnitt 5.5 auf das Ergebnis einer Klassifizierung, können aber genauso für diese Aufgabenstellung genutzt werden. Auch hier wurden wieder mit 250 Beiträgen und den Text-Modellen, beide Ansätze evaluiert. Hier ging *Soft Voting* als die leicht bessere Variante als Sieger hervor. An diesem Punkt ist anzumerken, das für `LinearSVC` (die SVM Implementierung von *scikit-learn*) die Funktion `predict_proba`, welche für *Soft Voting* eine Voraussetzung ist, nicht zur Verfügung steht, weshalb in diesem Fall *Hard Voting* zum Einsatz gekommen ist.

Programm 6.6: Umsetzung zu Hard und Soft Voting

```
def predict_hard_voting(classifier,features):
  predictions = classifier.predict(features)
  return pd.Series(predictions).value_counts().index[0]

def predict_soft_voting(classifier,features):
  predictions = classifier.predict_proba(features)
  sumPredictions = pd.DataFrame(predictions).sum(axis=0)
  return classifier.classes_[sumPredictions.idxmax()]
```

Kapitel 7

Resultate

In diesem Kapitel werden nun die Ergebnisse, in der gleichen Reihenfolge zu den Abschnitten im vorherigen Kapitel, präsentiert und analysiert. Alle relevanten Testergebnisse werden dabei als Chart dargestellt. Die Ergebnisse sind auch als *.csv*-Dateien digital verfügbar.

Da mit den zusammengeführten Kommentaren um einiges bessere Ergebnisse erzielt wurden, als mit den einzelnen Kommentaren, beziehen sich die Ergebnisse in Abschnitten 7.1 und 7.2 auf diese Variante. Die Ergebnisse für diese beiden Abschnitte für einzelne Kommentare befinden sich im Anhang A.2 bzw. A.3

7.1 Hyperparameter Tuning

In den folgenden Abbildungen wird die Auswahl der Parameter für jede Methode anhand von Balkendiagrammen dargestellt. Der dunkel eingefärbte Balken repräsentiert dabei die endgültige Auswahl. In machen Situationen (z. B. Abbildung 7.2) waren die Ergebnisse praktisch gleich, sodass die Laufzeit als Auswahlkriterium genommen wurde. Die einzige Ausnahme ist Abbildung 7.4. Hier wurde trotz der besseren Performance von 50 `n_estimators`, 25 gewählt, da die Laufzeit dadurch um einiges verkürzt wurde. Die Ergebnisse von Naïve Bayes finden sich bereits in Abbildung 6.1, weshalb auf ein erneutes Einbinden verzichtet wird.

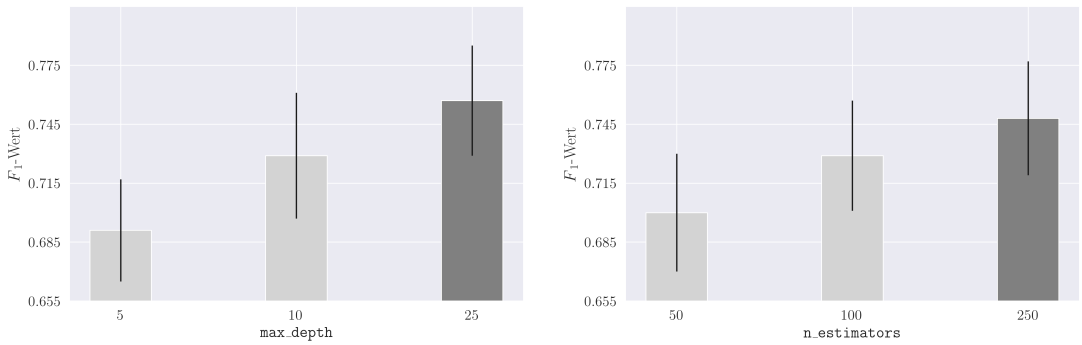


Abbildung 7.1: Random Forest

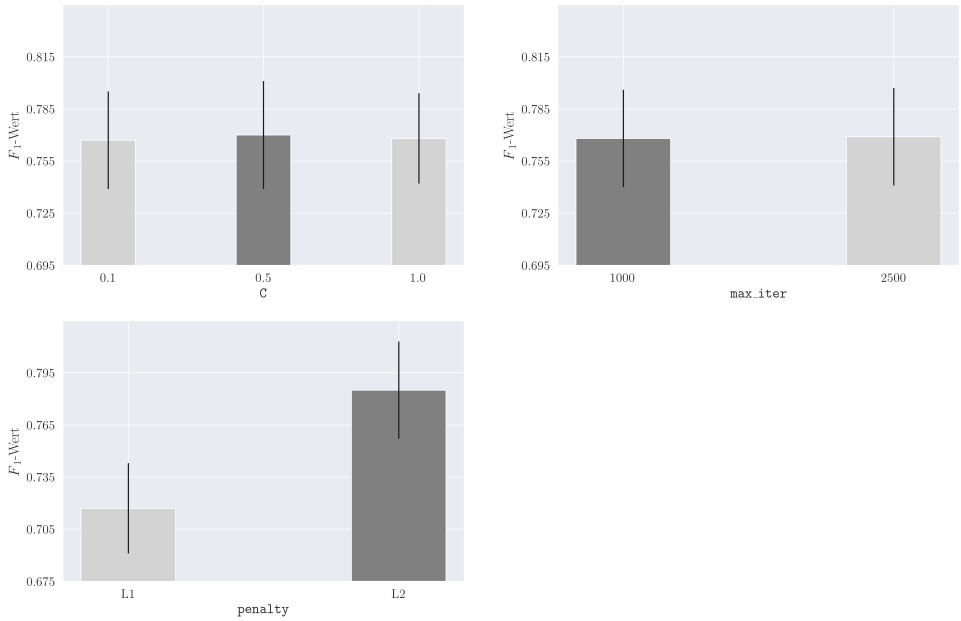


Abbildung 7.2: Support Vector Machine

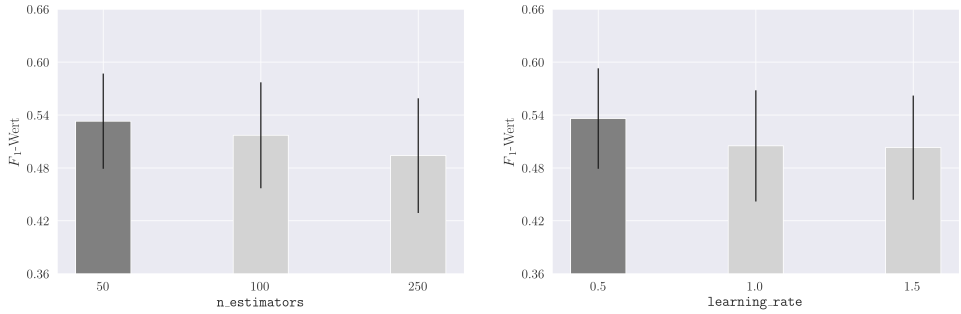


Abbildung 7.3: AdaBoost

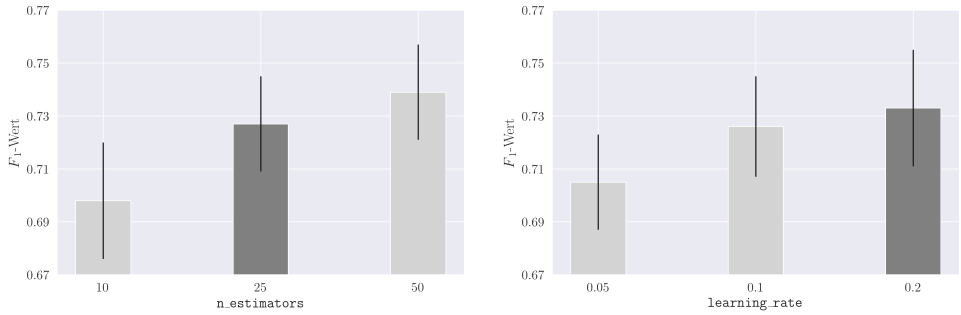


Abbildung 7.4: Gradient Boosting

7.2 Feature Auswahl

In Tabelle 7.1 sind die ausgewählten Features nach dem ersten Verfahren aufgelistet. Anzumerken ist hier die Abwesenheit der Autor Features. Hier gab es keine Korrelation zwischen den Features, welche in das festgelegte Limit gefallen ist, weshalb alle 10 bzw. 11 Autor Features in das zweite Auswahlverfahren aufgenommen wurden. Die Ergebnisse dazu befinden sich in der Tabelle 7.2. Da es sich hier um die Features für zusammengefasste Kommentare handelt, wird das ∅ Symbol angegeben, wenn es sich bei diesem Feature um den Durchschnitt der einzelnen Features handelt. Ist kein Symbol angegeben handelt es sich um die Summe. Für alle *Feature-Sets* werden Abkürzungen mit der jeweiligen Feature-Kategorie und der Anzahl an Features verwendet.

Tabelle 7.1: Ausgewählte Features nach Filter-Ansatz

Text	Forum
Benutzer	Punkte_Abs
Subreddit	Punkte_Positiv
∅ Fehlerhaft	∅ Intervall
∅ Editiert	Ist_Top
Pronomen	∅ Ist_Top
	∅ Kinder

Tabelle 7.2: Ausgewählte Text und Forum Features nach Filter-Verfahren

ID	*Feature-Set*
T_4	Pronomen, Benutzer, Subreddit, ∅ Fehlerhaft
T_5	Pronomen, Benutzer, Subreddit, ∅ Fehlerhaft, ∅ Editiert
F_3	Ist_Top, ∅ Ist_Top, ∅ Kinder
F_5	Punkte_Positiv, ∅ Intervall, Ist_Top, ∅ Ist_Top, ∅ Kinder
A_{10}	Alle 10 Autoren Features
A_{11}	Alle 10 Autoren Features + Eindeutige Autoren

7.3 Kommentare zusammengeführt

In Abbildung 7.5 werden zuerst die Ergebnisse mit den Text-Modellen und den fünf Methoden präsentiert. Hier ist schon einmal zu erkennen, dass alle Methoden bis auf *AdaBoost*, einen relativ guten F_1-Wert mit allen Text-Modellen erreichen. Bag-of-N-Words gilt nach diesem Ergebnis als schlechtes Text-Modell für diesen Anwendungsfall. Was auch auffällt ist, dass sowohl Naïve Bayes als auch SVM einen F_1-Wert von knapp 90% erreichen. Ein Grund für dieses gute Ergebnis könnte könnte an der Auswahl der 10

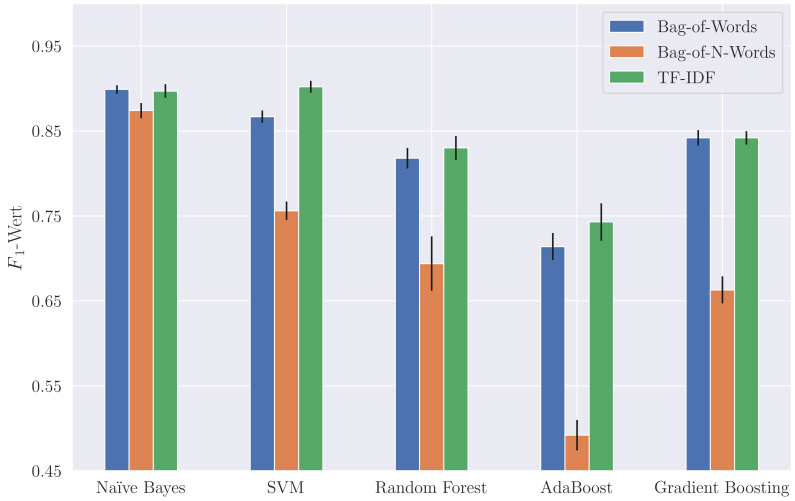


Abbildung 7.5: F_1-Wert für alle Methoden und Text-Modelle

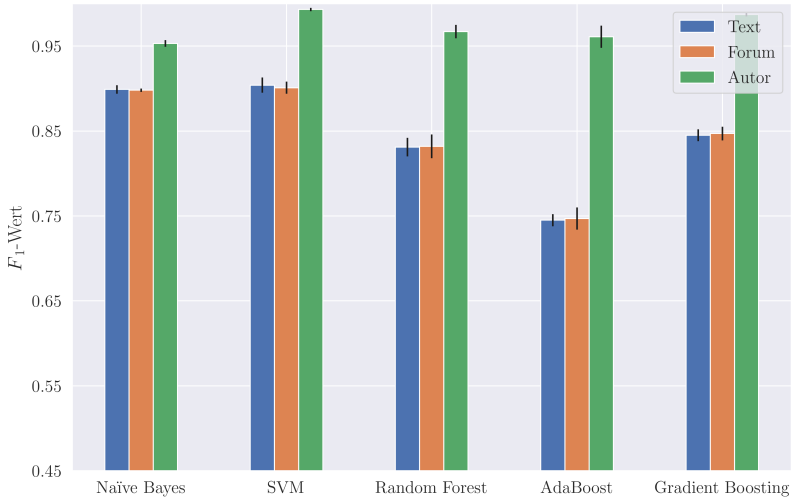


Abbildung 7.6: F_1-Wert für alle Methoden und Text-Modelle + *Feature-Sets*

Kategorien liegen. Diese sind von den einzelnen Themengebieten sehr gut voneinander abgegrenzt.

Bei den Ergebnissen in Abbildung 7.6 zu den Text-Modellen in Kombination mit den *Feature-Sets* gibt es jedoch, was den F_1-Wert betrifft, einen großen Sprung nach oben. Hier können sowohl SVM mit TF-IDF + A_{10} und Gradient Boosting mit Bag-of-Words + A_{10} einen F_1-Wert von 99% erreichen. Aber auch die anderen Methoden erreichen mit Text-Modellen + A_{10} F_1-Werte über der 95% Marke. Text und Forum *Feature-Sets* können zwar die Performance von allen Methoden steigern, erreichen aber auch nur F_1-Werte, welche auch nur mit Text-Modellen allein erzielt wurde.
A_{10} liefert aber nicht nur in der Kombination mit Text-Modellen gute Ergebnisse. Alle Methoden wurden extra noch einmal nur mit A_{10} trainiert und auch hier liefert SVM

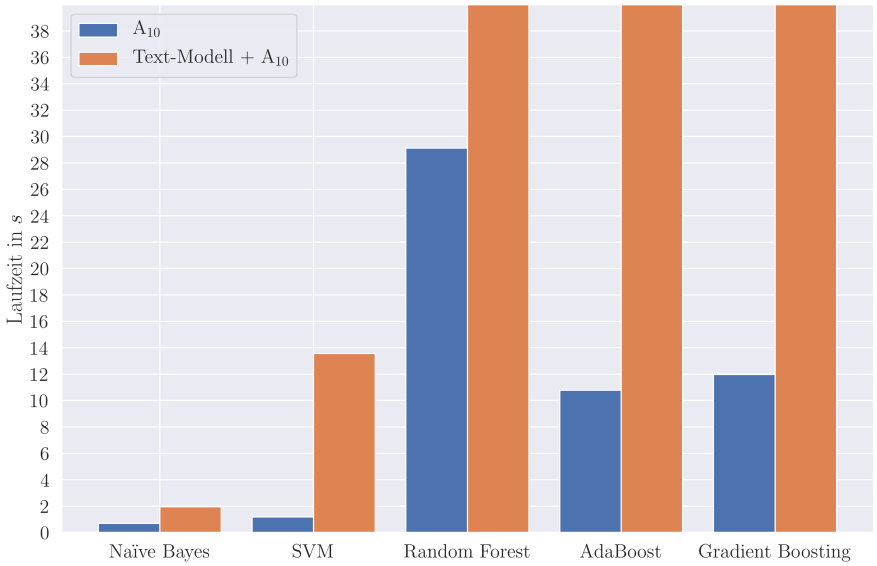


Abbildung 7.7: Vergleich der Laufzeit von A_{10} mit Text-Modell + A_{10}

einen F_1-Wert von 98%. Die anderen Methoden schneiden ähnlich ab. Damit liegt dieser Wert nur 1% unter der Bestmarke, die Trainingsläufe wurden jedoch nur in $\frac{1}{10}$ der Zeit durchgeführt, was definitiv einen großen Unterschied bei noch größeren Datenmengen ausmacht. In Abbildung 7.7 werden die einzelnen Methoden mit dem eigenständigen Trainingslauf von A_{10} und ihrem besten Ergebnis in Kombination mit einem Text-Modell anhand der Laufzeit verglichen. Dabei liegt die Laufzeit von Methoden wie *AdaBoost* oder *Gradient Boosting* weit über 100 Sekunden.

7.4 Kommentare einzeln

Nun zu den Ergebnissen mit einzelnen Kommentaren. Hier ist das Ergebnis wirklich sehr ernüchternd ausgefallen. Der beste F_1-Wert mit 13%, wird durch *AdaBoost* in Kombination mit Autoren Features erreicht und kommt damit nicht einmal annähernd an das Ergebnis mit zusammengeführten Kommentaren an. Die einzelnen Ergebnisse für jede Methode befinden sich in Abbildung 7.8.

Auch die Kombination von Text-Modellen mit den *Feature-Sets* bringt keine Besserung. Aufgrund diesen wirklich sehr niedrigen Ergebnissen (siehe Abbildung 7.9), die häufig sogar schlechter als eine komplett zufällige Prognose sind, ist ein Fehler in der Implementierung leider auch nicht gänzlich auszuschließen.

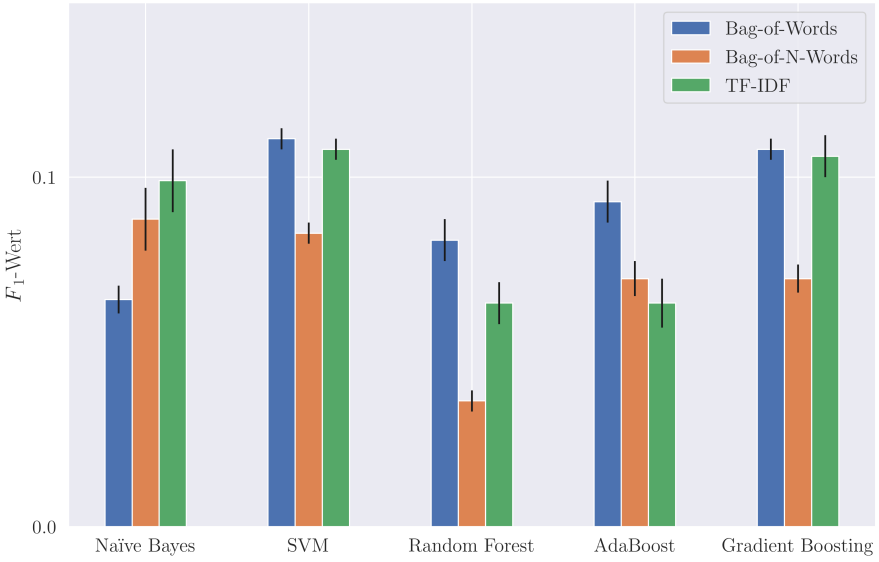


Abbildung 7.8: F_1-Wert für alle Methoden und Text-Modelle

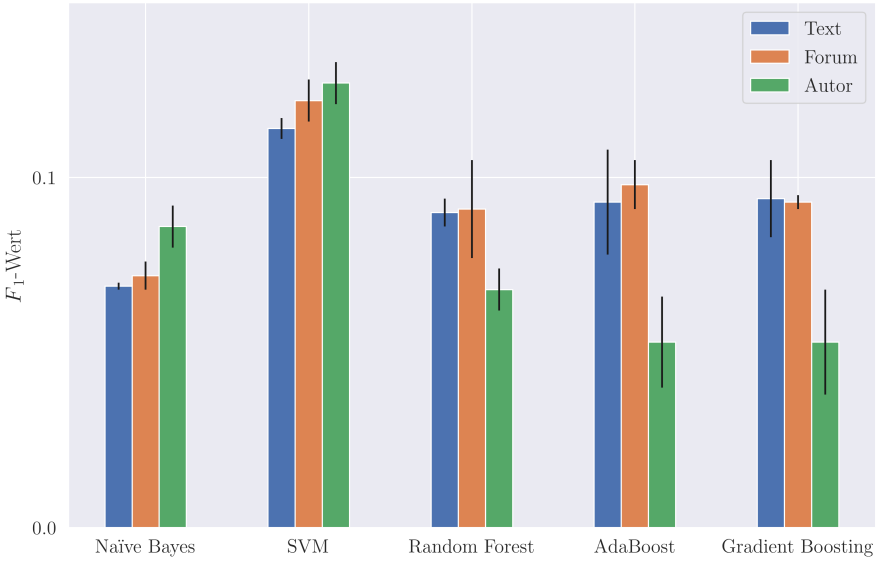


Abbildung 7.9: F_1-Wert für alle Methoden und Text-Modelle + *Feature-Sets*

Kapitel 8

Zusammenfassung

In diesem abschließenden Kapitel wird auf die Ergebnisse, aber insbesondere auf die Herausforderungen und Verbesserungsmöglichkeiten der Umsetzung eingegangen. Ziel dieser Arbeit war es, klassische Text-Modelle wie *Bag-of-Words*, durch die Kombination mit Features, welche sich aus den Eigenschaften eines Forums ableiten lassen, zu verbessern. Dieses Ziel wurde teilweise durch den Einsatz von Features, welche sich aus dem Benutzerverhalten ableiten lassen, erreicht. Trotz einem hohen F_1-Wert von 99%, kann nicht davon ausgegangen werde, dass szenario-spezifische Features immer einen Mehrwert gegenüber klassischen Ansätzen in der Text-Klassifikation bringen. Gerade auch, da ein so hoher Wert meist für eine Überanpassung spricht. Die eingesetzten Testdaten enthalten sehr viel Text, große Unterschiede in den Kategorien und eine aktive Gemeinschaft. Um aussagekräftigere Ergebnisse zu bekommen, sollten in einem weiteren Test Beiträge aus Kategorien gewählt werden, welche sich von der Thematik überschneiden und eine hohe Fluktuation bei den Benutzern erleben. Dadurch sollten die Text-Modelle aber auch die Autoren Features, wesentlich schlechter abschneiden. Es würde sich hier die Frage stellen, ob Forum Features unter diesen Umständen eine deutlichere Verbesserung bringen oder nicht.

Ein weiteres Problem dieser Arbeit ist, dass die Variante mit einzelnen Kommentaren sehr schlecht klassifiziert. Die Ursachen für dieses Problem können vielseitig sein. Entweder ist der Ansatz für das Szenario einfach ungeeignet oder es liegt an der Art wie die Klassifizierung implementiert wurde. Einen Grund zu Verbesserung gibt es definitiv bei der Umsetzung des *Kreuzvalidierungsverfahren*. Das Problem, welches bei einzelnen Kommentaren hier auftritt ist Folgendes. Hier werden nämlich die Beiträge und nicht die Kommentare in *Folds* eingeteilt, was auch der korrekten Vorgehensweise entspricht. Dabei kann es aber trotz `StratifiedKFold` zu einem extremen Ungleichgewicht der Kategorien kommen, wenn sich in einem *Fold* Beiträge mit mehreren hundert Kommentaren befinden. Das Problem wurde dahingehend entschärft, indem ein Limit für Kommentare gesetzt wurde. Auch hier könnte nach einer besseren Lösung, welche im Endeffekt auch die Klassifizierung verbessert, gesucht werden.

Anhang A

Zusätzliche Informationen

A.1 Korrelationsmatrizen zu den jeweiligen Feature-Sets

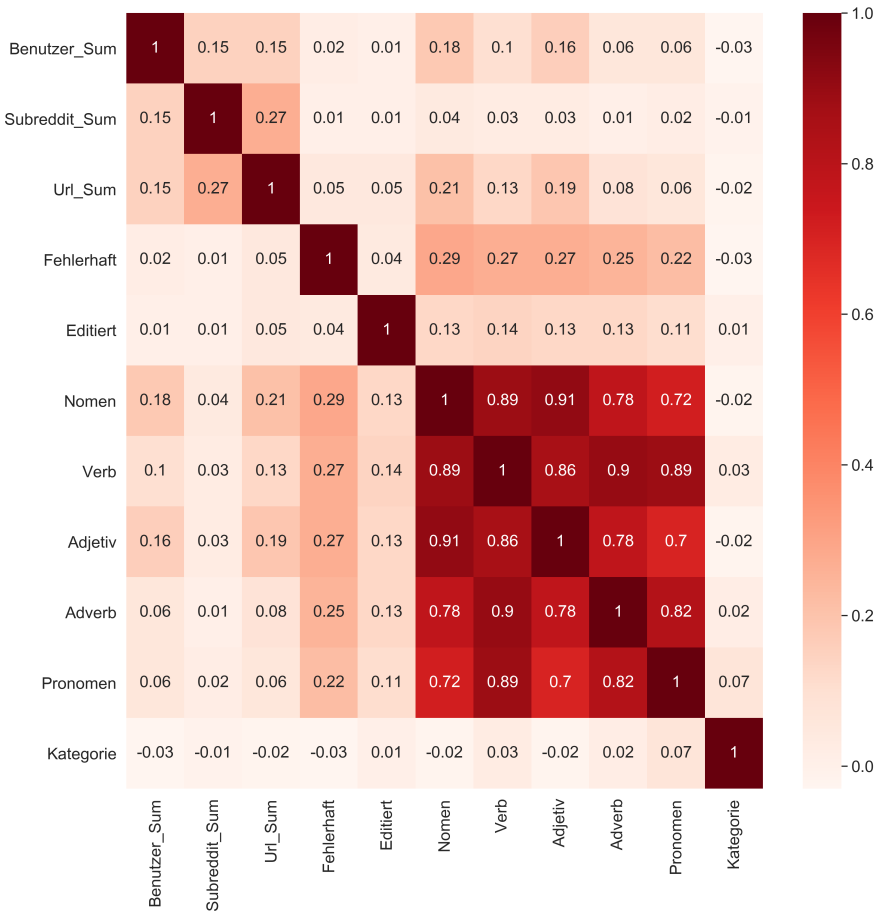


Abbildung A.1: Korrelationsmatrix für Text Features

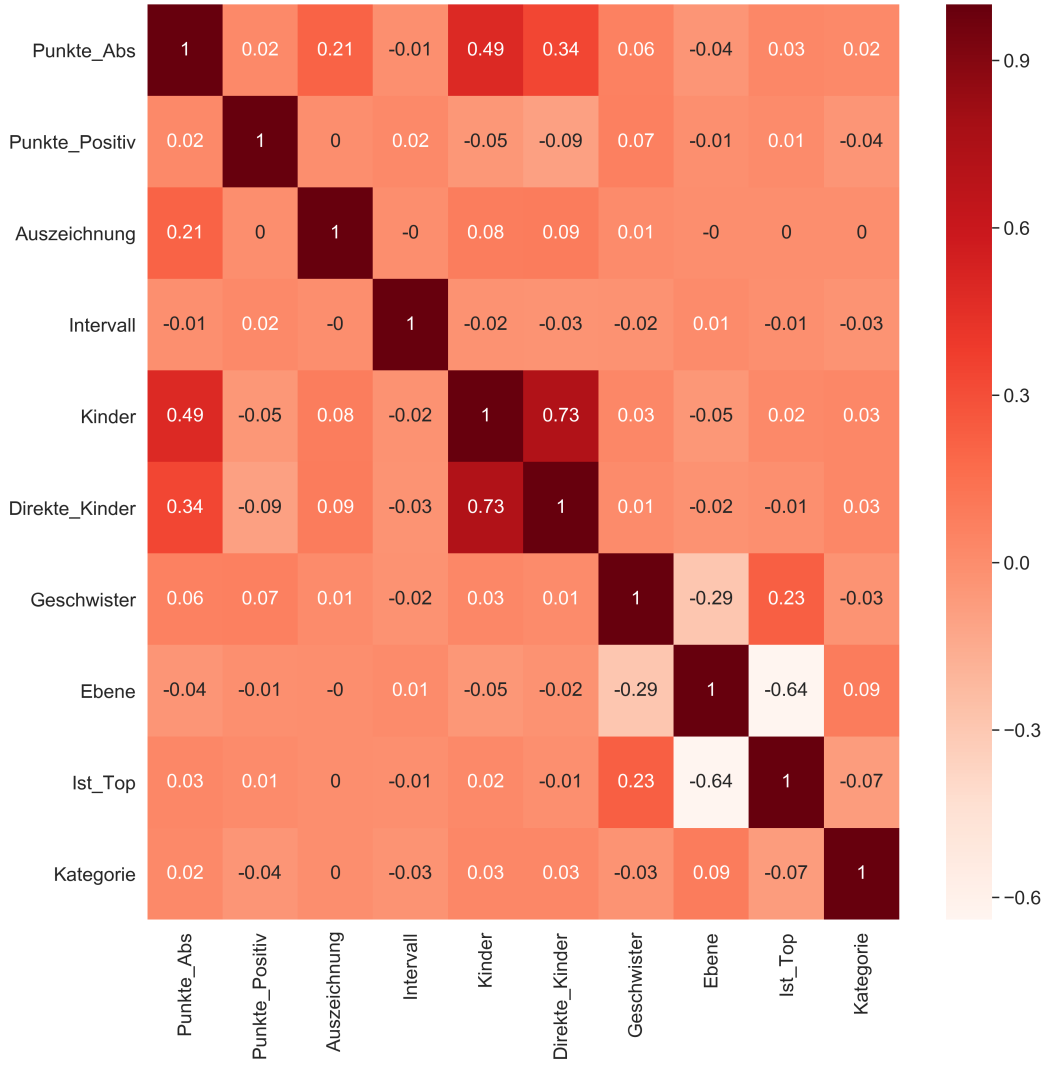


Abbildung A.2: Teil der Korrelationsmatrix für Forum Features

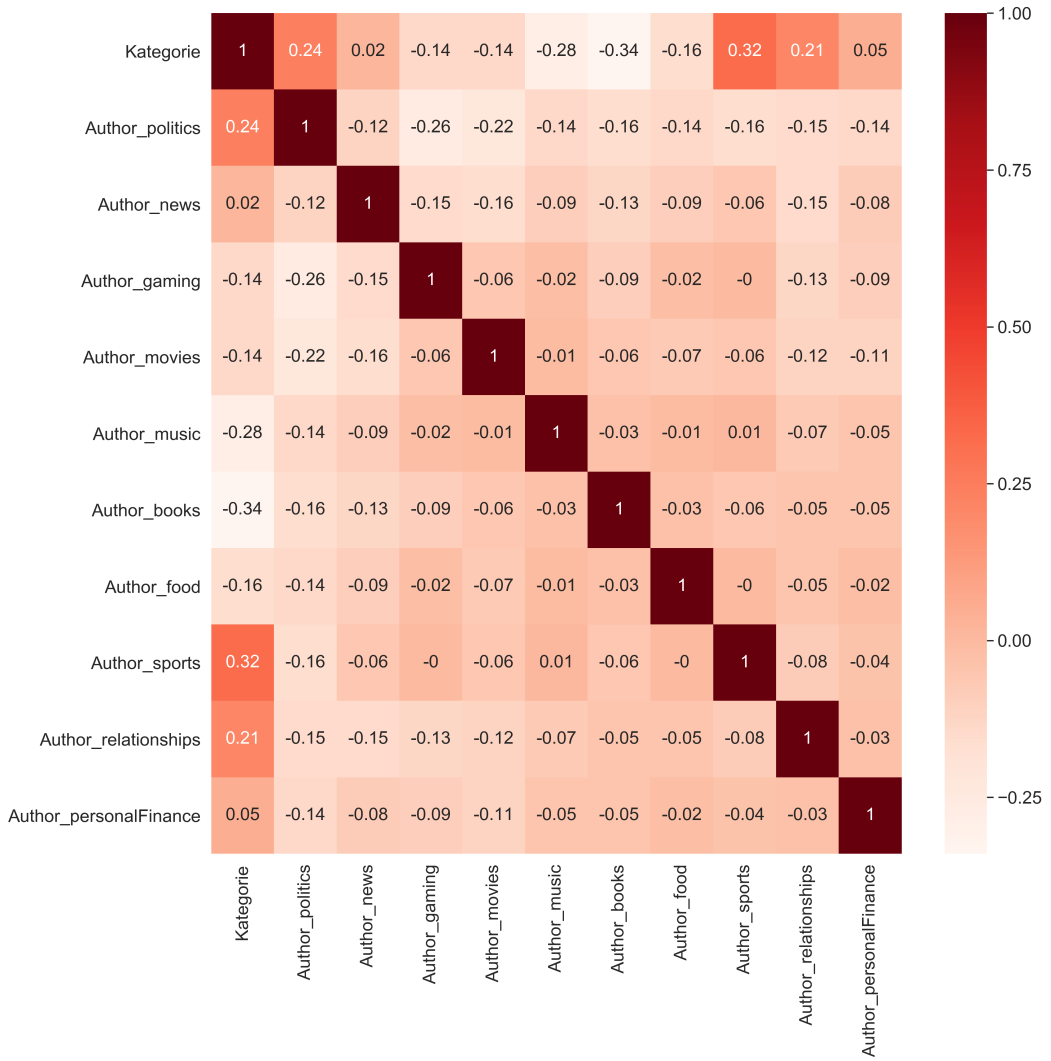


Abbildung A.3: Korrelationsmatrix für Autor Features

A.2 Hyperparameter Tuning - einzelne Kommentare

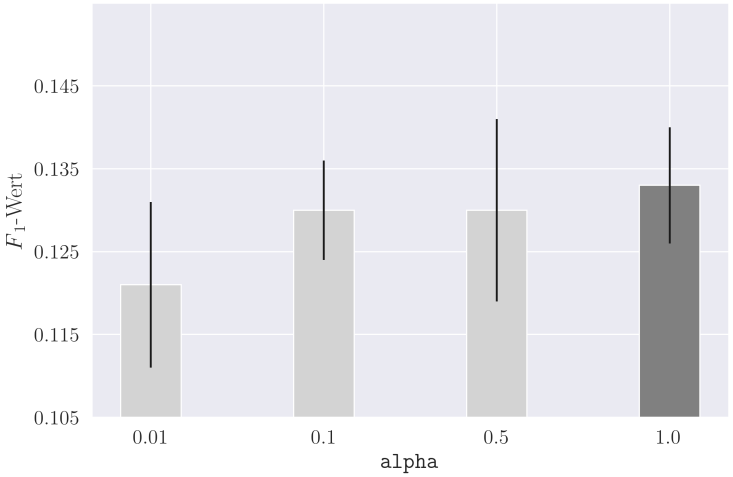


Abbildung A.4: Naïve Bayes

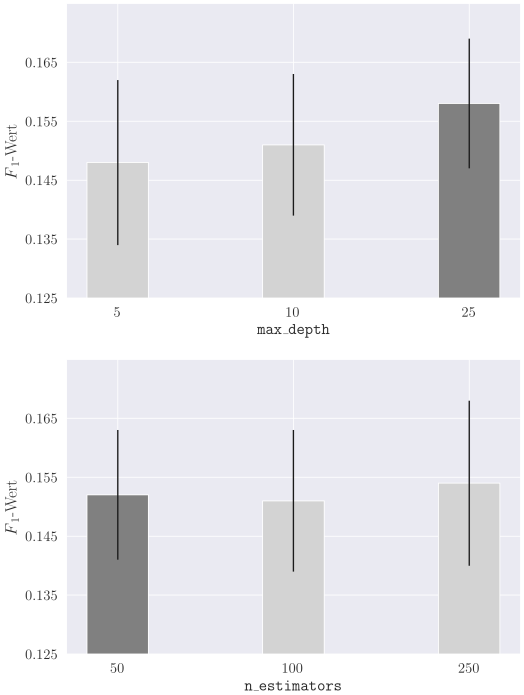


Abbildung A.5: Random Forest

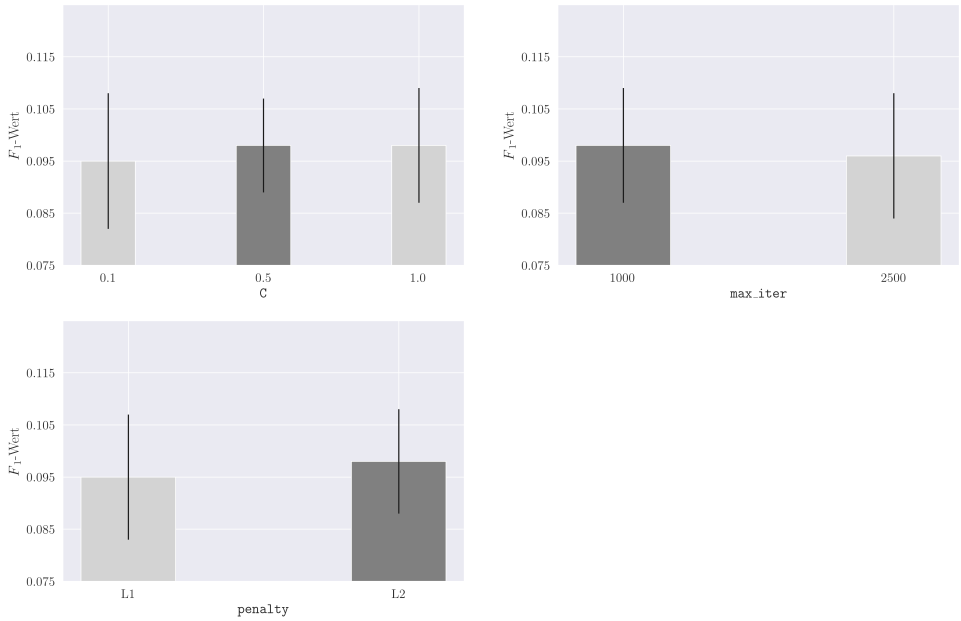


Abbildung A.6: Support Vector Machine

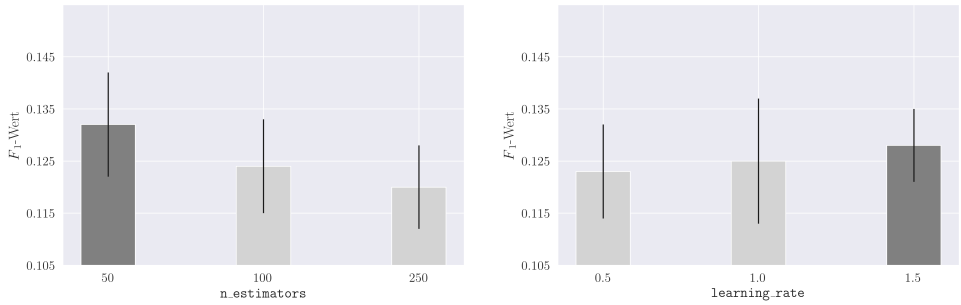


Abbildung A.7: AdaBoost

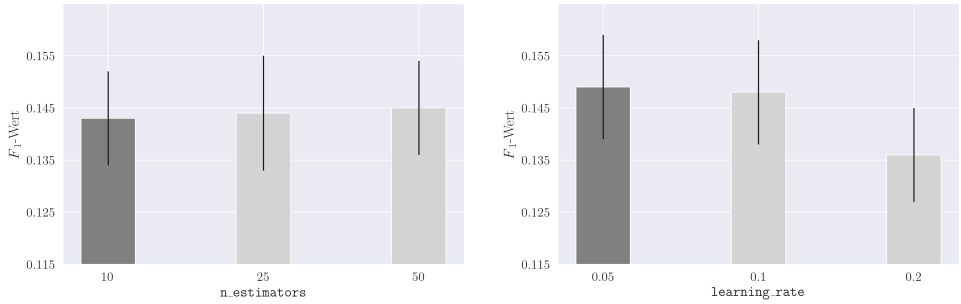


Abbildung A.8: Gradient Boosting

A.3 Feature Auswahl - einzelne Kommentare

Tabelle A.1: Ausgewählte Features nach Filter-Ansatz

Text	Forum
Benutzer	Punkte_Abs
Subreddit	Punkte_Positiv
Links	Intervall
Fehlerhaft	Auszeichnung
Editiert	Ebene
Pronomen	Kinder
	Geschwister

Tabelle A.2: Ausgewählte Text und Forum Features nach Filter-Verfahren

ID	*Feature-Set*
T_3	Fehlerhaft, Editiert, Pronomen
T_2	Editiert, Pronomen
F_2	Ebene, Geschwister
F_5	Punkte_Positiv, Auszeichnung, Punkte_Abs, Intervall, Ebene
A_7	Alle Autoren Features außer *Sport, PersonalFinance, Relationships*
A_7	Alle Autoren Features außer *Politik, PersonalFinance, Relationships*

Anhang B

Inhalt der CD-ROM/DVD

Format: CD-ROM, Single Layer, ISO9660-Format

B.1 PDF-Dateien

Pfad: /

_thesis_oehlinger.pdf .	Masterarbeit

B.2 Implementierung

Pfad: /project

sourcecode.zip	Python Files
datasource.zip	Datenbanken

Pfad: /results

results_single.zip	Ergebnisse Kommentare einzeln
results_multiple.zip . .	Ergebnisse Kommentare zusammengeführt

B.3 Sonstiges

Pfad: /

_online_sources.zip . .	Zitierte Online Quellen

Quellenverzeichnis

Literatur

[1] Hadeer Ahmed, Issa Traore und Sherif Saad. „Detection of online fake news using N-gram analysis and machine learning techniques“. In: *International Conference on Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments*. Springer, 2017, S. 127–138 (siehe S. 13).

[2] Baharum Baharudin u. a. „A Review of Machine Learning Algorithms for Text-Documents Classification“. *Journal of Advances in Information Technology* 1 (2010), S. 4–20 (siehe S. 11).

[3] Bengfort Benjamin, Rebecca Bilbro und Ojeda Tony. *Applied Text Analysis with Python. Enabling Language-Aware Data Products with Machine Learning*. O'Reilly, 2018 (siehe S. 6, 8, 9).

[4] Harold Borko und Myrna Bernick. „Automatic Document Classification“. *Journal of the ACM* 10.2 (1963), S. 151–162 (siehe S. 11).

[5] William W. Cohen. „Learning to Classify English Text with ILP Methods“. In: *Advances in ILP*. IOS Press, 1995, S. 124–143 (siehe S. 17).

[6] Xige Dang u. a. „An Improved Multi-classification Algorithm for Imbalanced Online Public Opinion Data“. In: *Artificial Intelligence and Security*. Springer, 2019, S. 57–66 (siehe S. 12).

[7] Matthias Dimter. „On text classification“. *Discourse and literature* 3 (1985), S. 215–225 (siehe S. 13).

[8] Sarkar Dipanjan, Bali Raghav und Sharma Tushar. *Practical Machine Learning with Python. A Problem-Solver's Guide to Building Real-World Intelligent Systems*. Apress, 2017, S. 26–27 (siehe S. 8).

[9] Susan Dumais u. a. „Using SVMs for text categorization“. *IEEE Intelligent Systems* 13 (1998), S. 21–23 (siehe S. 12).

[10] Eibe Frank und Remco R. Bouckaert. „Naive Bayes for Text Classification with Unbalanced Classes“. In: *Knowledge Discovery in Databases*. Springer, 2006, S. 503–510 (siehe S. 23).

[11] Andrew Giel, Jonathan NeCamp und Hussain Kader. *CS 229: r/Classifier-Subreddit Text Classification*. Techn. Ber. Stanford, CA: Stanford University, 2015 (siehe S. 13).

[12] Kevin Gimpel u. a. *Part-of-Speech Tagging for Twitter: Annotation, Features, and Experiments*. Techn. Ber. Pittsburgh, PA: Carnegie Mellon University, 2011 (siehe S. 13).

[13] Mykhailo Granik und Volodymyr Mesyura. „Fake news detection using naive Bayes classifier“. In: *IEEE First Ukraine Conference on Electrical and Computer Engineering*. IEEE, 2017, S. 900–903 (siehe S. 13).

[14] Alex Graves und Jürgen Schmidhuber. „Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks“. In: *Proceedings of the 21st International Conference on Neural Information Processing Systems*. Curran Associates Inc., 2008, S. 545–552 (siehe S. 11).

[15] Jacqueline Gutman und Richard Nam. *Text classification of reddit posts*. Techn. Ber. New York, NY: New York University, 2015 (siehe S. 13).

[16] Thorsten Joachims. „Text categorization with Support Vector Machines: Learning with many relevant features“. In: *Machine Learning: ECML-98*. Springer, 1998, S. 137–142 (siehe S. 11).

[17] Ramakanth Kavuluru u. a. „Classification of helpful comments on online suicide watch forums“. In: *Proceedings of the 7th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*. ACM, 2016, S. 32–40 (siehe S. 13, 14).

[18] Ron Kohavi und Foster Provost. „Glossary of Terms“. *Machine Learning* 2 (1998), S. 271–274 (siehe S. 7, 9).

[19] Siwei Lai u. a. „Recurrent Convolutional Neural Networks for Text Classification“. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI Press, 2015, S. 2267–2273 (siehe S. 11).

[20] Junze Li u. a. „Wikipedia Based Short Text Classification Method“. In: *Database Systems for Advanced Applications*. Springer, 2017, S. 275–286 (siehe S. 12).

[21] Y. H. Li und A. K. Jain. „Classification of Text Documents“. *The Computer Journal* 41.8 (1998), S. 537–546 (siehe S. 11).

[22] M. E. Maron. „Automatic Indexing: An Experimental Inquiry“. *Journal of the ACM* 8.3 (1961), S. 404–417 (siehe S. 11).

[23] Tomas Mikolov u. a. „Efficient Estimation of Word Representations in Vector Space“. In: *Proceedings of the International Conference on Learning Representations*. ICLR, 2013, S. 1–12 (siehe S. 12).

[24] Fahim Mohammad. „Is preprocessing of text really worth your time for online comment classification?“ In: *Proceedings of the International Conference on Artificial Intelligence*. CSREA Press, 2018, S. 447–453 (siehe S. 12).

[25] Alessandro Moschitti und Roberto Basili. „Complex Linguistic Features for Text Classification: A Comprehensive Study“. In: *Proceedings of the 26th European Conference on Information Retrieval*. Bd. 2997. 2004, S. 181–196 (siehe S. 12).

[26] Tony Mullen und Nigel Collier. „Sentiment Analysis using Support Vector Machines with Diverse Information Sources.“ In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. 2004, S. 412–418 (siehe S. 12).

[27] Andreas C. Müller und Sarah Guido. *Introduction to Machine Learning with Python. A Guide for Data Scientists*. O'Reilly, 2016 (siehe S. 23).

[28] Jasmina Novakovic, Perica Strbac und Dusan Bulatović. „Toward Optimal Feature Selection Using Ranking Methods And Classification Algorithms". *Yugoslav Journal of Operations Research* 21 (2011), S. 119–135 (siehe S. 20).

[29] Patrick Pantel und Dekang Lin. „SpamCop: A Spam Classification and Organization Program". In: *Learning for Text Categorization*. AAAI Press, 1998, S. 95–98 (siehe S. 1, 11).

[30] Jason D. M. Rennie u. a. „Tackling the Poor Assumptions of Naive Bayes Text Classifiers". In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*. AAAI Press, 2003, S. 616–623 (siehe S. 11).

[31] Sam Scott und Stan Matwin. „Feature Engineering for Text Classification". In: *Proceedings of the Sixteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., 1999, S. 379–388 (siehe S. 17).

[32] C Slamet u. a. „Web Scraping and Naïve Bayes Classification for Job Search Engine". *IOP Conference Series: Materials Science and Engineering* 288 (2018), S. 12–38 (siehe S. 11).

[33] Bharath Sriram u. a. „Short Text Classification in Twitter to Improve Information Filtering". In: *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2010, S. 841–842 (siehe S. 13, 15).

[34] Jee Ian Tam. *Classifying Reddit comments by subreddit*. Techn. Ber. Stanford, CA: Stanford University, 2017 (siehe S. 13, 14).

[35] Takenobu Tokunaga und Iwayama Makoto. „Text Categorization Based on Weighted Inverse Document Frequency". In: *Technical Report 94 TR0001*. Tokyo Institute of Technology, 1994, S. 33–39 (siehe S. 18).

[36] Giorgio Valentini und Francesco Masulli. „Ensembles of Learning Machines". In: *Neural Nets*. Springer, 2002, S. 3–20 (siehe S. 11).

[37] Tim Weninger, Xihao Avi Zhu und Jiawei Han. „An Exploration of Discussion Threads in Social News Sites: A Case Study of the Reddit Community". In: *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. ACM, 2013, S. 579–583 (siehe S. 13).

[38] Alice Zheng und Amanda Casari. *Feature Engineering for Machine Learning Models. Principles and Techniques for Data Scientists*. O'Reilly, 2018 (siehe S. 7).

[39] Zhi-Hua Zhou. *Ensemble Methods. Foundations and Algorithms*. Chapman und Hall/CRC, 2012 (siehe S. 26).

Online-Quellen

[40] *Clustering in Machine Learning*. URL: https://www.geeksforgeeks.org/clustering-in-machine-learning/ (besucht am 26. 08. 2019) (siehe S. 6).

[41] *Cross-Validation in Machine Learning*. URL: https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f (besucht am 08. 09. 2019) (siehe S. 8).

[42] *Decision Trees*. URL: https://www.kdnuggets.com/2019/08/understanding-decision-trees-classification-python.html (besucht am 11. 01. 2020) (siehe S. 26).

[43] *Definition of Machine Learning*. URL: https://wirtschaftslexikon.gabler.de/definition/machine-learning-120982 (besucht am 08. 01. 2020) (siehe S. 4).

[44] *K-nearest neighbor*. 2009. URL: http://scholarpedia.org/article/K-nearest_neighbor (besucht am 22. 08. 2019) (siehe S. 23).

[45] *MinMaxScaler*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html (besucht am 10. 01. 2020) (siehe S. 19).

[46] Gil Press. *Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says*. 2016. URL: https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#744c1c316f63 (siehe S. 7).

[47] *Reddit Metrics*. URL: https://redditmetrics.com/history (besucht am 15. 12. 2019) (siehe S. 30).

[48] *Support Vector Machines*. URL: https://www.kdnuggets.com/2019/09/friendly-introduction-support-vector-machines.html (besucht am 11. 01. 2020) (siehe S. 24).

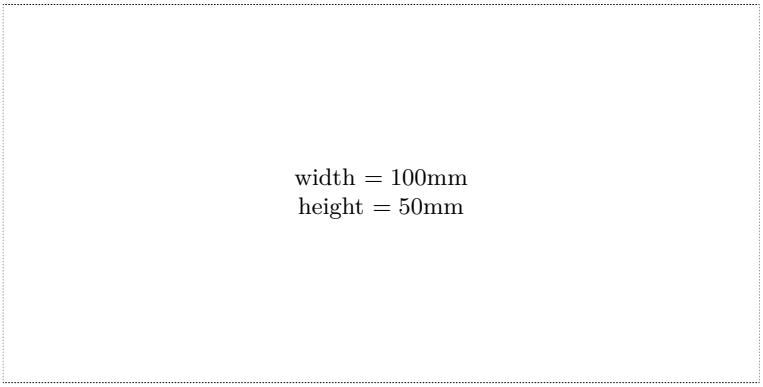
[49] *Tactics to Combat Imbalanced Classes*. Aug. 2015. URL: https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/ (besucht am 26. 08. 2019) (siehe S. 4).

[50] *Text Normalisierung*. URL: https://deacademic.com/dic.nsf/dewiki/2525629 (besucht am 05. 01. 2020) (siehe S. 16).

[51] *Topic Modeling*. 2018. URL: https://fortext.net/routinen/methoden/topic-modeling (besucht am 26. 08. 2019) (siehe S. 6).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —