TECHNISCHE HOCHSCHULE MITTELHESSEN
THM | CAMPUS GIESSEN | MNI
Mathematik, Naturwissenschaften und Informatik

Thesis

Die optimierte Erkennung von Hassrede in digitalen Medien durch sprachspezifisches Preprocessing in Sentimentanalysen

von

Kimberly-Annalena Munjal

7. Januar 2023

Referent: Prof. Dr. Peter Kneisel

Korreferent: Kevin Linne MSc.

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate deutlich kenntlich gemacht zu haben.

Gießen, den 9. Januar 2023 Kimberly-Annalena Munjal

Die Erkennung von Beleidigungen in digitaler Kommunikation ist immernoch eine Hürde, vor allem in weniger verbreiteten Sprachen. Während für Englisch zahlreiche Ressourcen bereitstehen, gibt es für die deutsche Sprache nur vereinzelte Module. Dies gilt auch für den Bereich des Preprocessings im Natural Language Processing, dem Anwendungsgebiet für Textverarbeitung in der künstlichen Intelligenz. Um zu zeigen, dass mit einem sprachspezifischen Preprocessing die Qualität der Ergebnisse von KI-Aufgaben erheblich verbessert werden kann, wird am Beispiel einer Sentimentanalyse ein experimenteller Vergleich von verschiedenen Kombinationen an Preprocessing Schritten durchgeführt. Dazu werden die bestehenden Ressourcen mit dem Wissen über die linguistischen Eigenheiten der Sprache kombiniert, um ein optimiertes Preprocessing als Pipeline bereitzustellen. Diese wird an vier supervised Modellen des Machine Learnings getestet: SVM, XGBoost, Logistic Regression und Naive Bayes. Es stellt sich heraus, dass die Pipeline für die meisten Modelle für Verbesserung des Endergebnisses sorgt, vor allem das spachabhängige Lemmatisieren hat sich positiv ausgewirkt.

Recognizing insults in digital communication is still a hurdle, especially in less common languages. While numerous resources are available for English, there are only isolated modules for German. This is also true for the area of preprocessing in Natural Language Processing, the application area for text processing in artificial intelligence. In order to show that language-specific preprocessing can significantly improve the quality of the results of AI tasks, an experimental comparison of different combinations of preprocessing steps is performed using the example of a sentiment analysis. For this purpose, existing resources are combined with knowledge about the linguistic peculiarities of the language to provide an optimized preprocessing pipeline. This is tested on four supervised models of machine learning: SVM, XGBoost, Logistic Regression, and Naive Bayes. It turns out that for most models the pipeline improves the final result, especially the language-dependent lemmatization has a positive effect.

Inhaltsverzeichnis

1 Einführung

Um Hatespeech in digitalen Medien zu erkennen, git es für die deutsche Sprache erheblich weniger Systeme, als fürs Englische. Daher wird in dieser Arbeit ein relevanter, sprachspezifischer Schritt in dem Prozess von einer konkreten NLP-Aufgabe optimiert, um die Forschung dahingehend voranzubringen: das Preprocessing.

Im Rahmen der Projektphase wurde für ein bestehendes Chat-System ein Modul entwickelt, welches vulgäre Ausdrücke[1] erkennt und Optionen zum Umgang mit diesen anbietet. Im Kontext der firmeninternen Kommunikation soll dadurch für einen seriösen Umgang miteinander gesorgt werden, aber auch außerhalb einer Firmenumgebung findet das Modul Anwendung. Beispielsweise soll in externen Social-Media-Communities erkannt und automatisch gemeldet werden, wenn die Kommunikation unfreundlich ausfällt. Um diesen Prozess weiter zu automatisieren eignet sich ein Ansatz mit künstlicher Intelligenz.

In dem Teilgebiet des *Natural Language Processings* (NLP) gibt es Modelle für jegliche Auswertungen von Daten aus der natürlichen Sprache [2]. Eine typisches Modell für die Erfassung der Atmosphäre von Texten ist die *Sentimentanalyse*, mit welcher die Stimmung eines Textes analysiert und als konkreter Wert eingeordnet werden kann. Der auszuwertende Text wird in einem Vorverarbeitungsschritt vorbereitet und anschließend in Vektoren übersetzt. Mit dem konvertierten, numerisch repräsentierten Text können nun Auswertungen gemacht werden.

1 Schimpfwörter
2 Eine von Menschen gesprochene Sprache, z. B. Englisch, Portugiesisch, Deutsch

1.1 Motivation

Viele Chatsysteme verwenden bereits Filter, um unangemessene Wörter zu erkennen. Diese werden zum Beispiel zensiert oder so eingesetzt, dass ein Abschicken solcher Nachrichten verhindert wird. Sobald der Inhalt der Textnachrichten allerdings eine andere Sprache als Englisch ist, werden die Ergebnisse fehlerhaft.

Szenario: Ein Chatteilnehmer beschreibt einen Bug im firmeninternen Programm als *Dickes Problem*. Ein typisches Profanitäts-Tool würde daraus möglicherweise erkennen, dass das Wort *dick* enthalten ist, welches in der englischen Sprache unangemessen in einer seriösen Umgebung wäre. Eine weitere Möglichkeit wäre, dass er das Wort wegen der Endung *-es* gar nicht erkennt.

Um auch in deutschsprachigen Netzwerken und Communities eine zuverlässige Performance von diesen Erkennungs-Programmen zu gewährleisten, bedarf es einiger Anpassungen in der Vorverarbeitung, dem sogenannten *Preprocessing*. Bisherige Forschung hat gezeigt, dass der Erfolg beim Machine Learning vom Preprocessing positiv beeinflusst werden kann. In dem Resultat von Sentimentanalysen können schon wenige Prozente mehr in der Genauigkeit für ein fundamental besseres Ergebnis in der Erkennung sorgen [Gar22] [CC18]. Mit sprachspezifischem Preprocessing sollen demnach solche Fehler wie im Szenario – im NLP gern *False Positives* genannt – vermieden werden.

1.2 Problembeschreibung

Für die englische Sprache gibt es bei der Erkennung von profanen Ausdrücken – der Popularität und Ressourcenmenge wegen – weit mehr Systeme als für die deutsche. Obwohl, zumindest im europäischen Raum, Deutsch die führende Muttersprache ist und dementsprechend eine Relevanz besteht [Dew]. Dadurch kommt die Frage auf, ob und inwiefern für die deutsche Sprache eine passende Preprocessing-Pipeline anders aussehen könnte. Um die wenigen, unvollständigen Ressourcen für die deutsche Sprache in der Textanalyse weiterzuentwickeln, soll in dieser Arbeit auf ein sprachspezifisches Preprocessing hingearbeitet werden. Deutsch ist mit Englisch verwandt, das heißt sie haben eine gleiche Sprachfamilie, eine verwandte Grammatik und ein ähnliches Alphabet [Bru04]. Somit können von einigen existierenden Beispielen ausgehend Tests gemacht und Vergleiche zu englischen Modellen gezogen werden, um dahingehend ein Konzept zu erarbeiten. Damit wird eine Preprocessing-Pipeline konzipiert, welche dabei hilft, deutsche profane Ausdrücke besser zu erkennen als mit existierenden Methoden.

Das Preprocessing ist der erste Schritt in einem typischen NLP-Workflow und umfasst je nach Modell spezifische Schritte wie beispielsweise *Tokenization*, *Lemmatisierung*

und die Entfernung von aussagelosen Zeichen und Wörtern (siehe 1.1). Im *Information Retrieval* haben diese Schritte eine Steigerung in der Systemleistung gezeigt [Dal21]. Dafür gibt es bestimmte Abläufe, Methoden und Algorithmen, die zur Vorverarbeitung bei Texten angewandt werden [Van13].

Dieser Workflow wird für die natürliche Sprache Deutsch untersucht, sowie die Wichtigkeit für eine solche Spezifizierung herausgearbeitet. Die Erkenntnisse können besonders für den Bereich der multilingualen Textverarbeitung und -analyse wertvoll sein und im Allgemeinen zu akkurateren Ergebnissen führen.

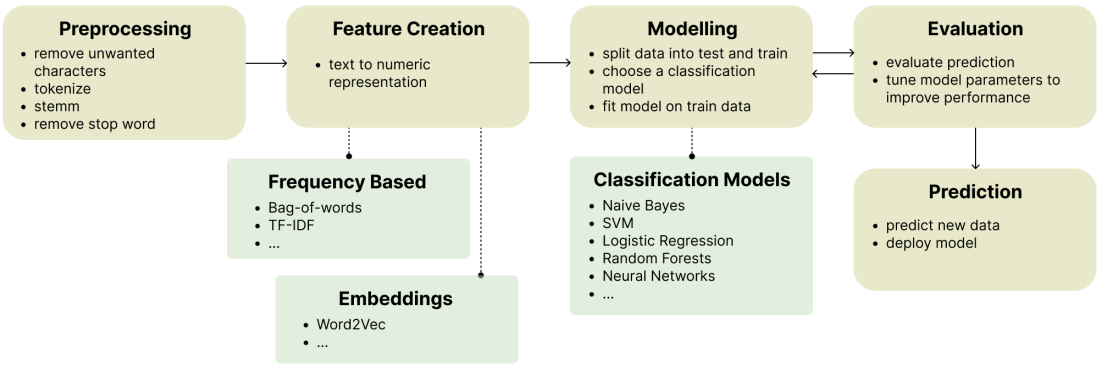

Abbildung 1.1: NLP-Workflow [Mic20]

1.3 Zielsetzung

Ziel dieser wissenschaftlichen Arbeit ist, die Unterschiedshypothese zu belegen, dass Sprach-, Kontext- und Domänenspezifisches Preprocessing ein wichtiger Faktor für das Ergebnis von NLP-Systemen ist. Es gilt außerdem zu belegen, dass die zu analysierende Sprache einen Unterschied macht und ein sprachspezifisches Preprocessing das Ergebnis beeinflusst.

> „The quality of system output is dependent on the quality of the system input.“ [Lid15, S. 112]

Fragestellungen
Forschungsfragen, die durch die Erarbeitung zu beantworten sind:

- **FF1:** Welche Forschung zu Methoden im Preprocessing gibt es bereits und welche Rolle spielen diese in Hinblick auf die Sprachabhängigkeit?
- **FF2:** Wie sieht der Ablauf des Preprocessings von deutschen Texten im Vergleich zu englischen aus?
- **FF3:** Inwiefern ist ein sprachspezifisches Preprocessing für die Genauigkeit der Ergebnisse von Sentimentanalysen ausschlaggebend?

  Hypothese: Ein sprachspezifisches Preprocessing macht einen signifikaten Unterschied in der Qualität der Ergebnisse von Sentimentanalysen.

1.4 Abgrenzung

Der Kontext ergibt sich aus dem vorangegangenen Projekt und ist damit eine interpersonelle, deutschsprachige, asynchrone Kommunikation in einer Chatumgebung. Die Textnachrichten werden zwischen zwei oder mehr Teilnehmern ausgetauscht.

Die Arbeit beschäftigt sich hauptsächlich mit dem spezifischen Teil der sprachabhängigen Preprocessing Schritte und behandelt die sprachunabhängigen nur oberflächlich, um Lücken im Konzept vorzubeugen.

Es wird sich an bestehenden Algorithmen und Techniken im Englischen orientiert, das Augenmerk liegt jedoch auf der deutschen Sprache. Es werden nach der Analyse und der Erstellung des Konzepts nur beispielhafte experimentelle Auswertungen durchgeführt, um die Hypothese zu belegen, kein kompletter Workflow umgesetzt.

1.5 Vorgehensweise

Im ersten Schritt werden die Grundlagen des Preprocessings dokumentiert und etablierte Methoden genauer untersucht und eingeordnet.

Im Anschluss werden ein oder mehrere sprachspezifische Preprocessing Schritte beispielhaft abgewandelt und an einem Datenset getestet. Das Ergebnis soll Aussagen darüber erlauben, ob die Abwandlung einen positiven Effekt auf die durchgeführte Machine Learning Aufgabe hatte. Im Detail sieht die Vorgehensweise wie folgt aus:

Datenerhebung
Zunächst werden Sekundärdaten erhoben, das heißt es wird eine geeignete Datenmenge in deutscher Sprache gewählt. Die Wahl des Datensatzes hinsichtlich seiner Form, Größe und seines Kontexts werden dargelegt und begründet.

Datenanalyse
Die unstrukturierten Rohdaten werden bereinigt und aufbereitet. Sie werden auf Vollständigkeit, ungewollte Zeichen, Sonderzeichen und weitere Störfaktoren überprüft und anschließend mithilfe von existierenden Preprocessing-Programmen analysiert. Die Wahl der Tools wird ebenfalls von vorher definierten Kriterien abhängig gemacht und dargelegt.

1. Es werden Anforderungen definiert, sodass explizite Kriterien für die Ergebnisse feststehen.

2. Auf bestehende Modelle im Preprocessing wird eingegangen und bezüglich Eignung für eine Analyse deutscher Sprache beurteilt.

3. Die Unterschiede und Gemeinsamkeiten zur Vorgehensweise im Deutschen (im Vergleich zum Englischen) sollen herausgearbeitet werden.

Beispielhaft werden die Programme abgewandelt und sowohl in ursprünglicher als auch veränderter Form auf den Datensatz angewandt. Getestet werden diese beispielhaft in einer geeigneten Programmiersprache.

1. Der Datensatz wird mithilfe von bestehender Preprocessing Software behandelt und anschließend wird damit eine Sentimentanalyse durchgeführt.

2. Der Datensatz wird mit angepasster Software behandelt. Das vorige Verfahren wird wiederholt.

Datenauswertung
Im letzten Schritt werden die Ergebnisse diskutiert und so aufbereitet, dass daraus

Schlüsse zur Hypothese gezogen werden können. Zudem wird der idealtypische Ablauf für ein entsprechendes Preprocessing dokumentiert.

2 Hintergrund

Um ein besseres Verständnis der folgenden Untersuchungen zu gewährleisten, wird in diesem Kapitel zentrale Begriffe der künstlichen Intelligenz erläutert. Es wird hauptsächlich über die Grundbegriffe des Natural Language Processings informiert und daraufhin detaillierter auf relevante Themenfelder wie verschiedene Modelle und insbesondere Preprocessing eingegangen.

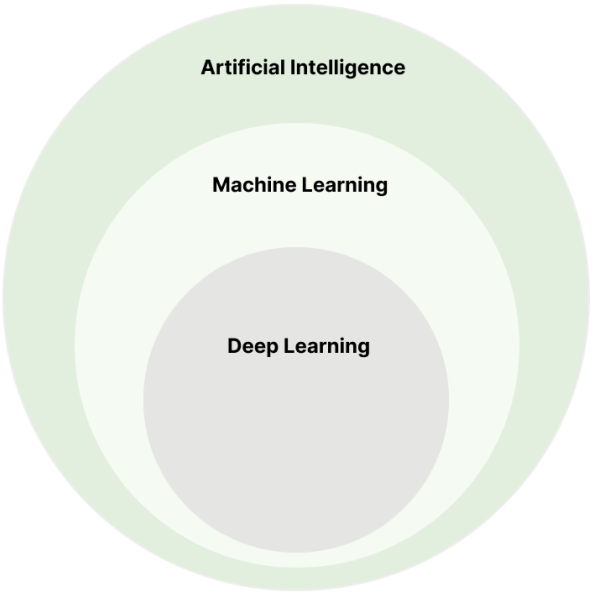

Abbildung 2.1: Kategorien der künstlichen Intelligenz nach [Wol22a]

2.1 Grundbegriffe des Natural Language Processing

Allgemein Natural Language Processing (NLP) beschäftigt sich mit dem Verarbeiten von natürlicher Sprache. Natürliche Sprache ist von Menschen und unter Menschen angewandte Sprache.
Eine der Herausforderungen der Verarbeitung natürlicher Sprache stellt die Komplexität und Mehrdeutigkeit der menschlichen Sprache dar. Da Computer nicht wie Menschen aus Erfahrung lernen können, um Sprache besser zu verstehen, müssen sie Algorithmen und Methoden der künstlichen Intelligenz und des maschinellen Lernens anwenden. Um die Textbedeutung als Ganzes zu erkennen, ist es notwendig, im Vorfeld eine Vielzahl von Daten zu erheben und die identifizierten Muster für die Analyse zu nutzen. Nebst künstlicher Intelligenz und maschinellem Lernen spielen dabei Big-Data-Technologien eine wichtige Rolle.

Als erstes wird der Unterschied zwischen Machine Learning und Deep Learning sowie jener zwischen überwachtem und unüberwachtem Lernen vorgestellt. Anschließend werden konkrete, für diese Arbeit relevante Machine Learning Modelle präsentiert, um später ihre Eignung für das Experiment dieser Arbeit beurteilen zu können. Anschließend wird der Gegenstand und Kontext der Untersuchung, die Sentimentanalyse, vorgestellt.

2.1.1 Machine Learning

Machine Learning (ML) wird häufig als Überbegriff für das Anlernen von Computer-Systemen aus Daten verwendet. Es werden Algorithmen angewandt, welche Muster erkennen und darauf basierend schlussfolgernd Aufgaben erfüllen können. Es werden also zum Beispiel Vorhersagen getroffen, ohne dass diese explizit mathematisch programmiert wurden. Ein Machine Leraning Modell kann *überwacht* (supervised) oder *unüberwacht* (unsupervised) angelernt werden. Dies wird in den folgenden Abschnitten noch genauer erläutert. [Wol22a]

2.1.2 Deep Learning

Das *Deep Learning* (DL) baut auf dem ML auf, wird allerdings eher als „Weiterentwicklung“ [Wol22a] angesehen 2.1. DL beschreibt Algorithmen, welche Daten ähnlich analysieren, wie ein menschliches Gehirn dies tun würde. Es richtet sich nicht mehr an rein statistischen Grundlagen, sondern hat viel mehr Schichten. Diese Schichten sind bilden ein künstliches neuronales Netz und sind sehr leistungsfähig 2.2. Auch hier ist

sowohl das *supervised* als auch das *unsupervised Learning* möglich. Zu den bekanntestetn Modellen zählen *LSTM*[1] sowie RNN und CNN[2].

2.1.3 Supervised Learning

Ein Modell mit überwachtem Lernen definiert sich dadurch, dass bereits ein logischer Zusammenhang zwischen den Daten bekannt ist. Die Daten werden von Experten kategorisiert und entsprechend dokumentiert. So können sie dem Algorithmus gegeben werden und bei neuem Input als Grundlage für das Ergebnis dienen. [Wol22b]

2.1.4 Unsupervised Learning

Das unüberwachte Lernen hat zunächst zum Ziel, eine gewisse Struktur oder Regelmäßigkeit zwischen den Daten zu finden 2.3. Es gibt verschiedene Methoden, wie die Maschine dies durchführt, zum Beispiel gibt es das *Lexikalische Lernen.* [Wol22b]

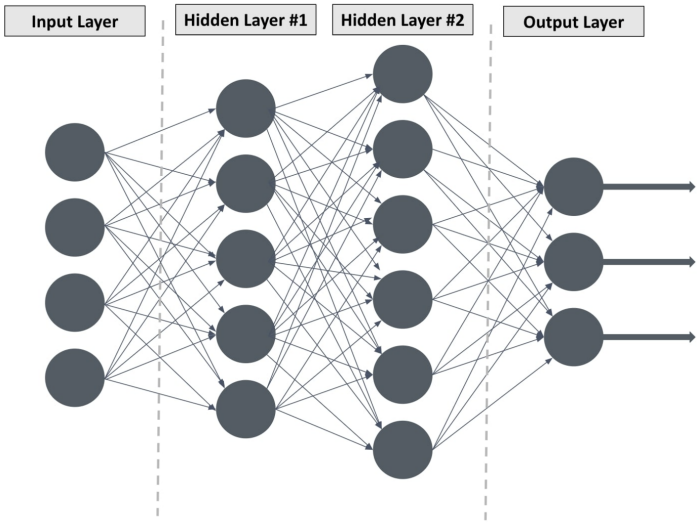

Abbildung 2.2: Ein simples künstliches neuronales Netz [Wol22a]

1 Long Short-Term Memory, ein künstliches neuronales Netz für Deep Learning
2 Recurrent und Convolutional Neural Network

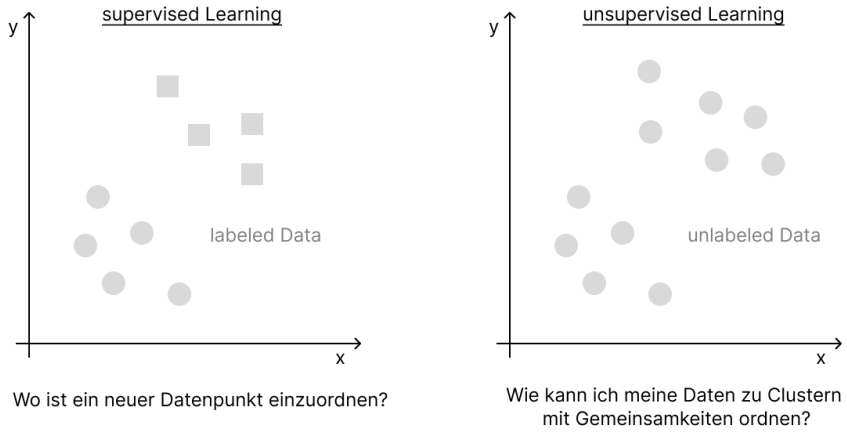

Abbildung 2.3: Das Ziel des überwachten Lernens ist es, Vorhersagen auf der Grundlage eines Trainingssatzes von gekennzeichneten Daten zu treffen. Das Ziel des unüberwachten Lernens ist es, unmarkierte Daten zu clustern. Nach [Wol22b]

2.1.5 Relevante Modelle

Im Weiteren werden aktuelle Machine Learning Modelle vorgestellt, welche in der experimentellen Untersuchung verwendet werden sollen. Es handelt sich um vier Machine Learning Modelle, welche den *supervised* Modellen zuzuordnen sind.

Logistic Regression

Die *Logistische Regression* ist eine Regressionsanalyse, welche eine Wahrscheinlichkeit zur Zugehörigkeit einer Kategorie liefert. Auf einem Graphen werden die bekannten Datenpunkte und anhand ihrer Abhängigkeiten eingeordnet und mithilfe der Sigmoid-Funktion[1] interpretiert. So kommt für eine Vorhersage eines neuen Datenobjekts ein Wert zwischen 0 und 1 heraus, welcher eine Tendenz zur Kategorienzugehörigkeit ausdrückt. [Uni22]

Naive Bayes

Dieses Modell ist eine sehr einfache Methode. Es wird eine bedingte Wahrscheinlichkeit P dafür berechnet, ob ein Objekt einer bestimmten Klasse zugehört und jene mit höchstem P gewinnt. Der Naive Bayes Algorithmus basiert auf dem Bayes Theorem und geht davon aus, dass alle Eingabevariablen und deren Ausprägung unabhängig

1 $S(x) = \frac{1}{1+e^{-x}}$

voneinander sind. Er beachtet bei Texten außerdem nicht die Reihenfolge der Wörter, was die Auswertung Kontext-unabhängig macht.

Zunächst wird der Bayes Klassifikator mit bereits klassifizierten Daten trainiert. Danach werden die Daten in ihre Klassen aufgeteilt und dann in Tokens, also einzelne Textbausteine, zerlegt. Anschließend wird das Vorkommen der einzelnen Tokens für jede Klasse gezählt und daraus eine Wahrscheinlichkeit für die Zugehörigkeit des Tokens zu einer Klasse berechnet. Hierfür wird das Bayes Theorem genutzt [Dom20].

$$P(A|B) = \frac{P(B|A)*P(A)}{P(B)}$$

Linear SVM

Die *Linear Support Vector Machine* (SVM) ist ein überwachter Lernalgorithmus, der in der Lage ist, lineare binäre Klassifizierungsprobleme zu lösen.Anhand vorhandener Daten und Labels wird entschieden, welcher Klasse ein neues Objekt zugeordnet werden kann. Auf Basis des Trainings werden Trennebenen erstellt, welche die Objektklassen einteilen 2.4. Die SVM strebt danach, den Abstand zwischen den Ebenen möglichst von der weit zu distanzieren. [Cro06]

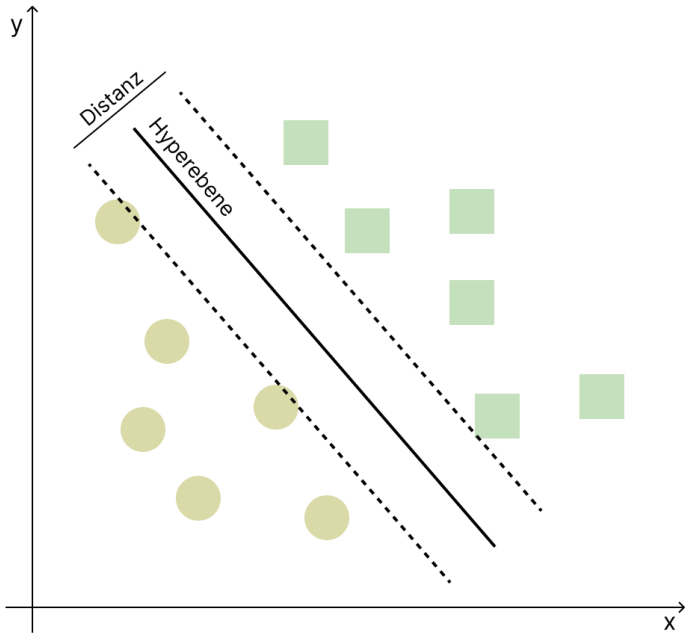

Abbildung 2.4: Hyperplane der SVM nach [Wes19, S. 12]

XGBoost

Die Grundidee von Boosting besteht darin, viele schwache Klassifikatoren zu kombinieren, um einen starken Klassifizierer zu bilden. Für jede Iteration werden Informationen aus dem vorherigen Baum verwendet und schrittweise die Fehlklassifizierungsrate verringert. Im Vergleich zu anderen Implementierungen von Gradient Boosting ist die Haupterweiterung von XGBoost die gesteigerte Effizienz für sehr große Datensätze. [Wes19]

Auch wenn der Datensatz nicht allzu groß ist, lohnt sich das Modell im Gegensatz zu dem sehr ähnlichen Algorithmus *Random Forest*, denn es wird beim Booster mehr Regularisierung verwendet, um Überanpassung zu vermeiden [Wes19]. Das Modell wurde bisher in verschiedenen Machine Learning Challenges eingesetzt und schneidet sehr gut ab. [Tia16]

2.2 Preprocessing

Preprocessing ist das Auf- und Vorbereiten der Daten, bevor eine spezifische Aufgabe darauf ausgeführt wird. Lernt man ein Machine Learning Modell für NLP-Tasks an, sollten die zu analysierenden Dokumente von Störfaktoren befreit werden.

Es gibt Kontext-spezifisches Preprocessing, bei dem untersucht wird, wie eine KI-Aufgabe verbessert (genauer) werden kann, indem man das verwendete Preprocessing an den Kontext der Domäne anpasst. Zum Beispiel sind bei der Datenvorverarbeitung von Blog Posts ganz andere Eigenschaften zu berücksichtigen als bei automatischen Nutzungsdaten einer cloudbasierten Plattform. Analog dazu gesehen kann man den Fokus zusätzlich auch auf die Art der Daten legen, zum Beispiel auf die natürliche Sprache bei textuellen Daten, wie in dem vorliegenden Anwendungsfall.

2.2.1 Ablauf

Während des Preprocessings werden verschiedene Phasen durchlaufen. Die große Datenmenge wird systematisch bereinigt und aufbereitet, geordnet und transformiert. Es wird absolut systematisch und ergebnisoffen mit statistischen Methoden gearbeitet, um noch im Vorfeld die Daten zu bewerten oder unabsichtlich zu kategorisieren. [Ily19, S. 13]

Zunächst wird davon ausgegangen, dass die Daten in einem strukturierten, tabellarischen Format vorliegen. Nach dem Preprocessing sollen aufbereitete, maschinenlesbare Daten in einer kompakten Form vorhanden sein, welche von Machine Learning Algorithmen verarbeitet werden können. Im Einzelnen handelt es sich um die folgenden Schritte:

1. **Datenbereinigung**
   Hier können bereits viele Daten wegfallen. Beim ersten Durchgehen werden sämtliche unvollständige Daten eliminiert, dazu gehören alle leeren Spalten, unvollständige Datensätze und Redundanzen. Möglicherweise können eindeutige, fehlende Werte ergänzt werden. Es wird auch als *Noise Removal* bezeichnet.

2. **Datenverständnis**
   In diesem Schritt werden zusammengehörige Tabellen vereint, um die Zusammenhänge des Sachverhalts besser verfügbar zu machen. So werden alle Informationen in einem Datensatz gesammelt.

3. **Datentransformation**
   Dies ist der Kern der Datenvorverarbeitung, hier werden alle Reduktionen durchgeführt, welche die Werte selbst betreffen. Die Daten werden durch verschiedene Techniken verdichtet und maschinenlesbar gemacht. Kategorien können als Zahlen enkodiert werden. Zeichen, Tags und Emojis aus dem Data Mining werden verarbeitet, das heißt je nach Bedeutsamkeit für die Analyse gelöscht oder ebenfalls enkodiert, und Textinhalte werden normalisiert, daher auch häufig *Normalisation* genannt.

   [Sar16]

Die einzelnen Schritte eines typischen Preprocessings im NLP werden zum besseren Verständnis einzeln erläutert. Sie lassen sich chronologisch in den zuvor beschriebenen Ablauf einordnen.

Bereinigung

Bei der Bereinigung werden in der Regel alle Zeichen, welche nicht zu dem Alphabet der Sprache der Subjekte gehören, gelöscht. Dies kann beim Anlernen des Algorithmus' allerdings zu einer schlechteren Performance führen, belegen Studien aus dem Jahr 2021 [Dal21].

Tokenization

Für das vorliegende Anwendungsszenario sind die relevanten Objekte vulgäre Wörter. Das heißt, das Programm soll an einem bestimmten Punkt Wörter einzeln verarbeiten können. Dabei stellt sich die nicht zu banale Frage, was Wörter im abstrahierten, informatischen Sinne für eine Maschine sind. Unter Tokenization versteht man das Aufspalten einer Textmenge in kleinere Bausteine. Dies kann auf Satzebene oder auf Wortebene

erfolgen. Die Aufspaltung auf Wortebene bringt uns optimale Voraussetzungen für die Vektorisierung und damit zu einem maschinenlesbaren Format für Wörter. [Sie, S. 19]

Stop Word Removal

Das sogenannte *Stop-Word-Removal* beschreibt ein Verfahren, bei welchem alle Wörter ohne polarisierende Bedeutung für den Satz entfernt werden [Kal21]. Beispiele dafür sind: *zu*, *und*, *oder* und viele mehr. Dieser Prozess kann recht effizient über Bibliotheken ausgeführt werden, wenn diese ein jenes Lexikon der entsprechenden Sprache zur Verfügung stellen.

Stemming und Lemmatization

Stemming und *Lemmatization* helfen bei der Textvorverarbeitung für NLP-Aufgaben. Beide Methoden machen mehrere Wörter in verschiedenen grammatikatlischen Formen auf eine gemeinsame abbildbar. So werden Zusammenhänge für den Algorithmus besser erkennbar [Wei20].

Beim Stemming werden Wörter auf ihren Wortstamm reduziert, das bedeutet der konjugierte Teil des Wortendes wird abgeschnitten. Um es genauer nach J. B. Lovins auszudrücken:

> „Ein Stemming-Algorithmus ist ein rechnergestütztes Verfahren, das alle Wörter mit demselben Wortstamm [...] auf eine gemeinsame Form reduziert, indem in der Regel jedes Wort von seinen Ableitungs- und Flexionssuffixen befreit wird. “ [Jul68]

Veranschaulicht kann ein Beispiel dafür so aussehen:

leiten → leit
Leiter → leit
Leiters → leit

Beim Bilden des Lemmas wird die Bedeutung mit in den Prozess einbezogen und komplett abweichende Wortformen werden mit Hilfe von lexikalischen Korpora auf das Ursprungswort zurückgeführt. [Múj08]

Leiters → Leiter
besser → gut

2.2.2 Sentimentanalyse

Die *Sentimentanalyse* (SA) ist eine von vielen Aufgaben, die man mit einem Algorithmus aus der künstlichen Intelligenz bearbeiten kann. Auch diese hat noch zahlreiche Unteraufgaben. Je nach Anwendungsfall soll nach der Analyse ein bestimmtes konkretes Ergebnis vorliegen. Eine Sammlung an möglichen konkreten Aufgaben ist in Abbildung 2.5 zu sehen. Dies ist keineswegs eine vollständige Übersicht, deckt aber die essentiellen Felder ab. [Sie]

Es gibt drei Herangehensweisen, der Komplexität von Intelligenz nach geordnet: Die regelbasierte SA, SA mit Machine Learning, SA mit Deep Learning. Erstere ist ein Abgleich mit einem Wörterbuch des einzuordnenden Sentiments, welches händisch erstellt wird. Letztere Methoden basieren auf Algorithmen aus dem Gebiet der künstlichen Intelligenz, wie in den vorigen Kapiteln beschrieben. [Kli]

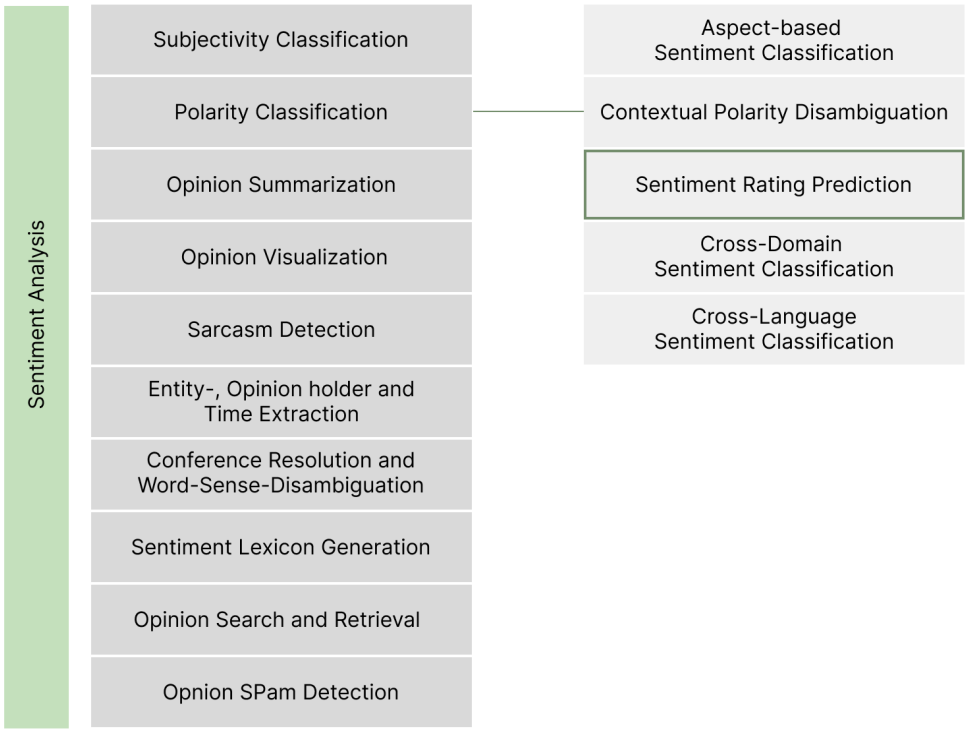

Abbildung 2.5: KI-Aufgaben nach [Sie, S. 11]

3 Verwandte Forschung

Hier wird zunächst vorgestellt, wer an dem Problem bereits gearbeitet hat und welche Lösungswege und Ressourcen dabei verwendet werden.

Es wird zum einen gezeigt, wo das Preprocessing und dessen Einfluss bereits erforscht wurde und zum anderen wird auf die Rolle von verschiedenen natürlichen Sprachen im Natural Language Processing eingegangen. Dabei soll vor allem vermittelt werden, zu welchen Ergebnissen diese Arbeiten gekommen sind und inwiefern dort noch Forschungslücken vorliegen, zu denen die vorliegende Arbeit ergänzend Antworten bringen kann. Eine Evaluation von verschiedenen Preprocessing-Techniken auf einem deutschen Datensatz wurde noch nicht im wissenschaftlichen Rahmen vollzogen. Daher werden hier verwandte Arbeiten zum Einfluss des Preprocessing und zu Daten in anderen Sprachen und dessen Ergebnisse vorgestellt.

Der Stand der Forschung und Technik kann in mehrere Abschnitte unterteilt werden, welche die Thematik von unterschiedlichen Sichten aus beleuchten. Die erste Forschungsfrage kann damit schon ansatzweise bearbeitet werden: „Welche Forschung zu Methoden im Preprocessing gibt es bereits und welche Rolle spielen diese in Hinblick auf die Sprachabhängigkeit?

3.1 Der Einfluss des Preprocessings

Besonders für Texte von Microblogging Portalen wie Twitter gibt es eine Reihe an Forschungsarbeiten zu Sentimentanalysen auf diesem Gebiet. Die Rolle von Preprocessing Techniken stellt sich dabei von besonderer Bedeutung heraus. Es wird – hauptsächlich zu englischen Texten, aber mittlerweile auch für andere Sprachen – untersucht welche Unterschiedlichen Techniken es gibt und welchen Einfluss auf das Endergebnis der Klassifizierung sie aufzeigen.

3.2 Preprocessing in anderen Sprachen

Der Einfluss von Preprocessing (PP) auf Sentimentanalysen (SA) der türkischen Sprache wird in einem Paper im Rahmen der *26. IEEE Signal Processing and Communications Applications Conference* untersucht. Das untersuchte Subjekt der Analyse sind türkische Film- und Produktrezensionen. Mehrere Rohdatensätze wurden verschiedenen Kombinationen von Preprocessing-Schritten unterzogen. Dabei handelt es sich um folgende: initiales Preprocessing, Stemming, Stopword, Common emotions, Negation. Bei dem initialen PP werden alle Inhalte, welche nicht relevant für die Polarität der Daten sind, entfernt. Dies umfasst das Entfernen von URLs, Benutzernamen, zeitliche Angaben wie Daten, Ziffern und Hashtag-Symbolen. Dieser Schritt wird immer augeführt, da er notwendig und unabhängig von der Sprache ist. Die anderen Methoden werden auf ihren Einfluss auf die Genauigkeit des Endergebnisses hin untersucht. Zum Klassifizieren wird ein hybrides Modell namens *Tw-StAR* verwendet, welches eine Kombination aus einem überwachten Machine Learning Modell mit SVM und NB sowie einem Lexikon-basierten Modell darstellt. [Mul18]

Die These, dass das Vorverarbeiten von Daten die Ergebnisse im Allgemeinen erheblich verbessert, konnte bestätigt werden. Der größte Einfluss konnte beim Stemming in einem Lexikon basierten Modell beobachtet werden. Bei dem Einbezug von Emoji-Tagging konnte sogar ein Verlust der Vorhersagegenauigkeit in der Sentiment-Analyse verzeichnet werden. Dies wird damit erklärt, dass in den verwendeten Datensets eine zu geringe Quantität dieser vorhanden war und den Algorithmus verwirrte. [Mul18]

In einer umfangreichen Sentimentanalyse im englischsprachigen Gebiet werden über 14 einzelne Preprocessing Techniken evaluiert. Im Ergebnis steht fest, dass Stemming, das Markieren von wiederholter Interpunktion und das Entfernen von Zahlen sich eindeutig positiv auf das Ergebnis auswirken. Abgeraten wird davon, Interpunktion komplett zu entfernen, Slang zu Ersetzen und Rechtschreibung zu korrigieren. Es wird allerdings auch darauf aufmerksam gemacht, dass das Ergebnis nur für die isoliert angewandten Techniken gilt und bei anderen Modellen weiter zu erforschen ist. [Ala19]

In einem anderen Bericht zur Analyse deutscher Tweets werden mit unterschiedlichen Trainings- und Test-Corpora zwei Classifier getestet: SVM und CNN. Der F1-Scores ist der Makro-Mittelwert der positiven und negativen F1-Scores. Resultat der Forschung war, dass das neuronale Netz besser abschneidet als die Feature-basierte Support-Vector-Machine. Die Unterschiede in der Genauigkeit der Klassifizierung sind allerdings im Vergleich zur fundamental längeren Laufzeit und Komplexität sehr gering. Jedoch ist bei beiden noch mehr Verbesserungspotenzial zu erwarten. [Cie17]

4 Konzept

Beginnend mit den Anforderungen an die Tools und Daten wird nun das Konzept erläutert. Es werden alle verwendeten Tools im Einzelnen dargestellt und der Gegenstand der Untersuchung präsentiert. Anschließend wird der Ablauf der Datenanalyse Schritt für Schritt dargelegt. Dabei handelt es sich im Einzelnen um Datenbereinigung, -verständnis und -exploration. Zum Schluss dieses Kapitels wird auf die Durchführung der Sentimentanalyse eingegangen und anhand welcher Größen die Ergebnisse bewertet werden.

4.1 Datenerhebung und Wahl der Tools

Zunächst werden die Anforderungen an einen geeigneten Datensatz für dieses Experiment anhand von Sekundärliteratur und Logik gesammelt. Verfügbare Datensätze werden nach diesen Kriterien untersucht und auf ihre Eignung für das Experiment beurteilt. Es werden ein oder mehrere geeignete Datensätze gewählt und für eine bessere Nachvollziehbarkeit ihre Inhalte visuell dargestellt. Zum Schluss wird noch erläutert, mit welchen Tools und welcher Software gearbeitet wird.

4.1.1 Anforderungen an den Datensatz

Bei der Wahl des Datensatzes sind folgende, der Priorität nach geordnete Kriterien entscheidend.

Grundvoraussetzung ist, dass die Sprache der zu analysierenden Inhalte Deutsch ist. Dies schließt nicht aus, dass einzelne englisch- oder anderssprachige Wörter enthalten sind, da diese mittlerweile tief in der Sprache verankert sind [Sch20].

Um eine Sentimentanalyse durchzuführen muss der Datensatz in einen Test- und Trainingsteil getrennt werden. Um die künstliche Intelligenz „anzulernen“, müssen Inhalte vorliegen, die binär oder anders kategorisiert eindeutig als obszön gekennzeichnet sind. Da Hass und Beleidigungen ein sehr subjektiv wahrgenommenes Thema sind, soll diese Einordnung von Experten durchgeführt worden sein. [Iri21]

Der Kontext ist zudem Grund für die bevorzugte Wahl von kurzen Textinhalten wie sie in einem schnellen schriftlichen digitalen Austausch vorliegen könnten. Favorisiert werden also kurze Texte von bis zu $\sim$ 300 Zeichen[1], welche überwiegend in Alltagssprache formuliert sind.

Die Aktualität des Datensatzes spielt insofern eine Rolle, dass das zu entwickelnde System im Chatkontext ständig neuen Input bekommt und daher auf die aktuellen Spracheigenheiten angepasst sein sollte. Daher gilt: Je neuer der Datensatz desto besser.

Eine hohe Quantität der Daten präzisiert das Endergebnis von Sentimentanalysen fundamental [Mee21]. Daher ist eine Größe von Daten im 5-stelligen Bereich erstrebenswert.

4.1.2 Wahl und Exploration der Daten

RP Datensatz

Dieser Datensatz beinhaltet deutschsprachige Kommentare der Rheinischen Post (RP). Die Erhebung dieser Daten erfolgte auf wissenschaftlichen Grundlagen und ist in [Ass] ausführlich dokumentiert. Das Ziel der Studie war, mehr Daten für Big-Data Projekte bereitzustellen, welche nicht aus Twitter stammen, da dieses Feld bereits ausgeschöpft ist [Mag16].

Die Inhalte stammen aus dem Zeitraum zwischen 2018 und 2020. Jeder kommentar wird vor der Veröffentlichung von einem Community Manager vormoderiert. Das „RP-Mod“ Datenblatt enthält dabei die Entscheidung dieser Moderatoren, ob ein Kommentar als beleidigend eingestuft wird. Es gibt drei Spalten: eine eindeutige Identifikationsnummer, den Textinhalt und die binäre Klassifizierung als Hasskommentar (siehe 4.1).

Multilingualer Datensatz

Um den zuvor erläuterten Anforderungen gerecht zu werden, wird ein umfangreicher, multilingualer Datensatz zu Hatespeech gewählt. Dieser beinhaltet Datensätze zu 13 Sprachen, welche jeweils in einen gesammelten Trainings- und einzelne Testdatensätze aufgeteilt sind. Der Umfang beläuft sich dabei auf $\sim$ 4000 bis 50 000 Zeilen pro Dokument. Außerdem liegen LASER 1024-Dimensional Embeddings für Training und Test vor. Damit sind insgesamt 40 Dokumente im Dateiformat .csv vorhanden.

1 `https://twitter.com/home?lang=de`

In [171]:
```
train.sample(10)
```

Out[171]:

	id	text	label
4614	2171179	Sehr glaubwürdig. Man kann es nicht mehr hören...	0.0
2529	2221467	Sexistische und speziesistische Kackscheiße in...	1.0
3006	2410451	Ach was! Der Hitler 2.0 will in die Staatskanz...	1.0
2185	2179761	Für mich ist diese Frau "irre", anders kann ma...	1.0
1134	2040245	Wann wird Deutschland endlich die ersten DITIB...	1.0
3209	1967791	Zuerst war es die Trennung von Raucher und Nic...	0.0
1503	2093257	Bravo ! Endlich jemand der sich den Öko-Terror...	1.0
4403	1935153	Verdi hat ja auch bald nicht anderes mehr zu t...	0.0
2928	2356029	Welchen Migrationshintergrund hat denn der Tät...	1.0
5518	2227011	Die Überschrift ist falsch. Sie muss lauten: K...	0.0

Abbildung 4.1: Stichprobe des RP-Datensatzes

Es gibt vier Spalten, welche eine eindeutigen Identifikationsnummer, den Inhalt, eine binäre Identifizierung von Hasskommentaren und die als Zahl codierte Sprache enthalten 4.2.

Beim verzeichneten Autor des Datensatzes auf Kaggle handelt es sich um einen Bachelor Studenten aus Balochistan, Pakistan [Bal22]. Dadurch kann nicht in Erfahrung gebracht werden, wo genau diese Daten herkommen, auch die Kategorisierung ist nicht nachvollziehbar. Stichproben zeigen allerdings eine gute Qualität und lassen den Datensatz

In [119]:
```
train.sample(10)
```

Out[119]:

	Unnamed: 0	text	label	language
99170	99170	Ich wünsche mir, dass die Koalition platzt und...	0.0	5
82609	82609	Erst jetzt? The earlier the better!	1.0	5
79868	79868	„Frecher Juden-Funktionär“ ist lt. OLG Hamm ei...	1.0	5
75409	75409	Hier ist dringend eine Gesetzesanpassung nötig...	1.0	5
93386	93386	Tja, war ja nicht anders zu erwarten. In der S...	1.0	5
83356	83356	In Berlin geht bei den Regierenden nur die Ang...	0.0	5
88768	88768	Über 50 Milliarden pro Jahr ---- kostet uns di...	1.0	5
84637	84637	Aha, jetzt weiss man auch warum "unsere allers...	0.0	5
81488	81488	Während der Bahnfahrt und auch in öffentlichen...	0.0	5
84090	84090	Holländer können doch gar kein Auto fahren!?? ...	1.0	5

Abbildung 4.2: Stichprobe des deutschen Teils des multilingualen Datensatzes

vertrauensvoll wirken. Für diese Arbeit wird angenommen, dass die Daten in ihrem Ausgangszustand seriös und zuverlässig verwertbar sind.

4.1.3 Wahl der Tools

Die gewählten Tool sollten zu Entwicklungszeiten möglichst effizient für die Laufzeit sein, daher werden externe Cloud-Services bevorzugt.

Kaggle

Kaggle ist ein Datenportal, das von der Community erstellte und externe Datensätze anbietet und eine Jupyter Notebook Integration hat (siehe nächster Abschnitt). Ein großer Vorteil ist die direkte Anbindung an die Datenquelle und der Export als Python-Dokument.

Es wird als Orientierung ein Kaggle-Notebook mit einer Sentimentanalyse verwendet, in welchem englische Tweets klassifiziert werden. Die im weiteren gewählten Modelle und das sprachspezifische Preprocessing werden implementiert. Die vorhandene Datenbereinigung wird im Vergleich der Ergebnisse als *Standard Cleaning* verwendet. [Kan20]

Python

Als Programmiersprache eignet sich Python, da Jupyter dies unterstützt und es viele integrierte Methoden zur Analyse v Text hat. Sehr viele Bibliotheken sind verfügbar, darunter welche, die open source sind.

Jupyter vs. Google CoLab

Jupyter[1] stellt Rechenleistung für Datenauswertungen zur Verfügung, unterstützt die gängigen Programmiersprache für Datenauswertungen (Python, R und Julia) Es gibt zudem vscode Unterstützung für jupyter Notebooks im Dateiformat .ipynb.

Google CoLab[2] ist eine spezialisierte Version von Jupyter Notebooks mit Anbindung an den Cloud Service von Google. Dies vereinfacht kollaboratives Arbeiten, ist aber auch eine Schwachstelle, da die Privatsphäre und Sicherheit bei jenem Service umstritten ist.

1 `https://jupyter.org/`
2 `https://colab.research.google.com/`

Anhand dieser Informationen wird die in Kaggle integrierte Jupyter-Umgebung gewählt.

forText

Als Standard Preprocessing (PP1) wird ein bestehendes einfaches Preprocessing Programm für deutsche Sprache aus dem Online-Portal *forText*[1] gewählt [Sch22]. Es soll einen Vergleich zu den in dieser Arbeit konzipierten Preprocessing-Pipeline darstellen.

4.1.4 Wahl des Modells

Um dem aktuellen Stand der Forschung möglichst gerecht zu werden, werden die etabliertesten Machine Learning Modelle gewählt, welche in mehreren Textanalyse-Aufgaben zuverlässig abschneiden. Dabei handelt es sich um die bereits vorgestellten Modelle: *Logistic Regression, Naive Bayes, Linear SVM and XGBoost.* Auch in verwandten Forschungsarbeiten werden diese Modelle häufig als Grundlage genutzt. [Wes19] [Che16]

4.2 Datenanalyse

Nun wird aufgezeigt, wie Preprocessing auf den gewählten Datensatz angewandt werden kann. Es werden zahlreiche Bereinigungsschritte vorgestellt und beschrieben auf welche Art und Weise diese durchgeführt werden können.

Oft bedienen sich die genannten Programme einer generischen Preprocessing Bibliothek, wie beispielsweise *Spacy*, welche viele Ressourcen zum Vorverarbeiten von textuellen Inhalten zur Verfügung stellt. Für die englische Sprache solche Bibliotheken aufzufinden ist problemlos möglich, es gibt quantitativ verhältnismäßig viel Auswahl. Für die deutsche Sprache stehen solche nur begrenzt zur Verfügung. Die vorhandenen inkludieren üblicherweise eine generische Bereinigung und manchmal Stopword-Removal.

Darüber hinaus gehen die bestehenden Programme meist nicht, da es aufwändiger ist, verschiedenste, auf die Sprache zugeschnittene Bibliotheken aufzufinden. Im Folgenden wird genau dies gemacht und eine umfangreiche Preprocessing-Pipeline zusammengestellt, welche im Anschluss experimentell getestet wird.

1 `https://fortext.net/routinen/lerneinheiten/preprocessing-mit-nltk`

4.2.1 Datenbereinigung

Die Daten sollen nur jene Informationen enthalten, welche von semantischem Wert sind. Alles, was dies übersteigt wird eliminiert oder ersetzt. Dies können auf kleinster Ebene Leerzeichen, vielfache Buchstaben oder Sonderzeichen sein. Auf Wortebene können es zum Beispiel Tags oder Usernamen sein, welche inhaltlich nicht zum Sentiment des Satzes beitragen. Die Grenze der Bereinigung ist allerdings in manchen Fällen nicht eindeutig vom Einfluss aufs Sentiment losgelöst. Jenes wurde bereits in einigen Untersuchungen getestet [Sar18]. Es stellt sich heraus, dass sich dies je nach Modell, Sprache und Datennatur unterschiedlich auswirken kann.

Leere Daten

Unvollständige Daten in die Analyse mit einzubeziehen verfälscht das Ergebnis, weil diese bei dem Anwendungsfall von Sentimentanalysen keine Aussagekraft und damit keine Relevanz hat. Es wird somit geprüft, welche Daten keinen Inhalt oder kein Label erhalten haben. Liegen solche vor, werden sie erkannt und entfernt.

HTML-Tags entfernen

Dies ist ein sprachunabhängiger Schritt und rein technischer Natur. Durch das automatisierte Sammeln der Daten über Bots (oder API-Schnittstellen) können Tags für Zeilenumbrüche oder Textstile in den Textobjekten enthalten sein. Um diese zu entfernen kann die Bibliothek *BeautifulSoup*[1] verwendet werden (siehe 4.3).

Extra Leerzeichen entfernen

Bei Kurznachrichten in Chatumgebungen können doppelte oder sogar mehr Leerzeichen auftreten. Diese haben keinen Einfluss auf die Aussage des Satzes und dienen lediglich

```python
from bs4 import BeautifulSoup

def remove_html(text):
    soup = BeautifulSoup(text, "html.parser") #remove html tags
    text = soup.get_text()
    return text
```

Abbildung 4.3: HTML-Tags entfernen

1 https://www.crummy.com/software/BeautifulSoup/bs4/doc/

der Strukturierung und Lesbarkeit. Um in einem späteren Schritt die einzelnen Wörter als Tokens zu extrahieren, sollten die Trennungszeichen zwischen Worten einheitlich lang sein.

Umlaute und Akzente normalisieren

Umlaute werden in ihre normalisierte Form umgewandelt, da sie ein Sonderkonstrukt der deutschen Sprache darstellen (siehe 4.4).

Akzente treten in der deutschen Sprache nicht besonders häufig auf, jedoch kommen sie vor, vor allem, wenn Wörter aus dem Französischen oder Spanischen verwendet werden. Daher sollte man diese ebenfalls in Betracht ziehen. Akzente werden durch die Dekodierung in Unicode entfernt (siehe 4.5). Sie machen Wörter wie „café“ und „cafe“ vergleichbar. Mit Akzent würden sie vom Algorithmus als unterschiedliche Dinge wahrgenommen werden, obwohl sie das gleiche referenzieren. [Val19, S. 46-47]

```python
def convert_umlauts(text):
    """
    Replace umlauts for a given text

    :param word: text as string
    :return: manipulated text as str
    """

    tempVar = word # local variable

    # Using str.replace()
    tempVar = tempVar.replace('ä', 'ae')
    tempVar = tempVar.replace('ö', 'oe')
    tempVar = tempVar.replace('ü', 'ue')
    tempVar = tempVar.replace('Ä', 'Ae')
    tempVar = tempVar.replace('Ö', 'Oe')
    tempVar = tempVar.replace('Ü', 'Ue')
    tempVar = tempVar.replace('ß', 'ss')

    return tempVar
```

Abbildung 4.4: Umlaute konvertieren

```
def remove_accented_chars(text):
# remove accented characters from text, e.g. café
text = unidecode.unidecode(text)
return text # cafe
```

Abbildung 4.5: Akzente entfernen

Emojis

Beim Umgang mit Emojis spalten sich die Meinungen. Nach einer Untersuchung an der Stanford Universität [Ale09] seien Emojis nicht bereichernd für die Klassifizierung von Texten. Grund dafür sei beispielsweise, dass man nicht einschätzen könne, wann ein positiver Emoji als Sarkasmus verwendet wird und wann nicht. Seitdem haben andere Untersuchungen allerdings ergeben, dass es sich für ein besseres Ergebnis rentiert, die Emojis mit einzubeziehen [Ala19]. In der durchzuführenden Analyse werden beide Varianten getestet. Einmal werden sämtliche Emojis entfernt 4.6 und einmal werden eine Auswahl an Emojis als Text kodiert, zum Beispiel „:-)“ wird „lächeln“ in den Inhalt aufgenommen.

Zahlen

Zahlen haben keine ausschlaggebende Wirkung auf die Aussage des Subjektes (des Satzes). Häufig werden sie daher entfernt. Ein Fall in welchem sie nicht entfernt werden sollten ist, wenn die Textanalyse-Aufgabe darin besteht, numerische Identifikationsnummern von erwähnten Tickets zu extrahieren.

Ausgeschriebene Zahlen werden als sinnvolle Tokens erkannt, solange sie in Textform vorliegen. Um auch diese zu erkennen und zu entfernen, gibt es im Englischen das sogenannte word2number-Modul, welches ausgeschriebene Zahlen und Zahlwörter konvertiert und sie anschließend automatisch gelöscht werden können.

```
import re,string,unicodedata

def remove_emojis(text):
    # Code to remove emojis
    emoji_clean= re.compile("["
                            u"\U0001F600-\U0001F64F"  # emoticons
                            u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                            u"\U0001F680-\U0001F6FF"  # transport & map symbols
                            u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                            u"\U00002702-\U000027B0"
                            u"\U000024C2-\U0001F251"
                            "]+", flags=re.UNICODE)
    text = emoji_clean.sub(r'',text)
    return text
```

Abbildung 4.6: Emojis entfernen

Sonderzeichen entfernen

Sämtliche Interpunktion wird entfernt, aber Satzendzeichen – das heißt der Punkt, das Ausrufezeichen und das Fragezeichen – bleiben erhalten, da sie vor allem in Kurznachrichten das Sentiment verstärken können.

Kleinbuchstaben

Alle tokenisierten Sätze werden zu Kleinbuchstaben gemacht. Dieser Schritt dient ebenfalls der Vergleichbarkeit der Wörter und ist nicht direkt mit der Sprache verbunden.

Im Englischen gibt es mit wenigen Ausnahmen für Nationalitäten und Eigennamen keinen bedeutenden Unterschied in der Bedeutung des Satzes durch Groß- und Kleinschreibung der Worte. Bei Deutsch dagegen kann die Großschreibung die Bedeutung des gesamten Satzes fundamental verändern. Ein Beispiel dafür wäre:

Der gefangene Floh. $\neq$ Der Gefangene floh.

Dafür gibt es das sogenannte *Part-of-Speech-Tagging*, welches mit in die Auswertung einbezieht, welche Rolle das Wort in einem Satz spielt. Dazu wird je Wort die grammatikalische Position im Satz analysiert und deklariert [War]. Dies hilft bei der semantischen Analyse später.

4.2.2 Datenverständnis

Da die Daten in einem einzelnen Dokument vorliegen, müssen diese nicht zusammengefügt werden; es gibt keine Beziehungen zwischen mehreren Tabellen. Ein nötiger Schritt ist allerdings die Extraktion der relevanten Sprache Deutsch, da die Modelle einen unilingualen Ansatz verfolgen. Deutsch ist mit der Zahl 5 codiert, sodass alle Daten mit diesem Sprachcode wie folgt extrahiert werden können.

```
1     data = data[lambda row: row['language'] == 5]
```

4.2.3 Datentransformation

Lemmatization

Mit der Bibliothek *Simplemma*[1] ist es analog zu *WordNetLemmatizer*[2] möglich, einzelne deutsche Wörter zu ihrer Stammform umzuwandeln. Simplemma bietet zahlreiche weitere Sprachen an, was das Tool auch für die Untersuchung anderer Sprachen und Modelle attraktiv macht.

```
1     word = simplemma.lemmatize(word, lang='de')
```

Aus den Wörtern [Hier, sind, Menschen] würde [hier, sein, Mensch] werden. Damit liegen die Tokens nach diesem Schritt in ihrer lemmatisierten Form vor.

Stopwords

Ebenfalls wenig Einfluss auf das Sentiment haben allgemeine Füllwörter wie „ich“, „es“, „damit“, usw. Diese werden mithilfe von Stopword-Bibliotheken entfernt. Auch hier ist der tokenisierte Zustand der Textobjekte wichtig, damit einzelne Wörter ausgelesen werden können. Spacy ist einer der größten Verzeichnisse solcher Wörter im Englischen. Für die deutsche Sprache gibt es ein Modul von spacy, welches einen Korpus mit deutschen *Stopwords*[3] enthält. Dabei handelt es sich um eine Liste, welche als Array eingebunden werden kann (siehe 4.7)

Es ist unter Umständen sinnvoll Negationen erkennen zu können, da diese das Sentiment sehr stark beeinflussen. Es sei zu prüfen, ob die Worte „nicht“ und „kein“ in der Stopword-Liste enthalten sind. In der vorliegenden Liste sind sie vorhanden und werden deselektiert, um sie in der Auswertung zu inkludieren.

```
# GERMAN STOP WORDS
from nltk.corpus import stopwords
german_stop_words = stopwords.words('german')
```

Abbildung 4.7: Deutsche Stopwords von nltk

1 https://pypi.org/project/simplemma/
2 https://www.nltk.org/_modules/nltk/stem/wordnet.html
3 https://spacy.io/models/de

4.2.4 sprachspezifisches Preprocessing

Bei der Recherche zu Bibliotheken und bekannten Preprocessing Techniken stellt sich heraus, welche Schritte von der zu analysierenden Sprache abhängig sind. Dieser Fall besteht stets, wenn die Datentransformation von konkreten Wörtern abhängig ist oder Spracheigenheiten normalisiert werden.

Letztlich werden jene Schritte hier mit ihrem Sprachbezug übersichtlich dargestellt, um damit auch die zweite Forschungsfrage zu beantworten. In der Auswertung werden die abgewandelten Schritte als „angepasstes Preprocessing“ (PP2) bezeichnet.

#	**Bezeichnung**	**Sprachbezug**
1	Umlaute normalisieren	Spezielle Zeichen: ä, ö, ü, ß
2	Emojis	Wörter bei Enkodierung anpassen
3	Zahlen	Ausgeschriebene Zahlen erkennen
4	Stopwords	Von Korpus in entsprechender Sprache abhängig
5	Lemmatization	Von Grammatik abhängig
6	Stemmer	s.o.

Tabelle 4.1: Preprocessing mit Sprachbezug

4.3 Sentimentanalyse

Das konkrete Ziel bei dieser Sentimentanalyse ist die *Sentiment Prediction* 2.5. Dabei wird in der Evaluation durch einen Testdatensatz geprüft, ob eine richtige Klassifizierung durch den Algorithmus vorgenommen wird.

Zunächst erfolgt der Initalisierungsschritt 4.8. Die Daten wurden bisher gesammelt und durch ein Preprocessing verarbetet. Nachdem die Daten verschiedenste Transformationen durchlaufen haben und in maschinenverwertbarem, numerischen Format vorliegen, können die gewählten Machine Learning Algorithmen damit trainiert werden. Sie werden auf unterschiedliche Arten mit zwei Datensets trainiert und getestet. Die Ergebnisse sollen Auskunft über den Einfluss des PP geben.

Zur Auswertung und Bewertung der Ergebnisse wird der F1-Score verwendet. Er setzt sich aus dem F1-Score der positiven und negativen Klassifizierung zusammen.

$$F1_{pos} = \frac{2 * precision_{pos} * recall_{pos}}{precision_{pos} + recall_{pos}} \qquad F1_{neg} = \frac{2 * precision_{neg} * recall_{neg}}{precision_{neg} + recall_{neg}}$$

$$F1 = \frac{(F1_{pos} + F1_{neg})}{2}$$

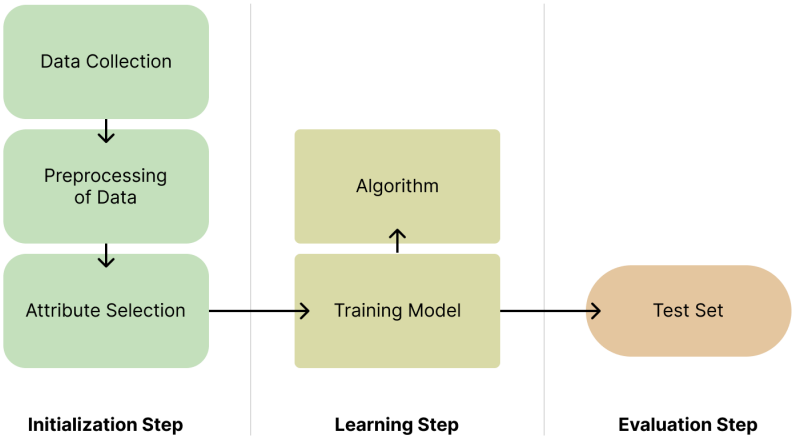

Abbildung 4.8: Erforderliche Schritte beim Training eines Klassifikators für Sentimentanalysen nach [Mag16]

Für eine genauere Darstellung und Transparenz der genauen Berechnung wird hier zusätzlich dargestellt, wie der Score im Einzelnen berechnet wird: [Sou22] [Bra]:

$precision_{pos}$ (Positive Klassifizierung) $= \frac{TP}{TP+FP}$

$precision_{neg}$ (Positive Klassifizierung) $= \frac{TN}{TN+FN}$

$recall_{pos}$ (Positive Klassifizierung) $= \frac{TP}{TP+FN}$

$recall_{neg}$ (Negative Klassifizierung) $= \frac{TN}{TN+FP}$

Dieser Score ist ein etablierter Maßstab für Qualität, weil er beide sowohl den Faktor *Precision* als auch den Faktor *Recall* miteinbezieht. Beide sind relevant für eine gute Vorhersagequalität bei einer *Sentiment-Prediction*. [Cie17]

5 Realisierung

Mithilfe der gegebenen Tools und Daten werden nun die Preprocessing Schritte im Kontext einer Sentimentanalyse experimentell angewandt. Dazu wird zunächst der Datensatz explorativ untersucht und anschließend werden alle geplanten Preprocessing-Schritte durchlaufen. Die einzelnen Schritte werden so implementiert, dass sie (de-)aktivierbar sind. So kann es verschiedenste Durchläufe in unterschiedlichen Kombinationen der einzelnen Schritte geben. Dadurch kann anhand der Ergebnisse festgestellt werden, welche Schritte die besten Klassifizierungen erzeugen, um am Ende eine für diesen Fall optimale Preprocessing-Pipeline vorstellen zu können.

Die Datenauswertung (dritter und letzter Schritt der Vorgehensweise) und Zwischenergebnisse werden zur Veranschaulichung dargestellt. Es werden der Datensatz „RP-Crowd-3“ und der deutsche Teil des Multilingualen Datensatzes, sowie der deutsche Testdatensatz verwendet.

5.1 Durchführung des Experiments

Wie im Konzept beschrieben, wird nun mithilfe der gewählten Daten und Tools eine Sentimentanalyse durchgeführt. Dazu werden mehrere Machine Learning Modelle trainiert und im Anschluss darauf basierend eine Sentiment Analyse durchgeführt.

Ein Python Programm wird mithilfe von Referenzen zu ähnlichen Sentimentanalysen mit den Modellen *SVM*, *Logistic Regression*, *Naive Bayes* und *XGBoost* aufgesetzt.

Wird die Information zum Datensatz abgerufen, liegen eindeutige Ergebnisse zur Vollständigkeit und Lückenlosigkeit vor.

```
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      6302 non-null   int64
 1   text    6302 non-null   object
 2   label   6302 non-null   float64
dtypes: float64(1), int64(1), object(1)
memory usage: 147.8+ KB
```

Abbildung 5.1: Statistik RP-Datensatz

```
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  27315 non-null  int64
 1   text        27315 non-null  object
 2   label       27315 non-null  float64
 3   language    27315 non-null  int64
dtypes: float64(1), int64(2), object(1)
memory usage: 1.0+ MB
```

Abbildung 5.2: Statistik multilingualer Datensatz

Da in alles Spalten gleich viele Werte vorliegen, welche *non-null* sind, also Inhalt haben, gibt es keine leeren Datenfelder. Daher kann der Schritt der Entfernung unvollständiger Daten übersprungen werden. Auch Duplikate liegen nicht vor.

5.2 Ergebnisse

Hier wird zusammengefasst, welche Preprocessing Schritte einen positiven Einfluss auf den schlussendlichen F1-Score haben, welcher die Qualität der Klassifizierung bewertet. Diese Schritte in Kombination erzielen die besten Ergebnisse bei deutscher Sprache in dem durchgeführten Experiment.

Zum Ablesen und Verständnis der Daten in den folgenden Tabellen wird eine Legende bereitgestellt.

F1	F1-Score siehe 4.3
SC	Standard Cleaning (inkludiert lediglich das Konvertieren in Kleinbuchstaben und eine ASCII-Normalisierung)
PP1	Standard Preprocessing [Sch22]
PP2	Sprachangepasstes Preprocessing
SWR	Stop Word Removal
unterstrichen	bestes Ergebnis innerhalb einer Kategorie
fett	bestes Ergebnis der Zeile (für ein Modell)

5.2.1 Angepasstes Preprocessing

In der Tabelle 5.1 kann man leicht erkennen, dass das angepasste Preprocessing (PP2) meist die besten Ergebnisse für den ersten Datensatz erzielt. Die Steigerung ist dabei recht gering, allerdings kann diese im NLP bereits zu verbesserten Ergebnissen führen.

Aufällig verhält sich das Modell *Naive Bayes*, welches ganz ohne jegliche Datenbereinigung oder Preprocessing am besten performt und sogar den besten Wert der gesamten

Modell	**F1 ohne PP**	**F1 mit SC**	**F1 mit PP1**	**F1 mit PP2**
SVM	0.722622	0.723065	0.700159	**0.723236**
XGBoost	0.649485	0.643404	0.630662	**0.646853**
Logistic Regression	0.71927	0.725615	0.714512	**0.731166**
Naive Bayes	**0.747819**	0.740682	0.730373	0.73751

Tabelle 5.1: Ergebnisse für den RP-Datensatz

Modell	F1 ohne PP	F1 mit SC	F1 mit PP1	F1 mit PP2
SVM	0.603275	0.607665	0.608168	**0.607876**
XGBoost	0.54639	**0.604571**	0.598163	0.545308
Logistic Regression	0.626762	**0.628592**	0.618159	0.625847
Naive Bayes	0.64287	**0.648728**	0.633352	0.639575

Tabelle 5.2: Ergebnisse für den multilingualen Datensatz

Tabelle darstellt. Dies kann auch anhand des zweiten Datensatzes in der Tabelle 5.2 beobachtet werden. Schaut man sich die aufgeschlüsselten Konfusionsmatrizen[1] an, wird ersichtlich, dass beim Naive Bayes die geringste Anzahl an *False Negatives* (FN) aufweist 5.3 und auch verhältnismäßig weniger *False Positives* (FP). Ein FN hat in der Realität mehr Bedeutsamkeit als ein FP. Beispielsweise wird eine eindeutig beleidigende Nachricht verfasst und diese fälschlicherweise nicht erkannt, was zu einem FN führt. Das Durchlassen der Beleidigung hat im echten Leben meist mehr Bedeutsamkeit als ein FP, welcher eine Nachricht als beleidigend einschätzt, obwohl sie es nicht ist. Der gute Score ohne Preprocessing kann damit zusammenhängen, dass dieses Modell sich Worthäufigkeiten als Merkmal zum Klassifizieren nimmt und davon ausgeht, dass diese unabhängig sind. Der inhaltsbasierte Klassifikator hat sich bereits in anderen Studien als effektiver Klassifikator erwiesen. [Cha]

Auch der Algorithmus des Modells *Logistic Regression* schneidet im Vergleich gut ab und zeigt den größten positiven Einfluss des angepassten Preprocessings mit einer Verbesserung des F1-Scores von $\sim 2\,\%$. Die anderen Modelle zeigen zwar auch bessere Ergebnisse, allerdings in einem geringeren Umfang.

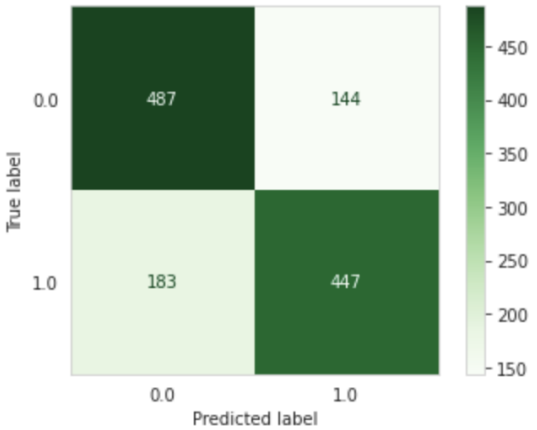

Abbildung 5.3: Konfusionsmatrix des Naive Bayes Modells

1 Basis für KI-Benchmark [Con19]

Beim anderen Datensatz 5.2 werden insgesamt wesentlich schlechtere Ergebnisse erzielt. Doch es zeigt sich auch hier die Tendenz, dass Datenbereinigung einen positiven Einfluss darstellt. Hier schneidet die *Support Vector Machine* mit dem angepassten Preprocessing gut ab, wird allerdings erneut vom Naive Bayes Modell überholt. Im folgenden wird für die tiefergehendere Analyse der Datensatz der Rheinischen Post bevorzugt, da dieser nicht nur bessere Werte liefert, sondern auch eine wissenschaftlichen Grundlage hat und so mehr Reliabilität hat.

5.2.2 Relevanz von Stopwords

Die Erkennung und Entfernung von Stopwords soll mehr Fokus auf die wesentliche Aussage des Satzes legen, indem Wörter ohne Bedeutung entfernt werden. Durch den Abgleich mit der Stopword-Liste des ntlk-Korpus für deutsche Sprache werden jene Wörter herausgefiltert.

Die Ergebnisse in der Tabelle 5.3 sprechen dafür, dass das *Stop Word Removal* (SWR) einen positiven Einfluss auf den F1-Score haben kann. Es scheint sowohl von den Daten als auch dem Modell abhängig zu sein. Erneut werden bei der *Logistic Regression* mitunter die besten Ergebnisse und größten Entwicklungen festgestellt.

5.2.3 Relevanz von Lemmatization

Da der Lemmatizer des PP1, *WordNetLemmatizer*, nicht für die deutsche Sprache geeignet ist, wird im PP2 *Simplemma* verwendet. Für die Auswertung wird also das PP2 inklusive SWR durchgeführt. Der variierende Faktor ist der Schritt der Lemmatization.

Anhand der Ergebnisse in Tabelle 5.4 kann eindeutig abgelesen werden, dass die Konvertierung der einzelnen Wörter in ihre Ursprungsform bei dem F1-Score konstant für eine erneute Verbesserung von mehr als 2 % sorgt. Dies gilt für alle Modelle.

Trotz der Steigerung von $\sim 3.2\,\%$ schneidet der Klassifikator mit XGBoost erneut am schlechtesten ab. Dem kann zugrunde liegen, dass der Datensatz nicht umfangreich

Modell	**PP1 ohne SWR**	**PP1 mit SWR**	**PP2 ohne SWR**	**PP2 mit SWR**
SVM	0.705036	0.700159	0.721154	**0.723065**
XGBoost	0.633043	0.630662	**0.644097**	0.643404
LogReg	0.710547	0.714512	0.727201	**0.731166**
Naive Bayes	0.727201	0.730373	**0.74544**	0.73751

Tabelle 5.3: Einfluss von SWR auf den F1-Score für den RP-Datensatz

genug für das Modell ist. Es ist bewusst darauf ausgelegt, besonders große Datensätze zu bearbeiten und bei kleinen keine überragende Leistung zu zeigen.

Abschließend ist festzuhalten, dass die angepasste Lemmatization einen eindeutig positiven Einfluss auf alle Klassifikatoren hat.

5.2.4 Idealtypischer Ablauf eines PP

Anhand der evaluierten Daten und Variation einiger Parameter stellt sich heraus, welche Schritte im Preprocessing am effektivsten abschneiden. Daraus lässt sich eine Preprocessing-Pipeline erstellen, welche hier visualisiert wird 5.4. Sie stellt die besten Ergebnisse anhand des durchgeführten Experiments dar.

Festzuhalten ist, dass die vorgestellte Pipeline in Kombination mit dem Modell *Logistic Regression* ab besten funktioniert hat, während der Naive Bayes Klassifikator mit den Rohdaten bessere Ergebnisse erzielt hat.

Modell	PP2 ohne Lemma	PP2 mit Lemma
SVM	0.723065	**0.752809**
XGBoost	0.643404	**0.675127**
LogReg	0.731166	**0.750198**
Naive Bayes	0.73751	**0.75337**

Tabelle 5.4: Einfluss von Lemmatization auf den F1-Score für den RP-Datensatz

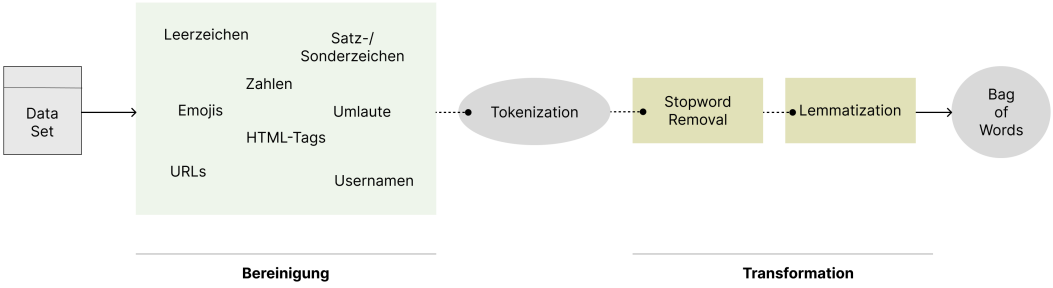

Abbildung 5.4: Preprocessing Pipeline

6 Abschluss

Abschließend wird die Arbeit zunächst rekapituliert. Die Problemstellung vom Anfang wird aufgegriffen, die zentralen Ergebnisse werden genannt und die Forschungsfragen beantwortet. Anschließend werden die methodische Vorgehensweise kritisch beleuchtet und interne sowie externe Validitätsgefährdungen in Betracht gezogen.
Als letztes wird ein Ausblick auf die praktische Implementierung der Resultate gegeben. Der letzte Punkt wird behandeln, welche anschließende Forschung zur Validierung der Arbeit nötig ist.

6.1 Zusammenfassung

Ziel dieser Forschungsarbeit war es, die begrenzten vorhanden Ressourcen für das Preprocessing von Daten für Modelle der künstlichen Intelligenz, speziell NLP-Aufgaben, zu sammeln und zu ergänzen. Diese sollen so getestet werden, dass am Ende Aussagen für eine angemessene Kombination dieser innerhalb einer Preprocessing Pipeline in etablierten ML Modellen getroffen werden können.

Zunächst wurden qualitativ hochwertig erhobene Daten gewählt und explorativ untersucht, um mit einer Datenanalyse fortzufahren. In der Analyse werden die Textobjekte bereinigt und aufbereitet. Die Wahl der Tools und Programme sowie die Anforderungen werden dargelegt und an einem Praxisbeispiel angewandt. Anschließend werden die Ergebnisse diskutiert und reflektiert, um daraus Erkenntnisse zur Beantwortung der Forschungsfragen zu ziehen.

- FF1: In bisherigen experimentellen Forschungen wurde der Einfluss des Preprocessings in einem ähnlichen Kontext und anderen Sprachen untersucht. Dabei wurde bereits belegt, dass Preprocessing bei der Verarbeitung von Daten im Natural Language Processing von hoher Relevanz ist und zu besseren Ergebnissen führt. In einer Analyse zu türkischer Sprache wirkt sich das Stemming und Lemmatisieren sehr positiv auf den F1-Score aus, auch wenn die Ressourcen dazu entsprechend begrenzt sind.

- FF2: Nach genauerer Betrachtung der Möglichkeiten im Preprocessing zeigten sich einige Schritte als sprachunabhängig. Andere dagegen konnten nicht isoliert von der Sprache betrachtet werden und benötigen in einer Analyse Anpassung. Dabei handelt es sich um jene: Umlaute normalisieren, Zahlen, Sonderzeichen und Stopwords entfernen, Emojis enkodieren und Lemmatization (auf Seite 29).
  Es wurde herausgearbeitet und gesammelt, mit welchen Techniken und Bibliotheken diese praktisch umgesetzt werden können. Die Anwendung dieser fand experimentell im Kontext von einer Klassifizierung durch Machine Learning Algorithmen statt.

- FF3: In der experimentellen Untersuchung konnte festgestellt werden, dass die Modelle SVM und Naive Bayes für die Aufgabe der Klassifizierung insgesamt die besten Ergebnisse erzielten. Das angepasste Preprocessing hatte bei dem Modell Logistic Regression den größten Einfluss mit einer 2-prozentigen Erhöhung der Vorhersagewahrscheinlichkeit.
  Die Entfernung von Stopwords wurde weiterhin genauer untersucht. In diesem Falle hat die Preprocessing Technik einen unausgewogenen Einfluss und verhält sich in verschiedenen PP-Umgebungen unterschiedlich. Die Unterschiede sind allerdings um vernachlässigbar klein. Insgesamt am besten schneidet eine Kombination aus dem sprachspezifischen Preprocessing mit Stopword-Removal und Lemmatization bei allen Modellen am besten ab.

6.2 Kritische Auswertung

Hypothese verifizieren

Die Hypthese, dass ein sprachspezifisches Preprocessing einen signifikaten Unterschied in der Qualität der Ergebnisse von Sentimentanalysen macht, kann somit mit einzelnen Einschränkungen bestätigt werden. Im Gesamten habe sich Verbesserungen gezeigt, diese sind allerdings von Modell zu Modell unterschiedlich signifikant. Dies liegt an der Arbeitsweise der Algorithmen und wie unabhängig sie Tokens auf Wortebene behandeln.

Validitätsgefährdungen

Es sind zu $\sim 10\,\%$ längere Textnachrichten dabei, die über den definierten Standard für Kurznachrichten hinausgehen, allerdings wäre die Quantität nach Eliminierung der unpassenden unzureichend, um ein Machine Learning Modell zu trainieren. Hier soll jedoch die Differenzierung gemacht werden zwischen Standards von einem Kurznachrichtendienst und wie die Länge von Nachrichten in einer realen Umgebung aussehen, wo durchaus längere Texte auftreten können.

Eine weitere externe Validitätsgefährdung stellt der multilinguale Datensatz dar. Während dieser ordentlich und reliabel wirkt, ist die Quelle nicht nachvollziehbar und ebenso nicht wissenschaftlich. Vor allem betrifft es die Kategorisierung und damit können Ergebnisse der Sentimentanalyse beeinträchtigt werden, wenn die Kategorien nicht professionell bearbeitet wurden.

6.3 Weitere Ansätze

Ein weiterer interessanter Anwendungsfall ist, wie sich jenes erarbeitetes Preprocessing in multilingualen Szenarien verhält. Das XLM Modell ist dabei mit aktuell hoher Relevanz besonders von Interesse [Iri21]. Offen bleibt bisher die Frage, wie die Sprache eindeutig erkannt wird, sodass das PP sprachbezogen bleibt.
Langfristig gesehen eignet sich für größere Netzwerke vielleicht die Methode des *Cross-Lingual Transfer Learnings* besser, bei welcher ein Training auf einer Sprache mit großem Korpus stattfindet und die Auswertungen über einen Übersetzer laufen. Dies kann effizienter und kostengünstiger sein, jedoch weniger akkurat. Diese Abwägung muss individuell nach Anwendungsgröße durchgeführt werden. [Iri21]

Außerdem zu untersuchen ist, wie sich der Ansatz in *Unsupervised Models* und in Deep Learning Modellen auswirkt. Es gibt Hinweise darauf, dass das Preprocessing dabei einen anderen Einfluss hat als in den hier verwendeten *Supervised Models* aus dem Machine Learning [Ala19]. Dabei sollten mindestens BERT, LSTM und CNN in Betracht gezogen werden, da diese aktuell die leistungsstärksten Modelle darstellen [Cie17]. Das Modell RNN (Recurrent Neural Network) sollte dabei außen vor gelassen werden, da dieses neuronale Netzwerk als nicht zukunftsfähig gilt. Es wird durch Transformatoren abgelöst, da diese trotz ähnlich guten Ergebnissen deutlich weniger ressourcenintensiv sind [Vas].

6.4 Ausblick

Mit den gewonnenen Erkenntnissen kann nun ein Machine Leraning Algorithmus mit einem sprachspezifischen Preprocessing innerhalb eines Chatsystems implementiert werden. Wie dieser effektiv genau verwendet wird, ist weiterhin sozialwissenschaftlich betrachtet zu untersuchen. Es kann sinnvoll sein, unfreundliche Nachrichten zu erkennen, sie aber nicht zwingend zu zensieren, da Zensur in Bezug auf die Meinungsfreiheit ein umstrittenes Thema ist [Iri21].

Literaturverzeichnis

[Ala19] Alam, Saqib und Yao, Nianmin: The impact of preprocessing steps on the accuracy of machine learning algorithms in sentiment analysis. *Computational and Mathematical Organization Theory* (2019), Bd. 25(3): S. 319–335

[Ale09] Alec Go; Huang Lei und Richa Bhayani: *Twitter sentiment classification using distant supervision: Processing 150* (2009)

[Ass] Assenmacher, Dennis; Niemann, Marco; Müller, Kilian; Seiler, Moritz V.; Riehle, Dennis M. und Trautmann, Heike: RP-Mod & RP-Crowd: Moderator- and Crowd-Annotated German News Comment Datasets

[Bal22] BS Program (2022), URL https://www.buitms.edu.pk/depart/uafa/documents/25-05-2022/BEEF%20BS%20List.pdf

[Bra] Brabec, Jan; Komárek, Tomáš; Franc, Vojtěch und Machlica, Lukáš: On Model Evaluation under Non-constant Class Imbalance, URL http://arxiv.org/pdf/2001.05571v2

[Bru04] Brugmann und Karl: Kurze vergleichende Grammatik der indogermanischen Sprachen (1904)

[CC18] Camacho-Collados, Jose und Taher Pilehvar, Mohammad: On the Role of Text Preprocessing in Neural Network Architectures: An Evaluation Study on Text Categorization and Sentiment Analysis (2018): S. 40–46, URL https://aclanthology.org/W18-5406.pdf

[Cha] Chandrasekar, Priyanga und Qian, Kai: The Impact of Data Preprocessing on the Performance of a Naive Bayes Classifier, S. 618–619, URL https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7552291

[Che16] Chen, Tianqi und Guestrin, Carlos: XGBoost: A Scalable Tree Boosting System (2016), Bd. 11(34): S. 785–794, URL http://arxiv.org/pdf/1603.02754v3

[Cie17] Cieliebak, Mark; Deriu, Jan Milan; Egger, Dominic und Uzdilli, Fatih: A Twitter Corpus and Benchmark Resources for German Sentiment Analysis, in: Lun-Wei Ku und Cheng-Te Li (Herausgeber) *Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media*, Association for Computational Linguistics, Stroudsburg, PA, USA, S. 45–51, URL https://aclanthology.org/W17-1106.pdf

[Con19] Congalton, Russell G. und Green, Kass: *Assessing the Accuracy of Remotely Sensed Data*, CRC Press (2019)

[Cro06] Crone, Sven F.; Lessmann, Stefan und Stahlbock, Robert: The impact of preprocessing on data mining: An evaluation of classifier sensitivity in direct marketing. *European Journal of Operational Research* (2006), Bd. 173(3): S. 781–800, URL https://www.researchgate.net/publication/222824928_The_impact_of_preprocessing_on_data_mining_An_evaluation_of_classifier_sensitivity_in_direct_marketing

[Dal21] Dalianis, Hercules: Master Thesis Proposals: Evaluation of preprocessing Natural Language Processing (NLP) software (2021), URL https://people.dsv.su.se/~hercules/three-more-master-thesis-proposals.pdf

[Dew] Dewint, Christelle: Europeans and their Languages: ANNEX

[Dom20] Dominik Schlomo Moog: *Spam-Erkennung in Crowdsourced Ideation*, Dissertation, Freie Universität Berlin, Berlin (16.01.2020)

[Gar22] Garg, Neha und Sharma, Kamlesh: Text pre-processing of multilingual for sentiment analysis based on social network data. *International Journal of Electrical and Computer Engineering (IJECE)* (2022), Bd. 12(1): S. 776

[Ily19] Ilyas, Ihab F. und Chu, Xu: *Data Cleaning*, Bd. 28 von *ACM Digital Library*, Association for Computing Machinery and Morgan & Claypool Publishers, New York, NY, first edition Aufl. (2019), URL https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=6954840

[Iri21] Irina Bigoulaeva; Viktor Hangya und Alexander Fraser: Cross-Lingual Transfer Learning for Hate Speech Detection (2021)

[Jul68] Julie Beth Lovins: Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics* (1968), (vol.11, nos.1 and 2)

[Kal21] Kalaivani, Rama und Ramesh, Marivendan: The Effect of Stop Word Removal and Stemming In Datapreprocessing (2021), (Vol. 25, Issue 6), URL https://www.proquest.com/openview/7d09e95c5e0d540c8765cf0d41a7ce39/1.pdf?cbl=2031963&pq-origsite=gscholar

[Kan20] Kant, Rajni: Tweeter Hate Speech Sentiment Analysis: Tweeter Sentiment Analysis (2020), URL https://www.kaggle.com/datasets/rajnikant2020/tweeter-hate-speech-sentiment-analysis?select=test_tweets.csv

[Kli] Klinger, Roman: Strukturierte Modellierung von Affekt in Text

[Lid15] Lidwell, William; Holden, Kritina und Butler, Jill: *The pocket universal principles of design: 150 essential tools for architects, artists, designers, developers, engineers, inventors, and makers*, Rockport Publ, Beverly, Mass., 1. publ Aufl. (2015)

[Mag16] Magliani, Federico; Fontanini, Tomaso; Fornacciari, Paolo; Manicardi, Stefano und otti, Eleonora: A Comparison between Preprocessing Techniques for Sentiment Analysis in Twitter (2016), URL https://www.researchgate.net/publication/311615347_A_Comparison_between_Preprocessing_Techniques_for_Sentiment_Analysis_in_Twitter

[Mee21] Meetei, Loitongbam Sanayai; Singh, Thoudam Doren; Borgohain, Samir Kumar und Bandyopadhyay, Sivaji: Low resource language specific pre-processing and features for sentiment analysis task. *Language Resources and Evaluation* (2021), Bd. 55(4): S. 947–969

[Mic20] Michael: Natural Language Processing of German texts: Part 1: Using machine-learning to predict ratings (2020), URL https://data-dive.com/german-nlp-binary-text-classification-of-reviews-part1

[Múj08] Újdricza, Éva und Syrota, Ganna: IR_Referat_Stemming_MujdriczaSyrota (04.02.2008), URL https://www.cl.uni-heidelberg.de/~mujdricz/software/hunPort/IR_Referat_Stemming_MujdriczaSyrota.pdf

[Mul18] Mulki, Hala; Haddad, Hatem; Ali, Chedi Bechikh und Babaoglu, Ismail: Preprocessing impact on Turkish sentiment analysis, in: *2018 26th Signal Processing and Communications Applications Conference (SIU)*, IEEE, S. 1–4

[Sar16] Sarkar, Dipanjan: *Text Analytics with Python*, Apress, Berkeley, CA (2016)

[Sar18] Sarkar, Dipanjan; Bali, Raghav und Sharma, Tushar: *Practical Machine Learning with Python*, Apress, Berkeley, CA (2018)

[Sch20] Schweizerische Eidgenossenschaft: Empfehlungen für den Umgang mit Anglizismen in deutschsprachi- gen Texten des Bundes (2020)

[Sch22] Schumacher, Mareike und Vauth, Michael (Herausgeber): *forTEXT: Literatur digital erforschen* (2022)

[Sie] Siegel, Melanie: Sentimentanalyse für die deutsche Sprache

[Sou22] Soumya S und Pramod K V: Hybrid Deep Learning Approach for Sentiment Classification of Malayalam Tweets. *(IJACSA) International Journal of Advanced Computer Science and Applications* (2022): S. 891–899, URL https://thesai.org/Downloads/Volume13No4/Paper_103-Hybrid_Deep_Learning_Approach_for_Sentiment_Classification_of_Malayalam_Tweets.pdf

[Tia16] Tianqi, Chen und Guestrin, Carlos: XGBoost: A Scalable Tree Boosting System (2016), URL https://github.com/dmlc/xgboost/tree/master/demo#machine-learning-challenge-winning-solutions

[Uni22] Universität Zürich: Logistische Regressionsanalyse (2022), URL https://www.methodenberatung.uzh.ch/de/datenanalyse_spss/zusammenhaenge/lreg.html#3._Logistische_Regressionsanalyse_mit_SPSS

[Val19] Vala, Jay Dilipbhai: *Classification of Multilingual Legal Text Using Deep Learning: Evaluation of General-Purpose Resources for Legal Domain-Specific Task*, Ma, Otto von Guericke Universität Magdeburg, Magdeburg (13.05.2019), URL https://wwwiti.cs.uni-magdeburg.de/iti_db/publikationen/ps/auto/Vala2019thesis.pdf

[Van13] Van de Kauter, Marjan: LeTs Preprocess: The multilingual LT3 linguistic preprocessing toolkit. *Computational Linguistics in the Netherlands Journal 3* (2013): S. 103–120

[Vas] Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Gomez, Aidan N.; Kaiser, Lukasz und Polosukhin, Illia: Attention Is All You Need, URL http://arxiv.org/pdf/1706.03762v5

[War] Wartena, Christian: A Probabilistic Morphology Model for German Lemmatization

[Wei20] Weissweiler, Leonie und Fraser, Alexander: Developing a Stemmer for German Based on a Comparative Analysis of Publicly Available Stemmers (2020), URL https://www.cis.lmu.de/~weissweiler/cistem/

[Wes19] Westermeier, Thomas: *Multi-Class Classification: Vergleich von Support Vector Machine mit anderen ausgewaehlten Algorithmen*, Dissertation, Ludwig-Maximilians-Universitaet Muenchen (2019), URL https://epub.ub.uni-muenchen.de/68478/1/BA_Westermeier.pdf

[Wol22a] Wolfewicz, Arne: Deep Learning vs. Machine Learning – What's The Difference? (2022), URL https://levity.ai/blog/difference-machine-learning-deep-learning

[Wol22b] Wolfewicz, Arne: How Do Machines Learn? A Beginners Guide (2022), URL https://levity.ai/blog/how-do-machines-learn

Abbildungsverzeichnis

Tabellenverzeichnis

A Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import nltk

from wordcloud import WordCloud,STOPWORDS
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer,
                                            TfidfVectorizer

# GERMAN STOP WORDS
from nltk.corpus import stopwords
german_stop_words = stopwords.words('german')

# OTHER PRETRAINED MODELS
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import f1_score, roc_auc_score
from sklearn.pipeline import make_pipeline


# RP DATA
train = pd.read_csv("/kaggle/input/rpcrowmod/RP-Crowd-3.csv")

# MULTILING DATA
#train = pd.read_csv("/kaggle/input/multilingual-hatespeech-dataset/
                                            Dataset/Training/
                                            MultiLanguageTrainDataset.csv")
#train = train[lambda row: row['language'] == 5]

# TEST DATA
test = pd.read_csv("/kaggle/input/rpcrowmod/RP-Crowd-3.csv")
#test = pd.read_csv("/kaggle/input/multilingual-hatespeech-dataset/
                                            Dataset/Testing/Test1/German_test.
                                            csv")  train.head(5)  train.sample(
                                            10)  test.head(5)  train.info()
                                            test.info()  import re
def process_text(text):
return " ".join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])", " ",text.
                                            lower()).split())  train['
```

```
                                        processed_texts'] = train['text'].
                                        apply(process_text)
34  test['processed_texts'] = test['text'].apply(process_text)
35
36  train = train.rename(columns = {'Text':'text'})  train['text'].isna().
                                        sum()
37  train['label'].value_counts() #count number of positive and negative
                                        comment
38  train['length'] = train['text'].apply(len)
39
40  [...]
41
42  #CountVectorizer will convert a collection of text documents to a
                                        matrix of token counts
43  #Initialize
44  vector = CountVectorizer()
45  #We fit and transform the vector created
46  frequency_matrix = vector.fit_transform(table.text)
47  #Sum all the frequencies for each word
48  sum_frequencies = np.sum(frequency_matrix, axis=0)
49  #Now we use squeeze to remove single-dimensional entries from the shape
                                         of an array that we got from
                                        applying np.asarray to
50  #the sum of frequencies.
51  frequency = np.squeeze(np.asarray(sum_frequencies))
52  #Now we get into a dataframe all the frequencies and the words that
                                        they correspond to
53  frequency_df = pd.DataFrame([frequency], columns=vector.
                                        get_feature_names()).transpose()
54  return frequency_df
55
56  def graph(word_frequency, sent):
57  labels = word_frequency[0][1:51].index
58  title = "Word Frequency for %s" %sent
59
60  [...]
61
62  # GERMAN STOP WORDS
63  from nltk.corpus import stopwords
64  german_stop_words = stopwords.words('german')
65  stopwordlist = german_stop_words
66
67  !pip install simplemma
68  from nltk.stem import WordNetLemmatizer
69  import simplemma
70
71  def preprocess(textdata):
72  processedText = []
73
74  # Create Lemmatizer and Stemmer.
```

```
75  wordLemm = WordNetLemmatizer()
76
77  # Defining regex patterns.
78  urlPattern        = r"((http://)[^ ]*|(https://)[^ ]*|( www\.)[^ ]*)"
79  userPattern       = '@[^\s]+'
80  alphaPattern      = "[^a-zA-Z0-9]"
81  sequencePattern   = r"(.)\1\1+"
82  seqReplacePattern = r"\1\1"
83
84  for text in textdata:
85  text = text.lower()
86
87  # Replace all URls with 'URL'
88  text = re.sub(urlPattern,' URL',text)
89  # Replace all emojis.
90  for emoji in emojis.keys():
91  text = text.replace(emoji, "EMOJI" + emojis[emoji])
92  # Replace @USERNAME to 'USER'.
93  text = re.sub(userPattern,' USER', text)
94  # Replace all non alphabets.
95  text = re.sub(alphaPattern, " ", text)
96  # Replace 3 or more consecutive letters by 2 letter.
97  text = re.sub(sequencePattern, seqReplacePattern, text)
98
99  textwords = ''
100 for word in text.split():
101 # Checking if the word is a stopword.
102 if word not in stopwordlist:
103 if len(word)>1:
104 # ENGLISH: Lemmatizing the word.
105 #word = wordLemm.lemmatize(word)
106 # GERMAN: Lemma
107 word = simplemma.lemmatize(word, lang='de')
108 # Re-Add word to sentence
109 textwords += (word+' ')
110
111 processedText.append(textwords)
112
113 return processedText  # WITH standard PP
114 #train['processed_texts'] = train['text'].apply(process_text)
115
116 # WITH PP1
117 #train['processed_texts'] = train['text'].apply(clean_text)
118
119 # WITH improved PP2
120 text = list(train['text'])
121 text[:3]
122 train['processed_texts'] = preprocess(text)
123
124 # WITHOUT PP
```

```
125 #train['processed_texts'] = train['text']  train.head(5)  train['
                                    Subjectivity'] = train['
                                    processed_texts'].apply(
                                    getSubjectivity)
126 train['Polarity'] = train['processed_texts'].apply(getPolarity)
127 train['Analysis'] = train['Polarity'].apply(getAnalysis)
128
129 train[lambda x: x['Analysis'] == 'Negative']  normal_words = ' '.join([
                                    word for word in train['
                                    processed_texts'][train['label'] ==
                                     0]])
130
131 from sklearn.model_selection import train_test_split
132 x_train, x_test, y_train, y_test = train_test_split(train["
                                    processed_texts"], train["label"],
                                    test_size = 0.2, random_state = 42)
                                      from sklearn.feature_extraction.
                                    text import CountVectorizer,
                                    TfidfTransformer
133 count_vect = CountVectorizer(stop_words=german_stop_words)
134 transformer = TfidfTransformer(norm='l2',sublinear_tf=True)
                                    x_train_counts = count_vect.
                                    fit_transform(x_train)
135 x_train_tfidf = transformer.fit_transform(x_train_counts)  print(
                                    x_train_counts.shape)
136 print(x_train_tfidf.shape)  x_train_counts  x_test_counts = count_vect.
                                    transform(x_test)
137 x_test_tfidf = transformer.transform(x_test_counts)  print(
                                    x_test_counts.shape)
138 print(x_test_tfidf.shape)  from sklearn.ensemble import
                                    RandomForestClassifier
139 model = RandomForestClassifier(n_estimators=100)
140 model.fit(x_train_tfidf,y_train)  predictions = model.predict(
                                    x_test_tfidf)  from sklearn.metrics
                                     import confusion_matrix,f1_score
141 from sklearn import metrics
142 cm = confusion_matrix(y_test,predictions)  print(cm)  print(metrics.
                                    classification_report(y_test,
                                    predictions))  rf_f1=f1_score(
                                    y_test,predictions)
143 rf_f1  from sklearn.metrics import accuracy_score
144 accuracy_score(y_test, predictions)  from sklearn.metrics import
                                    precision_score,recall_score,
                                    f1_score
145 f1_score(y_test,predictions)  precision_score(y_test,predictions)
                                    recall_score(y_test, predictions)
                                    # jaccard_index
146 from sklearn.metrics import jaccard_score
147 jaccard_score(predictions,y_test)  # Log Loss
148 from sklearn.metrics import log_loss
```

```
log_loss(y_test,predictions)  # ROC_AUC_Score
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,predictions)  from sklearn import svm
lin_clf = svm.LinearSVC()
lin_clf.fit(x_train_tfidf,y_train)  predict_svm = lin_clf.predict(
                                    x_test_tfidf)  from sklearn.metrics
                                     import confusion_matrix,f1_score
confusion_matrix(y_test,predict_svm)  svm_f1=f1_score(y_test,
                                    predict_svm)
svm_f1  from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(train["text"],
                                    train["label"], test_size = 0.20,
                                    random_state = 42)  from sklearn.
                                    feature_extraction.text import
                                    CountVectorizer, TfidfTransformer
count_vect = CountVectorizer(stop_words= german_stop_words)
transformer = TfidfTransformer(sublinear_tf=True)

X_train_cnt = count_vect.fit_transform(X_train)
X_train_TF = transformer.fit_transform(X_train_cnt)
print(X_train_cnt.shape)
print(X_train_TF.shape)

X_test_cnt = count_vect.transform(X_test)
X_test_TF = transformer.transform(X_test_cnt)
print(X_test_cnt.shape)
print(X_test_TF.shape)  from sklearn.ensemble import
                                    RandomForestClassifier as RFClass
ranForModel = RFClass(n_estimators=10, criterion = "entropy")
ranForModel.fit(X_train_TF, Y_train)

from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report as class_re
from sklearn.metrics import confusion_matrix as c_m
ranForPredict = ranForModel.predict(X_test_TF)
print("Predicted Class:",ranForPredict)
print("Confusion Matrix:\n",c_m(Y_test ,ranForPredict))
print("Accuracy:", accuracy_score(Y_test ,ranForPredict))
print("F_score:", f1_score(Y_test ,ranForPredict))
print("Classification Report:\n",class_re(Y_test ,ranForPredict))

[...]

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score as acc_score
from sklearn.metrics import classification_report as class_re
from sklearn.preprocessing import StandardScaler

```

```
train.drop('text',inplace = True, axis = 1)
train.Analysis = train.Analysis.map({"Neutral":0, "Negative":-1, "
                                     Positive":+1})
col_names = ["Subjectivity", "Polarity", "Analysis"]
target_name = ["label"]


X = train[col_names]
X_std_scal = StandardScaler().fit_transform(X)
Y = train[target_name]

X_train, X_test, Y_train, Y_test = train_test_split(X_std_scal, Y,
                                     test_size = 0.20, random_state = 42
                                     )

DTreeClass = DecisionTreeClassifier(criterion = "entropy", random_state
                                      = 42, max_depth = 7)
DTreeClass.fit(X_train, Y_train)
Y_pred = DTreeClass.predict(X_test)
Y_scored = DTreeClass.score(X,Y)
Y_scored2 = DTreeClass.score(X_train,Y_train)
Y_scored3 = DTreeClass.score(X_test,Y_test)


print("Classification Report:\n",class_re(Y_test, Y_pred))
print("Confusion Matrix:\n",c_m(Y_test ,Y_pred))
print("F_score:", f1_score(Y_test ,Y_pred))
print("Accuracy:", acc_score(Y_test, Y_pred))
print("Predicted Class:",Y_pred)
print("Scored Class (From all data):",Y_scored)
print("Scored Class (From training data):",Y_scored2)
print("Scored Class (From testing data):",Y_scored3)  #splitting into
                                     train and test
train, test= train_test_split(train, test_size=0.2, random_state=42)
Xtrain, ytrain = train['processed_texts'], train['label']
Xtest, ytest = test['processed_texts'], test['label']  #Vectorizing
                                     data

tfidf_vect = TfidfVectorizer() #tfidfVectorizer
Xtrain_tfidf = tfidf_vect.fit_transform(Xtrain)
Xtest_tfidf = tfidf_vect.transform(Xtest)


count_vect = CountVectorizer() # CountVectorizer
Xtrain_count = count_vect.fit_transform(Xtrain)
Xtest_count = count_vect.transform(Xtest)  import pandas_profiling

#[...imports]
%matplotlib inline  from xgboost import XGBClassifier
model_bow = XGBClassifier(random_state=22,learning_rate=0.9)
```

```
model_bow.fit(x_train_tfidf,y_train)  predict_xgb = model_bow.predict(x_test_tfidf)  from sklearn.metrics import confusion_matrix,f1_score
confusion_matrix(y_test,predict_xgb)  from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, plot_confusion_matrix, plot_roc_curve, plot_precision_recall_curve

plot_confusion_matrix(model_bow, x_train_tfidf, y_train,cmap = 'Greens')
plt.grid(False)  xgb_f1=f1_score(y_test,predict_xgb)
xgb_f1  from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, plot_confusion_matrix, plot_roc_curve, plot_precision_recall_curve

lr = LogisticRegression()
lr.fit(Xtrain_tfidf,ytrain)
p1=lr.predict(Xtest_tfidf)
s1=accuracy_score(ytest,p1)
print("Logistic Regression Accuracy :", "{:.2f}%".format(100*s1))
plot_confusion_matrix(lr, Xtest_tfidf, ytest,cmap = 'Greens')
plt.grid(False)  from sklearn.naive_bayes import MultinomialNB

mnb= MultinomialNB()
mnb.fit(Xtrain_tfidf,ytrain)
p2=mnb.predict(Xtest_tfidf)
s2=accuracy_score(ytest,p2)
print("Multinomial Naive Bayes Classifier Accuracy :", "{:.2f}%".format(100*s2))
plot_confusion_matrix(mnb, Xtest_tfidf, ytest,cmap = 'Greens')
plt.grid(False)

results = {'RandomForest':rf_f1, 'XgBoost':xgb_f1,'SVM':svm_f1, 'LogReg':s1, 'NaiveBayes': s2}
df = pd.DataFrame(results, index =['f1Score'])
df

[...]
```