|  |  |  |
| --- | --- | --- |
| Projekt-  dokumentation |  |  |

Abschlussarbeit zur Vollendung des Volontariats   
und zum erfolgreichen Erwerb des Zertifikats zum  
„wissenschaftlicher Dokumentar / Information Specialist“

**Spezifikation und PoC der KI-gestützten**

**„Beta-Klassen-Recognition“ in der Presserecherche**



**Volontärjahrgang 2022**

|  |  |
| --- | --- |
| vorgelegt von: | Timo Schumacher B.A., M.A. |
|  | Turmstr. 89  55120 Mainz |
|  |  |
|  |  |
| Volontariats-/Traineepartner: | SWR |
| Referent: | Prof. Geribert E. Jakob |  |

Zusammenfassung / Abstract

**1. Zielsetzung**

**Was war die Aufgabenstellung was waren die Rahmenbedingungen**

Die Recherche nach Interviews, Chronologien oder Kommentaren in Presseartikeln gehört zum Alltag am IDA-Desk des Südwestrundfunks (SWR). Im jetzigen System zur automatischen Verschlagwortung von Pressetexten werden nur Inhaltsklassen und Entitäten maschinell indexiert. Präsentationsformen werden bislang nur in Einzelfällen händisch indexiert. Dadurch wird die Recherche nach eben diesen erschwert und ist mit einem erhöhten Zeitaufwand verbunden. Die konkrete Suche nach Präsentationsformen ermöglicht es, schnell inhaltliche Schneisen zu schlagen. PAN soll ab dem 01.01.2023 durch MDH:Presse abgelöst werden. Dabei werden die bestehenden Daten in das neue System importiert und die PAN-spezifischen Präsentationsformen auf Normdatenbank-konforme Formen gemappt. Mit dem Umzug in MDH:Presse wird auch das aktuell eingesetzte Textmining abgelöst. In der zukünftigen Crossmedialen Suche in Medas kommen diverse Mining-Services zum Einsatz.   
Der Service soll die Recherche nach Beta-Klassen ermöglichen und dadurch die Qualität der Rechercheergebnisse für Rechercheure\*innen steigern. Dazu soll ein Proof of concept (PoC) erstellt werden, indem verschiedene in Frage kommende Technologien zur Kategorisierung von Textdaten getestet werden. Geplant ist das Testen von Modellen auf Basis von Support Vektor Maschinen (SVM) und Logistischer Regression, sowie Deep-Learning Netzwerken auf Transformer-Architektur wie zum Beispiel BERT. Als Trainingsmaterial dienen zunächst Presseartikel aus PAN. Die Modellanforderungen sind bislang noch nicht spezifiziert, da beim Einsatz von KI bzw. Machine Learning in den seltensten Fällen hun­dert­pro­zen­tige Genauigkeit erreicht werden kann.

**2. (vereinbarte) Ergebnistypen**

Folgende Ergebnistypen wurden als relevant für die erfolgreiche Durchführung des Projekts identifiziert und vereinbart:

* Auswahl, Testung und Finetuning verschiedener KI-Modelle zur Erkennung von Präsentationsformen in Pressetexten
* Ausarbeitung eines PoCs zur Weitergabe an die Entwickler\*innen der Mining-Plattform
* Erstellung einer Spezifikation des KI-Modells
* Einbettung des Modells in einen Service, welcher in die Mining-Plattform von Medas implementiert werden kann

**3. (gelieferte) Ergebnistypen**

Folgende Ergebnistypen wurden erarbeitet und geliefert:

* Tests verschiedener KI-Modelle zur Erkennung von Präsentationsformen in Pressetexten
* Ausgearbeiteter Proof of Concept inklusive einer Spezifikation des KI-Modells mit weiteren Handlungsempfehlungen
* Dokumentierte Python Skripte zur Implementierung in die Mining-Plattform

**[Erläuterung und Bewertung der Ergebnisse aus der Spiegelstrichliste]** Der Proof of Concept mit Handlungsempfehlungen und dokumentierten Python-Skripten konnte den Stakeholdern erfolgreich übergeben werden. Die Prüfung steht noch aus, da die Implementierung und weitere Tests auf das Jahr 2023 verschoben worden sind. Die vorläufigen Ergebnisse sind vielsprechend und zeigen, dass die Bestimmung von Beta-Klassen auf Basis der Pressetexte eingeschränkt möglich ist und somit die Recherche erleichtern kann.

**4. Change Requests**

Die meisten Ergebnistypen konnten geliefert werden und das Projekt konnte in großen Teilen fertiggestellt werden. Verzögerungen im Prozessverlauf und vorstellbar-kritische Zwischenergebnisse führten zu einer Anpassung der Ergebnistypen. Die Implementierung in den Miningservice, sowie Tests von weiteren KI-Modellen wurden auf 2023 verschoben. Stattdessen wurden die programmierten Skripte dokumentiert und sind zur Weitergabe an die Entwickler\*innen der Mining-Plattform bereit.

Analytic Design and Results

…

Synopsis

Spezifikation, Entwicklung und Implementierung eines KI-basierten Services zur Bestimmung der Präsentationsformen von Texten für die Presserecherche.

Schlagwörter

Künstliche Intelligenz, Textklassifikation, Verschlagwortung, Presse

Vorwort/Persönliche Danksagung

…

Inhaltsverzeichnis

Zusammenfassung / Abstract 1

Vorwort/Persönliche Danksagung 3

1 Einleitung 1

2 Projektbeschreibung und Einordnung 1

2.1 Zielsetzung und projektbezogene Ergebnistypen 5

2.2 Stakeholderanalyse 5

2.3 SWOT-Analyse 6

3 Grundlagen und Forschungsstand 8

3.1 Natürliche Sprachverarbeitung 8

3.2 Vorprozessierung 11

3.2.1 Segmentierung und Tokenisierung 11

3.2.2 Stammformreduktion und Lemmatisierung 18

3.2.3 Stoppwortentfernung 18

3.2.4 Part-of-speech Tagging 18

3.3 Text als Zahlen 18

3.3.1 Sprachmodelle 18

3.3.2 Vektorisierung und Gewichtung 18

3.4 Textklassifkation 22

3.4.1 Machine Learning zur Textklassifikation 22

3.4.2 Logistische Regression 26

3.4.3 Support Vektor Machine 31

3.4.4 BERT / Deep Learning 36

3.4.5 Evaluation von Modellen 36

3.4.6 Warum Python 36

4 Datengrundlage 37

4.1 Struktur 37

4.2 Statistiken 37

5 Methodik 40

5.1 Warum Python 40

5.2 Pipeline 41

5.3 Parameter 45

6 Ergebnisse 48

6.1 Proof of Concept 48

7 Schluss 49

7.1 Best practices 49

7.2 Lessons learned 49

7.3 Ausblick 49

8 Projektbeschreibung und Einordnung 50

Dies ist eine Überschrift ohne Zahl – Ebene 1 51

8.1 Dies ist eine Überschrift x.x (z.B. 1.1.) – Ebene 2 51

8.2 Zielsetzung und projektbezogene Ergebnistypen 51

8.3 Konkrete Lieferobjekte 51

8.3.1 Dies ist eine Überschrift n.n.n – Ebene 3 51

8.4 Analyse, Ist-Zustand und Rahmenbedingungen 52

8.5 Brauch ich nicht Vorgehensweise und Methoden zur Bedarfsanalyse 52

9 Analyse- und Gestaltungsteil 53

9.1 Stakeholderanalyse 53

9.2 SWOT-Analyse 53

Quellen 54

Abbildungen 55

Tabellen 56

Abkürzungen und Akronyme 57

Anhang 58

Eidesstattliche Erklärung 59

# Einleitung

Um mit wachsenden Datenmengen umzugehen sind die privaten und öffentlich-rechtlichen Rundfunkarchive einerseits gefordert Innovationen voranzutreiben, um sie zu verarbeiten. Andererseits sehen sie sich mit strengen Kostensparzielen konfrontiert, die die Kapazitäten in Form von human Ressourcen für die Entwicklung solcher Innovationen beschränken.

Technische Innovationen beruhen dabei häufig auf Verfahren des maschinellen Lernens. Ein Teilgebiet davon ist die natürliche Sprachverarbeitung oder im Englischen *Natural Language Processing* (**NLP**). Viele Archivanwendungen inkorporieren bereits Verfahren zur Verarbeitung von natürlicher Sprache. In der ARD werden für das Audiomining beispielweise *Speech-To-Text*-Verfahren angewendet, um Wort- und Bewegtbildbeiträge zu verschriftlichen. Auf Grundlage dieser Transkripte können Rechercheur\*innen Suchabfragen in den jeweiligen Datenbanken starten. Dokumentar\*innen nutzen die Transkripte dagegen für die Erschließung. Die Transkripte können zudem als Grundlage für weitere Verfahren wie die Erkennung von Entitäten oder die inhaltliche Verschlagwortung (Indexierung) verwendet werden[[1]](#footnote-1).   
In den Workflows der Pressearchive sind diese Verfahren bereits fester Bestandteil. Dennoch muss an vielen Stellen noch händisch nachbearbeitet oder fehlende Metadaten ergänzt werden. Das liegt zum einen an der teilweise mangelnden Qualität der automatisierten Verfahren. Zum anderen liegt es daran, dass manche Aspekte noch nicht automatisch erschlossen werden können oder die bereits angesprochenen Kapazitäten fehlen, um Innovationen voranzutreiben. Aktuell werden deshalb die verschiedenen Arbeiten und Workflows priorisiert. Das hat zur Folge, dass nicht mehr alle Quellen voll erschlossen werden und somit potenziell-wichtige Metadaten in der Recherche fehlen oder den Dokumentar\*innen bei der Erschließung nicht zur Verfügung stehen. Deshalb sind stetige Innovationen und Verbesserungen der Workflows in den Archiven durch Verfahren der natürlichen Sprachverarbeitung von großer Bedeutung, da sie bereits jetzt viele Aufgaben ohne menschlichen Input übernehmen und der Trend hin zu weiteren vollautomatischen Anwendungen geht.  
So sagen \citet[1]{bengfortAppliedTextAnalysis2018} unabhängig von den Anwendungsmöglichkeiten in den Archiven für die Zukunft von NLP-Anwendungen folgendes voraus:

*„We believe applications that rely on natural language interfaces are only going to become more common, replacing much of what is currently done with forms and clicks.“*

Ein Aufgabenbereich der natürlichen Sprachverarbeitung ist die Textklassifikation. Textklassifikation kann sich dabei auf verschiedene sprachliche Ebenen beziehen. Die Auszeichnung der grammatischen Funktion von Wörtern, das sogenannte *Part-of-Speech-*Tagging (POS-Tagging), ist ebenso eine Aufgabe der Textklassifikation wie die Zuweisung eines Themas oder die Auszeichnung der Präsentationsform bei einem Pressetext. Zum Einsatz kommen dabei verschiedene Algorithmen. In der Forschung geht es vor allem um die Optimierung und deren Vergleich, gemessen an der Klassifikationsleistung. Dabei spielen verschiedene Faktoren eine Rolle. Neben dem Klassifikator und den Parametereinstellungen sind dabei die zugrundeliegenden Daten ein Hauptfaktor. Natürliche Sprache ist komplex und die Auftretungserscheinungen variantenreich. Die Verarbeitung für eine Maschine ist daher nicht trivial.

Hier noch besserer Übergang  
Im zurzeit eingesetzten System zur automatischen Verschlagwortung von Pressetexten werden nur inhaltliche Aspekte wie Entitäten, Geographika oder Themen maschinell indexiert. Präsentationsformen zählen zu den sogenannten Beta-Klassen bzw. formalen Klassen, da sie nicht direkt den Inhalt eines Textes beschreiben, sondern dessen Form. Den Pressetexten wird als Standardeinstellung immer die Klasse „Bericht“ vergeben. Andere Formen werden händisch nachgetragen, sofern sie überhaupt nachgetragen werden. Allerdings gehören Recherchen nach Interviews, Chronologien oder Kommentaren in Presseartikeln am multimedialen Recherchedesk[[2]](#footnote-2) der Abteilung Information, Dokumentation und Archive(**IDA**) des Südwestrundfunks (**SWR**) zum Alltag, sodass eine automatisierte Klassifikation der Präsentationsformen von Pressetexten der Recherche Abhilfe verschaffen würde. Um die Recherche zukünftig zu ermöglichen, wurde im Rahmen des Projektes Text in Medas (**TiM**) untersucht, welche Methoden der künstlichen Intelligenz bzw. des maschinellen Lernens geeignet sind, Präsentationsformen automatisch zu bestimmen. Das Projekt TiM hat generell die Realisierung einer gemeinsamen Presseanwendung der Rundfunkanstalten im Media Data Hub (**MDH**) zum Ziel.   
Im Rahmen dieser Arbeit wird ein Proof of concept (**PoC**) erstellt, indem verschiedene in Frage kommende Technologien zur Kategorisierung von Textdaten getestet werden. Als Trainingsmaterial dienen Presseartikel aus dem PAN-Datenbankbestand der Jahre 2013-2021. Die Modellanforderungen sind nicht spezifiziert, da beim Einsatz von KI bzw. Machine Learning in den seltensten Fällen hun­dert­pro­zen­tige Genauigkeit erreicht werden kann. Welche Merkmale pro Gattung für eine automatisierte Erkennung genutzt werden können ist dabei schwierig vorab zu definieren, da eindeutige und trennscharfe Kriterien für einen algorithmischen Ansatz nicht in jedem Fall identifiziert werden können. Auch für Dokumentar\*innen ist die Zuordnung in der Praxis schwierig. Da die Selektierung von geeigneten, im Text vorhandenen Merkmalen für alle Gattungen aufwändig ist und damit nicht garantiert ist, dass die trainierten Klassifikatoren anschließend funktionieren, wird grundsätzlich darauf verzichtet. Eine Fragestellung, die der PoC am Ende der Testphase beantworten soll, ist, inwieweit die „rohen“ Pressetexte ohne die Auswahl von bestimmten textlichen Merkmalen als Grundlage ausreichen, um die Gattung zu bestimmen.   
Teilaspekte im Klassifikationsprozess, die einfach beeinflusst werden können, ist unter anderem der Einfluss verschiedener **Preprocessing-**Schritte, wie das **Stemming** oder die **Lemmatisierung**. Außerdem werden verschiedene Vektorisierungsmethoden geprüft. Dazu zählen ein einfacher **Count-Vectorizer**, sowie eine **binäre** Vektorisierungsmethode (**One-hot**) und die häufig verwendete **TF:IDF‑**Vektorisierung. Bei der Auswahl des Klassifikationsalgorithmus werden drei Modelle aus dem Bereich des überwachten maschinellen Lernens miteinander verglichen, da die Untersuchung von allen Algorithmen, die zur Textklassifikation in Fragen kommen, den Rahmen dieser Arbeit sprengen würde. Diese werden häufig als B**aseline**-Modelle bezeichnet, weil sie entweder sehr simpel sind oder viel Potenzial besitzen, ohne große Rechenaufwände zu benötigen. Zu diesen zählen der **Naive Bayes**, die **Logistische Regression** und die **Support Vektor Maschinen** (SVM). Wegen seiner „naiven“ Annahme wird der Naive Bayes Algorithmus in der Literatur als simpel bezeichnet. Die Logistische Regression und die Support Vektor Maschinen erreichen allgemein gute Ergebnisse. In der Theorie wird auch ein nicht überwachtes Modell vorgestellt, dass aber aus praktischen Gründen nicht mehr angewendet werden konnte.[[3]](#footnote-3)

In Untersuchungen, die sich mit dem Vergleich verschiedener Algorithmen beschäftigen, mangelt es in vielen Fällen an einer tieferen Betrachtung der Funktionsweise. Es werden häufig lediglich die Ergebnisse der verschiedenen Algorithmen präsentiert. Währenddessen gibt es unzählige Anleitungen und Tutorials, die die praktische Anwendung nur bei „Spielzeugdaten“ veranschaulichen. Sie werden oft so bezeichnet, weile sie nur für einen bestimmten Einsatz optimiert sind. Diese Arbeit verbindet die ausführliche Herleitung und Erklärung der verschiedenen Prozesse, zeigt welche Aspekte bei der Klassifikation beachtet werden müssen und arbeitet dabei mit echten Datensätzen. Die genaue Beschreibung ist dabei essenziell für die Erstellung des PoCs, damit nicht nur die Entwickler\*innen die Funktionsweise verstehen, sondern auch Dokumentar\*innen diese nachvollziehen können.  
Die folgende Arbeit strukturiert sich in XY Kapitel. In Kapitel 2 wird die Arbeit in den Projektkontext eingeordnet. Anschließend werden in Kapitel 3 die grundlegenden Begriffe und Techniken der Textklassifikation näher beschrieben und erläutert. Anschließend wird die Datengrundlage in Kapitel 4 vorgestellt, bevor in Kapitel 5 die Methodik und Kapitel 6 die Ergebnisse präsentiert werden. Im Schlussteil in Kapitel 7 werden dann die Best Practices und die Lessons learned diskutiert.

# Projektbeschreibung und Einordnung





## Zielsetzung und projektbezogene Ergebnistypen

## Stakeholderanalyse

Was ist Text in Medas

Was ist MDH?

****

## SWOT-Analyse

Vor und Nachteile von KI

Der menschliche Faktor

![Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung]()

# Grundlagen und Forschungsstand

## Natürliche Sprachverarbeitung

Eines der Alleinstellungsmerkmale der Menschheit ist die hochkomplexe Kommunikation mit Hilfe von natürlicher Sprache. Sie entwickelte sich im Lauf der Zeit und wurde extrem effizient \cite[vgl.][88]{kamathDeepLearningNLP2019}. Dieser Prozess ist dabei nie abgeschlossen. Denn Sprache und Wörter sind mehrdeutig. So könnte mit *Maus* beispielsweise ein Nagetier oder die Maus des Computers gemeint sein. Diese Ambiguität wird meistens durch den Kontext aufgelöst. So ergibt sich auch für \citet[20]{wittgensteinPhilosophicalInvestigations1968} die Bedeutung eines Wortes aus seiner Nutzung in der Sprache. Mit der Veränderung der Wortform geht auch immer eine Änderung der Bedeutung einher. In \citet[96]{jurafskySpeechLanguageProcessing2020} wird dieses Phänomen das \textit{Kontrastprinzip} genannt. So würde $H\_{2}0$ in einem wissenschaftlichen Kontext durchaus angebracht sein, in einer Getränkewerbung eher nicht. Auch wenn sich die Bedeutung von Wörtern aus deren Nutzen ergibt, sorgt der tägliche Gebrauch für eine stetige Bedeutungsveränderung \cite[vgl.][Vorwort xf]{bengfortAppliedTextAnalysis2018}.

Diese stetige Veränderung und die Variation von Sprache machen den maschinellen Umgang mit ihr zu einer Herausforderung \cite[vgl.][88]{kamathDeepLearningNLP2019}. Maschinelle Systeme müssen damit umgehen, dass Texte verschiedene Sprachen beinhalten, verschiedenen Genres angehören oder verschiedene demographische Eigenschaften der Sprecher:innen oder Schreiber:innen reflektieren \cite[vgl.][13]{jurafskySpeechLanguageProcessing2020}. Jedes Wort, jeder Satz oder jede Äußerung trägt in unterschiedlicher Ausprägung die Konnotation der Person mit \cite[vgl.][97]{jurafskySpeechLanguageProcessing2020}. Algorithmen müssen sich deshalb mit der Sprache verändern \cite[vgl.][291]{bengfortAppliedTextAnalysis2018}.\\

Sprache ist hochdimensional und Wörter korrelieren miteinander \cite[vgl.][79]{aggarwalSurveyTextClustering2012}. Wie in Abschnitt \textbf{\ref{section:Text als Zahlen}} näher beschrieben wird, muss Text in eine nummerische Darstellung umgewandelt werden, damit Algorithmen ihn verarbeiten können. Dadurch entstehen komplexe, hoch dimensionale Vektoren, die je nach verwendeter Algorithmusarchitektur schwerer zu verarbeiten sind. Mit Korrelation von Wörtern ist gemeint, dass der Inhalt oder die Aussage eines Dokumentes sich aus dem Gesamtkontext ergibt - also aus der Korrelation aller Wörter eines Dokumentes. Eine Reihe von Algorithmen betrachten einzelne Wörter auch unabhängig ihres Kontextes, da dies die Berechnung vereinfacht und in vielen Fällen auch zu akzeptablen Ergebnissen führen kann (siehe Abschnitt \textbf{\ref{subsection: Naiver Bayes}}).\\

Bei der natürlichen Sprachverarbeitung geht es generell darum, dass Maschinen, \ac{bzw.} Algorithmen, Sprache verstehen und interpretieren können \cite[vgl.][11]{kamathDeepLearningNLP2019}. Deren Bedeutung nimmt stetig zu, denn viele Anwendungen inkorporieren in irgendeiner Form Technologien der natürlichen Sprachverarbeitung. Diese sind mittlerweile schon zu festen Bestandteilen im Alltag geworden, dass sich viele Nutzer:innen an sie gewöhnt haben \cite[vgl.][1]{bengfortAppliedTextAnalysis2018}. Dazu zählt unter anderem die Verwendung von Suchmaschinen, die Übersetzung von Texten in Echtzeit (\ac{z.B.} bei \href{https://www.deepl.com/translator}{\underline{deepl.com/translator}} oder \href{translate.google.de/}{\underline{https://translate.google.de}} oder auch die Autokorrektur auf Smartphones. Letztere besitzen, ähnlich wie Smart-Home-Systeme oder moderne PKW, Schnittstellen, um gesprochene Sprache zu verarbeiten. Zum Beispiel um eine Nachricht einzusprechen und zu versenden, die Temperatur im Haus zu steuern oder die Navigation einzustellen. Diese und andere Endanwendungen bauen dabei wieder auf verschiedenen Technologien der natürlichen Sprachverarbeitung auf, die nicht direkt von Nutzer:innen wahrgenommen werden.\\

Texte, Videos und Bilder werden \ac{z.B.} teilweise automatisiert für die effizientere Suche in Mediatheken oder großen Dokumentensammlungen wie Wikipedia verschlagwortet. Eine andere populäre Technologie ist die automatische Klassifikation von Texten, wie die Klassifikation von Emails als Spam. Dabei entscheidet ein Algorithmus mit Hilfe von bereits als Spam markierten Emails, ob eine neue Nachricht Spam ist.\\

In vielen Fällen gibt es vorher keine bereits markierten Daten. Hier kann das Textclsutering eingesetzt werden. Es sucht im Gegensatz zur Klassifikation latente Muster in Texten auf Basis diverser Ähnlichkeitsmaße. Diese bestimmen ohne menschliches Eingreifen Gemeinsamkeiten in Texten \cite[vgl.][113]{kamathDeepLearningNLP2019}\footnote{Die Liste an Anwendungsgebieten ist lang. Die Beispiele in dieser Arbeit sind deshalb nicht erschöpfend und eine detailierte Ausarbeitung aller Bereiche ist für den Zweck dieser Arbeit nicht dienlich. Eine gute Übersicht findet sich auf \href{https://en.wikipedia.org/wiki/Category:Tasks\_of\_natural\_language\_processing}{Wikipedia: TasksOfNaturalLanguageProcessing}}.\\

\newpage

Für die Verschlagwortung oder Bestimmung von Spam müssen Algorithmen relevante Informationen erkennen und extrahieren, um sie nutzen zu können. Dieser Vorgang wird im Englischen auch als \textit{Information Extraction} bezeichnet \cite[vgl.][325]{jurafskySpeechLanguageProcessing2020}. Dazu wird \ac{z.B.} die sogenannte \textit{Named entity recognition} genutzt. Dabei werden Stellen im Text gesucht, die Entitäten wie Personen, Orte, Produkte oder Organisationen bezeichnen, und die Art der Entität anschließend bestimmt. Wie effektiv diese oder andere Methoden sind, hängt letztlich von der Anwendung, dem Genre, dem Medium und der Sprache ab \cite[vgl.][327]{jurafskySpeechLanguageProcessing2020}.\\

Ob nun Emails in Spam oder nicht Spam oder Dokumente anhand ihrer Schlagwörter in verschiedene Kategorien aufgeteilt werden, ist methodisch ähnlich. So schlussfolgern \citet[109]{kamathDeepLearningNLP2019}, dass die meisten Probleme im Bereich der natürlichen Sprachverarbeitung letztlich eine Klassifikationsaufgabe sind. Für sie bietet \citet[1]{sebastianiMachineLearningAutomated2002a} eine treffende Definition in dem er sie allgemein als "[...] \textit{the activity of labeling natural language texts with thematic categories from a predefined set} [...]" bezeichnet.\\

Während die Klassifikation in den meisten Fällen in größere Anwendungen eingebunden ist, erfolgt sie selbst in mehreren Teilschritten. Ist ein Textkorpus vorhanden, folgen dann im ersten Schritt eine Reihe von Vorverarbeitungsprozessen (im englischen \textit{Preprocessing}) die unter der Normalisierung von Text zusammengefasst werden können \cite[vgl.][2]{jurafskySpeechLanguageProcessing2020}. Ziel dieser Schritte ist die Auswahl linguistischer Merkmale, die den später angewendeten Klassifikationsalgorithmen als Basis dienen, um ihre Entscheidungen über die Zugehörigkeit zu einer Klasse zu treffen.

Hier gibt es eine Vielzahl an verschiedenen Möglichkeiten, von denen einige im nächsten Abschnitt \textbf{\ref{section: Preprocessing}} vorgestellt werden.\\

Nachdem die Textdaten vorverarbeitet wurden, folgt anschließend die Umwandlung in eine Struktur, die die Algorithmen verarbeiten können. Dazu werden die Daten in Zahlen transformiert und als Vektormatrizen repräsentiert. Dieser Vorgang wird in Abschnitt \textbf{\ref{section:Text als Zahlen}} näher betrachtet. Schließlich werden die diversen Klassifikationsalgorithmen mit den transformierten Daten trainiert und ihre Leistung und Genauigkeit evaluiert. Die grundprimitiven Algorithmen zur Klassifikation werden in den Abschnitten \textbf{\ref{section: Maschinelles Lernen zur Textklassifkation}} und \textbf{\ref{section: Evaluation von Modellen und Algorithmen}} beleuchtet und ihre Funktionsweise hergeleitet. Letztlich kann der Klassifikationsschritt wiederholt werden, wenn das Ergebnis nicht zufriedenstellend ist. In diesem Fall könnten die Vorprozessierungschritte angepasst oder die Parameter der Algorithmen justiert werden. Am Ende erfolgt dann eine Re-Evaluation der Ergebnisse.

## Vorprozessierung

### Segmentierung und Tokenisierung

Nicht nur vor der Klassifikation von Texten, sondern vor nahezu jeder Verarbeitung natürlicher Sprache müssen Normalisierungsprozesse erfolgen \cite[vgl.][14]{jurafskySpeechLanguageProcessing2020}. Ziel ist häufig die sogenannte Reduktion der Dimensionen, also die Entfernung von sprachlichen Merkmalen oder Features zur Erhöhung der Genauigkeit und Leistung der Klassifikationsalgorithmen \cite[vgl.][2]{ikonomakisTextClassificationUsing2005a}. Dafür müssen die Texte im ersten Schritt in irgendeiner Form tokenisiert \ac\*{bzw.} segmentiert werden. Dies bedeutet allgemein die Zerlegung des Textes in kleinere Einheiten. Je nach Anwendungsfall, erfolgt diese Verarbeitung auf unterschiedlichen Ebenen. Ein Roman kann in seine Kapitel zerlegt werden und ein Kapitel in Absätze. Absätze können wiederum in Sätze, Sätze in einzelne Wörter und diese in Morpheme aufgeteilt werden. Im Bereich der natürlichen Sprachverarbeitung wird der Begriff der Tokenisierung meistens verwendet, um die Zerlegung in einzelnen Wörter zu bezeichnen. Dabei können zwei Arten unterschieden werden, um über Wörter zu sprechen. Mit \textit{Types} sind alle einzigartigen Wörter eines Korpus gemeint. Diese Wörter bilden das sogenannte Vokabular $|V|$ eines Korpus \cite[vgl.][1]{baayenWordFrequencyDistributions2001}. Im Vergleich dazu meint \textit{Tokens} "jedes einzelne Vorkommen eines Wortes (oder einer anderen Einheit) in einem Text" (vgl. \href{https://de.wiktionary.org/wiki/Token}{\underline{de.wiktionary.org/wiki/Token}}). Während also jeder Type in einem Korpus einzigartig ist, können Tokens beliebig oft auftauchen.\\

Die Relation zwischen Types und Tokens (\ac{TTR}) beschreibt also, wie häufig jedes einzigartige Wort in einem Korpus vorkommt. Dafür wird die Anzahl der Types durch die Anzahl der Tokens geteilt.

\begin{align}\label{Formel: TTR}

TTR = \frac{Count(Types)}{Count(Tokens)}

\end{align}

Bei einem Wert von $1$ würde auf jeden Type genau ein Token kommen. Je näher der Wert an $1$ ist, desto vermeidlich komplexer ist der Text. Die \ac{TTR} sollte allerdings mit Vorsicht betrachtet werden. Mit einem wachsenden Korpus wird der Wert für die \ac{TTR} kleiner. Ein Korpus kann aus beliebig vielen Tokens bestehen. Die Types sind aber für jede natürliche Sprache begrenzt. Dadurch wird das Verhältnis von Types und Tokens schief. Die Häufigkeitsverteilung von Wörtern in einem Korpus ist dabei tendenziell immer sehr ungleichmäßig \cite[vgl.][425]{leopoldTextCategorizationSupport2002}.\\

Diese Ungleichmäßigkeit wird durch das \textit{Zipfsche Gesetz} beschrieben. In seiner Lebenszeit ordnete Zipf Wörter seiner Texte absteigend nach ihrer Häufigkeit \cite[vgl.][13]{baayenWordFrequencyDistributions2001}. Daraus ergab sich immer eine ähnliche Häufigkeitsverteilung. Als Faustregel beschreiben \citet[425]{leopoldTextCategorizationSupport2002}, dass ungefähr $50$ Prozent aller Types nur einmal vorkommen. Diese ungleiche Verteilung verläuft dabei exponentiell \cite[vgl.][93]{kamathDeepLearningNLP2019}. Abbildung \textbf{\ref{Figure: Zipfsche Verteilung der ZDF-Daten}} zeigt die Verteilung der $1000$ häufigsten Wörter im ZDF-Korpus. Die Zahlen auf der X-Achse beschreiben dabei die Indizes der Wörter, welche absteigend nach der Häufigkeit sortiert sind. Das häufigste Wort hat also den Index $0$, das zweihäufigste den Index $1$ usw. Die exponentielle Verteilung ist dabei gut zu erkennen. Das Zipfsche Gesetz trifft für diesen Datensatz zu. Von ungefähr $93.000$ einzigartigen Types kommen ca. $42.000$ lediglich einmal vor. Das entspricht ungefähr $39$\%\footnote{Detaillierte Analysen folgen in Abschnitt \textbf{\ref{section: Datenanalyse}}.}.

\begin{figure}[ht]%option kann noch anders gesetzt werden.

\centering

\includegraphics[width=0.85\textwidth]{./Zipf\_Verteilung.png}

\caption[Zipfsche Verteilung der ZDF-Daten]{Zipfsche Verteilung der ZDF-Daten}

\label{Figure: Zipfsche Verteilung der ZDF-Daten}

\end{figure}

Bevor Häufigkeitsverteilungen von Types und Tokens eines Korpus erstellt werden können, müssen die Grenzen zwischen einzelnen Tokens bestimmt werden. Dabei ergibt sich eine Reihe von Schwierigkeiten. Generell könnte angenommen werden, dass Tokens an Leerzeichen getrennt werden \cite[vgl.][12]{jurafskySpeechLanguageProcessing2020}. Damit werden die meisten Token korrekt erkannt, allerdings sollen folgende Beispielsätze einige Probleme bei der eindeutigen Bestimmung illustrieren:

% Blockzitat-Format

\begin{quote}\label{Quote: Beispiel Tokenisierung}

Der Jahresetat der Johannes Gutenberg-Universität Mainz belief sich im Jahr $2018$ bei $504$ Mio. EUR. Davon kamen \ac{ca.} $157.000.000$€ aus Drittmitteln. Mit \ac\*{ca.} $32$ Tausend Studierenden ist die JGU eine von Deutschlands größten und vielfältigsten Unis. Mit der Johann Wolfgang Goethe-Universität Frankfurt am Main und der Technischen Universität Darmstadt bilden die drei gemeinsam die Rhein-Main-Universitäten. Von $2011-2013$ gehörte die JGU zu den $20$ Universitäten mit der höchsten Bewilligungsstufe im Bereich Naturwissenschaften.

\end{quote}

(Zahlen und Fakten dienen nur der Veranschaulichung und sind nicht geprüft.)

Je nachdem wie die Tokengrenzen definiert werden, entstehen bei den obigen Sätzen verschiedene Relationen von Types und Tokens. \\

Es wird deutlich, dass es in vielen Fällen Sinn ergibt, Schreibweisen zu vereinheitlichen. Zum Beispiel ergibt die Vereinheitlichung bei Zahlen und Währungsangaben (\textit{$504$ Mio. EUR, $157.000.000$€ oder $32$ Tausend}) Sinn, sowie bei possessiv Formen von Nomen wie \textit{Deutschlands} im Vergleich zu \textit{Deutschland´s} \cite[vgl.][16f]{indurkhyaHandbookNaturalLanguage2010}. Darüber hinaus ist es sinnvoll Abkürzungen wie \textit{JGU} und die ausgeschriebenen Varianten (\textit{Johannes Gutenberg-Universität Mainz}) zu standardisieren, da sie dieselbe Entität beschreiben.\\

Des weiteren entstehen Probleme, wenn Tokens am Leerzeichen getrennt werden, da es viele Begriffe gibt, die sich aus mehreren Wörtern zusammensetzen (\textit{Frankfurt am Main, Technische Universität Darmstadt, $32$ Tausend Studierenden}). In moderneren Textformen wie Tweets ist häufig auch die Prozessierung von \textit{Hashtags \#} oder Emoticons wie \textit{xD)} nötig \cite[vgl.][2]{jurafskySpeechLanguageProcessing2020}. Diese bestehen teilweise aus Satzzeichen, was die Disambiguierung erschwert.\\

Ein anderer Faktor ist die Groß- und Kleinschreibung. Die Schreibweise kann vereinheitlicht werden. Das kann Auswirkungen darauf haben, wie gut Nomen von Verben und Adjektiven unterschieden werden können. Dafür werden typischerweise alle Wörter in Kleinschreibweise formatiert. Für manche Gebiete wie die Erkennung natürlicher Sprache macht die Schreibweise der Wörter keine Unterschiede \cite[vgl.][12]{jurafskySpeechLanguageProcessing2020}. Für Sentimentanalysen und anderen Textklassifikationsaufgaben, sowie für die Extraktion von Informationen oder auch die maschinelle Übersetzung kann die Groß- und Kleinschreibung hilfreich sein. Die Beibehaltung der Unterschiede zwischen zwei Wörtern durch ihre Schreibweise kann den Vorteil der Generalisierung überwiegen \cite[vgl.][21]{jurafskySpeechLanguageProcessing2020}.\\

Die Einordnung von Satzzeichen ist generell herausfordernd. Während Frage- und Ausrufezeichen laut \citet[][22]{jurafskySpeechLanguageProcessing2020} relativ eindeutige Funktionen haben, sind Satzzeichen wie Punkte, Kommata, Anführungszeichen, Apostrophe oder Bindestriche ambig \cite[vgl.][16]{indurkhyaHandbookNaturalLanguage2010}. Der Punkt kann \ac{z.B.} nicht nur ein Satzende markieren, sondern auch als Dezimaltrennzeichen (157.000.000) fungieren oder am Ende von Abkürzungen stehen (\textit{Mio., \ac\*{ca.}}). In diesen Fällen sollten Punkte entfernt werden \cite[vgl.][17]{indurkhyaHandbookNaturalLanguage2010}\footnote{Bei der Identifizierungvon Satzenden kann der Kontext eine entscheidene Rolle spielen. \citet[25]{indurkhyaHandbookNaturalLanguage2010} nennen diverse Methoden zur Disambiguierung von Sätzen. Die Beschreibung dieser führt hier allerdings zu weit.}. Für die Erkennung von Abkürzungen kann ein, in die Anwendung eingebundenes, Wörterbuch mit Abkürzungen helfen \cite[vgl.][22]{jurafskySpeechLanguageProcessing2020}. Bindestriche erfüllen ebenfalls verschiedene Funktionen. Während der Bindestrich bei \textit{$2011-2013$} einen Zeitraum markiert, verbindet er in \textit{Rhein-Main-Universitäten} mehrere Begriffe miteinander. Bei letzterem sollte die Verbindung durch die Bindestriche nicht aufgespaltet werden, aber gleichzeitig andere Schreibweisen wie \textit{Rhein Main Universitäten} vereinheitlicht werden. Wie mit Satzzeichen umgegangen wird und ob sie als einzelne Tokens gezählt oder ignoriert werden, hängt von der Anwendung ab \cite[vgl.][11]{jurafskySpeechLanguageProcessing2020}. \\

Eine andere Herausforderung ist die Verarbeitung von Komposita, da sie in der Deutschen Sprache recht komplex werden können \cite[vgl.][18]{indurkhyaHandbookNaturalLanguage2010}. Bei manchen Systemen ist es wünschenswert, die einzelnen Wörter aus dem Kompositum zu extrahieren (\ac{z.B.} für Suchmaschinen), um das Finden des Kompositums durch die einzelnen Bestandteile zu ermöglichen. Die Autoren des Snowballstemmers, auf den in Abschnitt \textbf{\ref{subsection: Stammformreduktion und Lemmatisierung}} näher eingegangen wird, sind allerdings der Meinung, dass das Problem der Trennung von Komposita etwas überbewertet sei (\href{https://snowballstem.org/algorithms/germanic.html}{vgl. \underline{snowballstem.org/algorithms/germanic.html}}).\\

Wie gezeigt werden kann, ist die Tokenisierung bzw. Segmentierung von Text mitunter schwierig. Es hängt immer vom speziellen Anwendungsfall ab, welche Tokenisierungs- und Segmentierungsschritte erfolgen müssen\cite[vgl.][16]{indurkhyaHandbookNaturalLanguage2010}.\\

Werden in den obigen Beispielsätzen alle Ambiguitäten aufgelöst, also Wörter normalisiert und Satzzeichen vereinheitlicht, bestehen die Sätze aus insgesamt $46$ Types und $65$ Tokens, was einer Relation von \ac\*{ca.} $0,71$ entspricht, einem relativ hohen Wert.\\

Algorithmen zur Tokenisierung bzw. Segmentierung beruhen dabei oft auf sogenannten \textit{Regulären Ausdrücken} \cite[vgl.][16]{jurafskySpeechLanguageProcessing2020}. Sie sind algebraische Notationen, um Gruppen von Strings (Zeichenketten) zu charakterisieren und besonders nützlich, wenn Muster in Texten bestehen \cite[vgl.][3]{jurafskySpeechLanguageProcessing2020}. Für diese Aufgabe werden mit Regulären Ausdrücken Muster definiert, nach denen gesucht wird. Bei einem Treffer könnte das gefundene Muster \ac\*{z.B.} dann durch eine andere Zeichenkette ersetzt oder ganz entfernt werden. Reguläre Ausdrücken werden von den meisten gängigen Texteditoren unterstützt. Ein Beispiel soll die Funktionsweise kurz illustrieren. Mit eckigen Klammern [] wird eine Zeichenauswahl definiert und mehrere Suchparameter werden direkt hintereinandergeschrieben. [\textit{A-Za-z$0-9$}] in diesem Beispiel werden alle lateinischen Buchstaben in Groß- und Kleinschrift, sowie alle Ziffern im definierten Suchbereich gefunden.

\subsection{Stammformreduktion und Lemmatisierung}\label{subsection: Stammformreduktion und Lemmatisierung}

Die aus der Tokenisierung entstanden Tokens könnten nun dem Algorithmus zur Klassifikation übergeben werden. Es gibt allerdings diverse Gründe, die für weitere Schritte bei der Vorprozessierung der Daten sprechen. Angenommen ein Text mit dem der Algorithmus trainiert wird, enthält die Wörter \textit{Lauf} und \textit{Läuferin}. Ein Text aus den Daten, mit denen der Algorithmus getestet wird, enthält hingegen nur das Wort \textit{laufen}. Es wird außerdem davon ausgegangen, dass beide Texte ein ähnliches Thema haben. Menschen erkennen diesen Zusammenhang leicht. Für den Algorithmus ist dies ohne weiteres nicht möglich, weil das Wort aus den Testdaten nicht in den Trainingsdaten vorkommt\footnote{Die Begriffe Trainings- und Testdaten werden in Abschnitt \textbf{\ref{subsection: Machinelles Lernen}} genauer erklärt.}. Deswegen kann es in vielen Anwendungsbereichen sinnvoll sein, nach der Tokenisierung bzw. Segmentierung von Textdaten weitere Schritte zur Normalisierung der Daten vorzunehmen.\\

Eine Reduzierung auf den Stamm hilft bei dem oben genannten Beispiel. Diese Aufgabe übernehmen spezielle Algorithmen, die im englischen \textit{Stemmer} genannt werden\footnote{Im weiteren Verlauf dieser Arbeit wird der englische Begriff verwendet}. Ein Stemmer trennt die Affixe der Wörter ab, sodass nur noch der Wortstamm übrig bleibt \cite[vgl.][72]{bengfortAppliedTextAnalysis2018}. Der bereits erwähnte Snowballstemmer\footnote{\href{https://snowballstem.org/}{\underline{https://snowballstem.org}}} ist ein Paket aus anpassbaren Algorithmen zur Stammformreduktion für nicht-englische Sprachen wie Deutsch. Die Wörter \textit{laufen} und \textit{Lauf} werden von diesem Stemmer beide auf \textit{lauf} reduziert\footnote{Dass nach der Reduzierung auf den Stamm ein Wort übrig bleibt, ist häufig nicht der Fall. Der Snoballstemmer reduziert \ac{z.B.} \textit{Studierende} auf \textit{studier}.}. Die auf den Stamm reduzierten Dokumente kann der Klassifikator nun erkennen und dasselbe Thema bestimmen. Hierbei macht es dann auch keinen Unterschied, dass für das Wort \textit{Läuferin} der Stemmer beispielsweise den Stamm \textit{Lauferin} erkennt, da das Klassifikationsmodell den Zusammenhang durch andere Wörter herstellen kann.\\

Ein Stemmer kommt bei stark flektierenden Sprachen wie der Deutschen an seine Grenzen. Oft ändern sich in diesen Sprachen Stämme. Im Deutschen passiert dies oft bei der Pluralbildung wie in \textit{Haus} und \textit{H\textbf{äu}ser}. Für solche Sprachen bietet sich deshalb die Lemmatisierung von Wörtern an.\\

\citet[3]{jurafskySpeechLanguageProcessing2020} bezeichnen sie bei komplexen Sprachen sogar als essenziell. Hierbei werden Wörter auf ihre Grundform, das \textit{Lemma}, zurückgeführt \cite[vgl.][3]{jurafskySpeechLanguageProcessing2020}. Beispielsweise lassen sich die Wörter \textit{gewesen}, \textit{war} und \textit{seiest} auf die Grundform \textit{sein} zurückführen. Zu beachten ist, dass Wörter zwar derselben Wortfamilie angehören können und auch ähnliche Bedeutungen haben, sich die Grundformen aber unterscheiden können. \textit{Läuferin} und \textit{laufend} teilen sich den Ursprung im Verb \textit{laufen}, haben aber unterschiedliche Grundformen (\textit{Läufer} und \textit{laufen}). Algorithmen zur Lemmatisierung von Wörtern greifen häufig auf Wörterbücher zurück, um die Lemmata zu erkennen \cite[vgl.][72]{bengfortAppliedTextAnalysis2018}. Die Verwendung von Wörterbüchern macht die Lemmatisierung anfällig für Rechtschreibfehler, da so falsch geschriebene Wörter nicht erkannt werden \cite[vgl.][92]{kamathDeepLearningNLP2019}. Außerdem sind diese Algorithmen relativ komplex und zeitaufwendig, weshalb oft darauf verzichtet wird und lediglich ein Stemmer zum Einsatz kommt \cite[vgl.][21]{jurafskySpeechLanguageProcessing2020}. Dies erfolgt dann auf Kosten der Genauigkeit und Generalisierungsfähigkeit für spätere Sprachmodelle.

### Stammformreduktion und Lemmatisierung

### Stoppwortentfernung

### Part-of-speech Tagging

## Text als Zahlen

### Sprachmodelle

### Vektorisierung und Gewichtung

Während die Entscheidung darüber, ob die Textdaten unbehandelt bleiben oder vorprozessiert werden, diskutabel ist, ist die Vektorisierung der Daten ein unerlässlicher Schritt für die Textklassifikation \cite[vgl.][55]{bengfortAppliedTextAnalysis2018}. Die Autor:innen weisen weiter daraufhin, dass dafür ein Umdenken darüber stattfinden muss, wie Sprache verstanden wird. Formal muss Sprache also weg von einer Sequenz von Wörtern hin zu Punkten in hoch-dimensionalen Räumen verstanden werden. Diese Punkte können, so beschreiben Sie weiter, nahe beieinander, weit voneinander entfernt, eng gebündelt oder gleichmäßig verteilt sein. Daraus ergibt sich, dass Dokumente mit ähnlicher Bedeutung im Vektorraum auch näher zusammenstehen und nicht verwandte Dokumente weiter voneinander entfernt sind.\\

Mit einem Vektorraum wird eine Kollektion von einzelnen Vektoren beschrieben. Jeder Vektor steht dabei für ein einzelnes Dokument \cite[vgl.][100]{jurafskySpeechLanguageProcessing2020}.

Bei der Modellierung des Vektorraums gibt es nun verschiedene Optionen. Eine Variante ist die Umwandlung in das sogenannte \textit{\ac{BoW}}-Modell \cite[vgl.][55]{bengfortAppliedTextAnalysis2018}. Dazu werden alle Dokumente als Vektor mit der Länge des Vokabulars $|V|$ des Gesamtkorpus repräsentiert. Wie in Abschnitt \textbf{\ref{subsection: Segmentierung und Tokenisierung}} beschrieben, wird mit dem Vokabular eines Korpus die Menge aller einzigartigen Wörter (Types) bezeichnet.

\newpage

Ein Dokument im Vektorraum enthält also für jedes Wort im Vokabular $|V|$ eine Dimension, auch wenn das Dokument nicht alle Wörter aus $|V|$ enthält. Der Vektorraum $X$ eines Korpus mit $n$ Dokumenten könnte wie in Abbildung \textbf{\ref{figure: Vektormatrix}} aussehen. Jede Zeile steht dabei für ein Dokument im Korpus, dass aus Featuren $x\_{i}$ mit der Länge des Vokabulars $|V| = n$ besteht.

\begin{figure}[h]

\begin{align}\label{figure: Vektormatrix}

\mathbf{X} = \begin{bmatrix}

x\_{11} & x\_{12} & ... & x\_{1n} \\

x\_{21} & x\_{22} & ... & x\_{2n} \\

\vdots & \vdots & \vdots & \vdots \\

x\_{n1} & x\_{n2} & ... & x\_{nn}

\end{bmatrix}

\end{align}

\end{figure}

Anstelle der Wörter steht in jeder Dimension, wenn nicht weiter spezifiziert, dann die Häufigkeit ihres Auftretens (\textit{Termfrequenz}) im Dokument. An vielen Stellen im Vektor stehen lediglich Nullen für die Wörter im Vokabular, die gleichzeitig aber nicht im Dokument auftauchen. Die Vektoren, die daraus entstehen, werden deshalb als spärlich bezeichnet. Außerdem geht in dieser Repräsentation die Wortfolge und somit der Kontext verloren. Dadurch ist das Modell einerseits zwar relativ simpel aufgebaut und einfach nachzuvollziehen, andererseits entstehen durch den Verlust der Wortfolge eine Reihe von Problemen. So können zwei Sätze bei Verwendung gleicher Wörter die gleiche Repräsentation, aber unterschiedliche Bedeutungen haben \cite[vgl.][1]{leDistributedRepresentationsSentences2014}.

\begin{example}

Die Katze frisst die Maus.

\end{example}

\begin{example}

Die Maus frisst die Katze.

\end{example}

Diese Feinheiten oder andere sprachliche Besonderheiten wie Sarkasmus werden so nicht von den Modellen erfasst [vgl.][5]\cite{leDistributedRepresentationsSentences2014}. Darüber hinaus stellen \cite{sriramShortTextClassification2010} in ihrer Arbeit zur Klassifikation von Kurznachrichten bei Twitter fest, dass zu kurze Texte nicht ausreichen, um effektive \ac\*{BoW}-Modelle zu bauen. Trotz der offen liegenden Nachteile sind \ac\*{BoW}-Modelle als Basismodell nützlich \cite[vgl.][125]{bengfortAppliedTextAnalysis2018}.\\

Ein andere Art der Repräsentation sind sogenannte \textit{n-gram}-Modelle. Anders als bei \ac\*{BoW}-Modellen wird bei diesen ein kleines Kontextfenster für jedes Wort betrachtet und somit zumindest die unmittelbare Wortfolge berücksichtigt. Das $n$ steht für die Anzahl der Wörter im Kontextfenster. So ergeben sich bei $n = 2$ im Satz: "\textit{Das Wetter ist heute schön.}" fünf sogenannte \textit{bi-Gramme}. Der Punkt als Satzzeichen wird mitgezählt.

\begin{example}

$[$Das Wetter$]$

$[$Wetter ist$]$

$[$ist heute$]$

$[$heute schön$]$

$[$schön .$]$

\end{example}

Obwohl durch n-Gram-Modelle ein kurzer Kontext erfasst wird, leiden sie ebenso wie \ac\*{BoW}-Modelle an hoher Dimensionalität und damit spärlichen Vektoren \cite[vgl.][1]{leDistributedRepresentationsSentences2014}. \citet[125]{bengfortAppliedTextAnalysis2018} merken dazu an, dass bei n-Gramm-Modellen viele, nicht aussagekräftige Kandidaten entstehen und dadurch der Rechenaufwand die Verwendung dieser Modelle nicht rechtfertigt. Speziell bi-gram-Modelle ($n=2$) für die Textklassifikation seien nicht sehr populär, da Schlüsselwörter im Text oft alleine schon indikativ genug für ein Thema seien \cite[vgl.][4]{leDistributedRepresentationsSentences2014}. Deshalb werden auch für die spätere Analyse der ZDF-Daten lediglich \ac\*{BoW}-Modelle berücksichtigt.\\

Wie bereits erwähnt, wird ein \ac\*{BoW}-Modell initial mit den Termfrequenzen der Wörter gebildet. Für viele Anwendungen genügt diese Repräsentation von Dokumenten, obwohl dadurch ein signifikanter Verlust an Informationen mit einhergeht \cite[vgl.][95]{kamathDeepLearningNLP2019}. Neben der Termfrequenz werden in den nächsten Abschnitten sinnvolle Alternativen für die Vektorisierung bei der Textklassifikation besprochen.

\subsection{Vektorisierung und Gewichtung}\label{subsection: Vektorisierung und Gewichtung}

\subsubsection{Termfrequenz}

Grundsätzlich wird mit der Vektorisierung der Textdaten direkt eine Gewichtung vorgenommen. Bei der bereits kurz eingeführten Termfrequenz erhalten Wörter, die besonders häufig auftauchen, höhere Gewichte als seltene Wörter. Das spiegelt auch die Zipfsche Verteilung natürlicher Sprache wider \cite[vgl.][59]{bengfortAppliedTextAnalysis2018}. Diese Tatsache hat ihre Vor- und Nachteile. Zur Wiederholung: Bei der Termfrequenz steht in jeder Position des Vektors die Häufigkeit eines Wortes ($t\_{i}$) im Dokument ($d$).

\begin{align}\label{Formel: Termfrequenz}

tf\_{t\_{i}, d} = count(t\_{i}, d)

\end{align}

Der Vektor hat insgesamt eine Länge von $|V|$. Ein Vektor enthält für jedes Wort, dass nicht im Dokument, aber im Vokabular ist, eine Termfrequenz von $0$.\\

Es ist generell sinnvoll, die Termfrequenz noch zu normalisieren. Zum Beispiel durch die Dokumentenlänge \cite[vgl.][57]{bengfortAppliedTextAnalysis2018}. Dies ist nötig, da Dokumente eines Korpus unterschiedliche Längen haben und somit Wörter häufiger in langen Dokumenten auftauchen können, als in Kurzen. Dadurch kann dann die Annahme entstehen, dass ein häufiges Wort in einem langen Dokument wichtiger sei, als in einem kurzenm Dokument. Die einfache Termfrequenz ist deshalb verzerrt \cite[vgl.][105]{jurafskySpeechLanguageProcessing2020}.

\newpage

Für die Normalisierung kann die Termfrequenz eines Wortes $t\_{i}$ durch die Anzahl von allen Termen im Dokument $count(t\_{all}, d)$ geteilt werden.

\begin{align}

tf\_{normalized} = \frac{count(t\_{i}, d)}{count(t\_{all}, d)}

\end{align}

Alternativ dazu kann die Termfrequenz auch logarithmiert werden. Dadurch rücken die Werte näher zusammen. Das führt dazu, dass Ausreißer weniger Gewicht erhalten \cite[vgl.][105f]{jurafskySpeechLanguageProcessing2020}.

\subsubsection{One-Hot Enkodierung}

Die zweite Möglichkeit Textdokumente in Vektoren darzustellen, ist durch die \textit{One-Hot} Enkodierung. Statt der Häufigkeit jedes Wortes, steht hierbei in jedem Vektor lediglich der Wert $1$, wenn ein Wort im Dokument vorkommt oder der Wert $0$, wenn es nicht vorkommt. Mehrfachnennungen spielen also keine Rolle, wodurch das Ungleichgewicht durch die Termfrequenz entfällt \cite[vgl.][60]{bengfortAppliedTextAnalysis2018}. \cite{bengfortAppliedTextAnalysis2018} führen weiter an, dass dieser Ansatz besonders gut für sehr kleine Korpora funktioniert, die nicht viele wiederholende Wörter enthalten. Nachteilig ist zum einen, dass ähnlich wie bei der Termfrequenz, der Kontext keine Rolle mehr spielt. Zum anderen sind alle daraus entstehenden Vektoren gleich weit voneinander entfernt. Also nicht verwandte Wörter oder Begriffe wie \textit{Schalke $04$} und \textit{Meistertitel} haben im Vektorraum dieselbe Entfernung wie verwandte Wörter (z.B. \textit{laufen} und \textit{Läufer}). In dieser Vektordarstellung sind deshalb Normalisierungsmethoden wie die Stammformreduktion und Lemmatisierung besonders wichtig, um verwandte Wörter so zu vereinheitlichen \cite[vgl.][61f]{bengfortAppliedTextAnalysis2018}.

## Textklassifkation

### Machine Learning zur Textklassifikation

Allgemein dient die Klassifikation jeglicher Art der Ordnung und besseren Strukturierung. Die stetig wachsende Menge an neuen Daten und deren Verwaltung erfordert deshalb deren automatisierte Klassifikation. Die Klassifikation ist eine Methode des überwachten maschinellen Lernens, weil Algorithmen Daten eine vordefinierte Klasse zuweisen\footnote{Im weiteren Verlauf werden solche Algorithmen allgemein als \textit{Klassifikatoren} bezeichnet.}. Die Entscheidung über die Zugehörigkeit der Daten zu einer Klasse lernt der Klassifikator aus bereits gelabelten Daten. Dabei können einerseits konkrete Klassen bestimmt werden. Andererseits können auch nur Wahrscheinlichkeiten über die Zugehörigkeit zugewiesen werden. Ein Klassifikator der nur Wahrscheinlichkeiten zuweist, wird auch als probabilistisch bezeichnet \cite[vgl.][57]{jurafskySpeechLanguageProcessing2020}. Außerdem kann die Zuweisung auch erst ab einer gewissen Wahrscheinlichkeit konkret vorgenommen werden. Dazu wird vorher ein Schwellenwert definiert, ab dem Daten zu einer Klasse geordnet werden. Diese Form der Klassifikation kann dann beispielsweise von einer Person genutzt werden, um eine finale Einordnung vorzunehmen \cite[vgl.][4]{sebastianiMachineLearningAutomated2002a}.\\

Grundsätzlich sind Klassifikatoren in der Lage, jegliche Art von Daten zu verarbeiten. Unterschiede gibt es nur in der Art der Vorprozessierung und Vektorisierung. Die für Textdaten spezifischen Schritte wie die Normalisierung, Stoppwortentfernung oder dem \ac\*{POS}-Tagging der Textdaten wurden bereits ausführlich besprochen. Genauso vielseitig wie Textdaten sind auch die Einsatzgebiete von Textklassifikatoren. \citet[56f]{jurafskySpeechLanguageProcessing2020} geben hier eine gute Übersicht. Darunter zählt die allgemeine Kategorisierung von Texten oder Büchern beispielsweise nach Themen, Genres oder Autor:innen. Die Erkennung letzterer bildet dabei wiederum ein eigenständiges Einsatzgebiet namens \textit{authorship attribution}. Besonders interessant ist diese im Bereich der digitalen Geisteswissenschaften. Hier könnte ein Klassifikator auf einen Korpus aus Texten einer Autorin trainiert werden, um herauszufinden, ob ein Text von ihr stammen könnte.\\

Ebenso interessant für die digitalen Geisteswissenschaften, aber vor allem für die Wirtschaft, ist die Sentimentanalyse. Bei dieser wird die Stimmung eines Textes bestimmt. Eine Rezension zu einem Produkt kann \ac\*{z.B.} positiv, negativ oder auch neutral formuliert sein. Mit diesen Informationen könnten Unternehmen einschätzen, wie gut ihr Produkt bei den Kunden ankommt oder ein Chatbot kann auf die Emotionen der Chatpartner reagieren.\\

Darüber hinaus ist die bereits vorgestellte Markierung der Wortart in Abschnitt \textbf{\ref{subsection: Part-of-speech Taging}} im Grunde genommen auch eine Klassifikationsaufgabe.

\citet{jurafskySpeechLanguageProcessing2020} nennen außerdem noch die Möglichkeit für die Forensik Täter:innen, anhand von klassifizierten Texten, zu überführen\footnote{Siehe dazu auch die Arbeit von \citet{decherchiTextClusteringDigital2009}, in der das Clustering von Texten für die digitale Forensik diskutiert wird.}.\\

Die Grundlage einer jeden Klassifikation sind die gelabelten Daten. Diese werden oft in Handarbeit und unter hohem Zeitaufwand erstellt, bis genug Daten für das Training des Klassifikationsmodells vorhanden sind \cite[vgl.][104]{nigamTextClassificationLabeled2000}. Ändern sich die Kategorien im Laufe der Zeit, müssen neue Daten gesammelt werden, um sie modellieren zu können \cite[vgl.][7]{aurangzebReviewMachineLearning2010}. \citet[113]{kirkThoughtfulMachineLearning2017} hält das händische Labeln von Texten am sinnvolsten. Der Autor betont allerdings, dass sich Menschen nicht immer über den Inhalt oder Labels einig sind. Das führt dazu, dass immer eine gewisse Grundvarianz beim Labeln durch Menschen gegeben ist. Diese Tatsache könnte auch generell für die Klassifikation verallgemeinert werden. Dazu liegt dieser Arbeit aber keine Quelle vor. Das manuelle Labeln bedeutet, wie bereits erwähnt, einen hohen Zeit- und damit einhergehend auch ein hoher Kostenaufwand. Um beide Faktoren zu minimieren, wird in der Praxis häufig nur ein Teil der Daten eines Korpus händisch gelabelt. Jeder Text wird dabei dann im Vieraugen-Prinzip von mehr als einer Person markiert, um verschiedene Antworten zu mitteln \cite[vgl.][113]{kirkThoughtfulMachineLearning2017}. Mit dem gelabelten Teil des Korpus' können die restlichen Daten automatisiert gelabelt und gegebenenfalls händisch korrigiert werden.\\

Bei den oben beschriebenen Anwendungsgebieten muss noch ein wichtiger Punkt unterschieden werden. Spam in Emails zu erkennen, ist eine binäre \textit{Ja-Nein} Entscheidung. Eine Email ist entweder Spam oder sie ist es nicht. Bei der Analyse des Sentiments oder der Bestimmung des \ac\*{POS}-Tags muss hingegen eine Entscheidung aus mehr als nur zwei Kategorien getroffen werden. Des weiteren kann ein Text mehrere Themen beinhalten und müsste somit auch mehr als einer Kategorie zugeordnet werden. In diesen Fällen wird in der Literatur von einer \textit{multilabel} Klassifikation gesprochen. Hierbei gibt es wieder unterschiedliche Varianten, um dieses Problem zu lösen. Zum einen mit einer \textit{one vs. all} und zum anderen einer \textit{multinomialen} Klassifikation \cite[vgl.][68]{jurafskySpeechLanguageProcessing2020}. In ersterer erhält jedes Label einen eigenen Klassifikator. Dieser trifft dann nur noch die binäre Entscheidung darüber, ob die Daten zur Klasse gehören oder nicht.\\

Im multinomialen Fall ist ein Klassifikator für die Bestimmung aller Labels verantwortlich. Die binäre bzw. one vs. All Klassifizierung ist für alle Arten der Klassifizierung geeignet, da nicht binäre Entscheidungen einfach in binäre ummodelliert werden können \cite[vgl.][3]{sebastianiMachineLearningAutomated2002a}.\\

Das Ergebnis aller Algorithmen ist immer die Zuweisung einer Klasse. Wie sie ihre Entscheidungen treffen, unterscheidet sich je nach Architektur der Algorithmen. \citet[75]{jurafskySpeechLanguageProcessing2020} differenzieren grundsätzlich zwischen \textit{generativen} und \textit{diskriminierenden} bzw. \textit{unterscheidenden} Klassifikatoren. Diesen Unterschied illustrieren die Autoren anhand folgenden Beispiels: Angenommen es sollen Bilder von Katzen und Hunden klassifiziert werden. Ein generativer Klassifikator würde versuchen, die Merkmale der Tiere zu lernen. Dabei handelt es sich um die Features, die sie auszeichnen. Erhält er nun ein zu klassifizierendes Bild, entscheidet der Klassifikator, welches Tier anhand aller gelernten Features am wahrscheinlichsten abgebildet ist. Im Gegensatz dazu würde ein diskriminierender Klassifikator nur versuchen, die Features zu lernen, die Katzen von Hunden unterscheiden. Ist beispielsweise das Tragen eines Halsbandes ein Merkmal, dass Hunde von Katzen unterscheidet, würde die Entscheidung lediglich anhand dieses Features getroffen werden. Ein Beispiel für einen generativen Klassifikator ist der \textit{Naive Bayes} (siehe \textbf{\ref{subsection: Naiver Bayes}}). Beispiele für Diskriminierende sind die \textit{Logistische Regression} (Abschnitt \textbf{\ref{subsection: Logistische Regression}}) und \textit{\acsp{SVM}} (Abschnitt \textbf{\ref{subsection: Support Vector Maschinen}}). Diese werden in den folgenden Abschnitten vorgestellt und später angewendet (Kapitel \textbf{\ref{chapter: Methodik}}). Neben der Verwendung eines einzelnen Klassifikators gibt es Ansätze, die verschiedene Klassifikatoren miteinander kombinieren und solide Ergebnisse erzielen \cite[vgl.][]{wangBaselinesBigramsSimple2012}.\\

Neben diesen sind eine Reihe anderer Algorithmen des maschinellen Lernens auch für die Klassifikation von Daten geeignet. Darunter sind \textit{Maximum Entropy Modelle} (\cite{Nigam99usingmaximum}), \textit{Entscheidungsbäume} (\cite{patelStudyAnalysisDecision2018}) oder Modelle, die auf \textit{K-nächste-Nachbarn} beruhen (\cite{guoUsingKNNModel2006}). Ein anderer Ansatz ist die Verwendung von Ontologien (siehe \cite{allahyariOntologyBasedTextClassification2014}).

\newpage

Letztere erfordern allerdings eine detaillierte Modellierung der Domäne und skalieren schlecht mit großen Datenmengen \cite[vgl.][15]{bengfortAppliedTextAnalysis2018}.\\

Der Fokus der Forschung im letzten Jahrzehnt liegt aber vor allem auf dem Bereich der \textit{neuronalen Netzwerke}. Die steigende Computerleistung und die Entwicklung neuer Sprachmodelle (\cite{mikolovEfficientEstimationWord2013a}, \cite{petersDeepContextualizedWord2018} oder \cite{devlinBERTPretrainingDeep2019}) ermöglichten große Fortschritte in diesem Bereich. Da letztgenannte Methoden mittlerweile als \textit{State-of-the-Art} bezeichnet werden, wird ihnen auch ein Abschnitt in dieser Arbeit gewidmet (\textbf{\ref{subsection: DeepLearning}}). Der Aufwand diese Modelle anzuwenden, würde den Rahmen dieser Arbeit allerdings sprengen. Außerdem bieten die klassischen Methoden oft Ergebnisse, die an die Leistung von neuronalen Netzwerken anknüpfen können \cite[vgl.][1]{joulinBagTricksEfficient2016}. Darüber hinaus gibt es berechtigte Kritik am Trend, alle Probleme der natürlichen Sprachverarbeitung mit neuronalen Netzwerken lösen zu wollen \cite[vgl.][5]{joulinBagTricksEfficient2016}. Auf eine genaue Erklärungen der Funktionsweise der genannten Algorithmen wird verzichtet, da sie in dieser Arbeit nicht verwendet werden. Gute Zusammenfassungen bieten aber (\cite{sebastianiMachineLearningAutomated2002a}, \cite{ikonomakisTextClassificationUsing2005a} oder \cite{thangarajTextClassificationTechniques2018}). Eine kompakte und übersichtliche Auflistung findet sich bei \citet[17f]{kirkThoughtfulMachineLearning2017}.

### Logistische Regression

Die Logistische Regression ist die Übertragung der linearen Regression auf ein Klassifikationsproblem. Wie der Naive Bayes vergibt sie auf Basis von Textdaten, die in irgendeiner Form umgewandelt werden, eine Klasse. Ähnlich ist auch, dass die Logistischen Regression binäre Klassenentscheidungen treffen oder auch mit mehr als zwei Klassen umgehen kann. Es bietet sich wieder an, mehrere Klassifikatoren für jede Klasse zu bauen. Der Grund ist derselbe. Die Berechnung und Veranschaulichung bei Zwei-Klassen-Problemen ist einfacher \cite[vgl.][75]{jurafskySpeechLanguageProcessing2020}.\\

Um die Logistische Regression zu verstehen, bietet sich ein kleiner Exkurs zur linearen Regression an. Bei Regressionsanalysen wird generell versucht, eine abhängige Variable durch mehrere unabhängige Variablen zu beschreiben. In der einfachsten Form kann ein linearer Zusammenhang mit einer Geraden visualisiert werden. Diese Gerade versucht auf Basis der Daten möglichst viele Datenpunkte in einer Punktewolke zu treffen. Je kleiner der Abstand der Datenpunkte zur Geraden, desto besser beschreibt die Regressionsgerade den Zusammenhang. Um eine Regressionsgerade durch die Datenpunkte zu zeichnen, muss die abhängige Variable metrisch skaliert sein. Das bedeutet, dass sie einen kontinuierlichen Wert annehmen muss. Zum Beispiel kann der Zusammenhang von erreichten Punkten in einer Klausur in Abhängigkeit von der Lernzeit (in Stunden) dargestellt werden. Wenn statt einem stetigen Wert für die Lernzeit lediglich bekannt ist, ob gelernt wurde oder nicht, ist die Darstellung der Abhängigkeit nicht mehr durch eine lineare Regression beschreibbar. An dieser Stelle ermöglicht die Logistische Regression die Darstellung dieser kategorialen Variablen. Das Ergebnis einer Logistischen Regression liegt zwischen $0$ und $1$. Binär ausgedrückt also ob ein Schüler gelernt hat oder nicht oder, auf das Problem der Textklassifikation übertragen, ob ein Dokument zu einer Klasse gehört oder nicht. Die Linien, zu denen die Abstände der Datenpunkte minimiert werden sollen, sind in Abbildung \textbf{\ref{fig: Vergleich Regressionen}} gegenübergestellt\footnote{Abbildung \textbf{\ref{subfigure: Beispielhafte LinReg}} wurde mit randomisierten Daten erstellt. Abbildung \textbf{\ref{subfigure: Beispielhafte Sigmoid}} basiert auf Code aus \href{geeksforgeeks.org/implement-sigmoid-function-using-numpy/}{\underline{geeksforgeeks.org/sigmoid-function/}}}.

\begin{figure}[h]

\begin{subfigure}[b]{0.49\textwidth}

\includegraphics[width=1\textwidth]{./Lin\_Gerade.png}

\subcaption{Lineare Regression}

\label{subfigure: Beispielhafte LinReg}

\end{subfigure}

\hfill

\begin{subfigure}[b]{0.49\textwidth}

\includegraphics[width=1\textwidth]{./Sig\_Kurve.png}

\subcaption{Logistische Regression}

\label{subfigure: Beispielhafte Sigmoid}

\end{subfigure}

\caption[Vergleich von linearer und logistischer Regression]{Vergleich von Gerade bzw. Kurve, die bei den Regressionen optimiert werden sollen. }

\label{fig: Vergleich Regressionen}

\end{figure}

Die Logistische Regression ist, ebenso wie der Naive Bayes, ein probabilistischer Klassifikator \cite[vgl.][76]{jurafskySpeechLanguageProcessing2020}. Sie vergibt also nur Wahrscheinlichkeiten für Klassen. Die Werte in Abbildung \textbf{\ref{subfigure: Beispielhafte Sigmoid}} nähern sich also nur der $0$ im Minimum und der $1$ im Maximum an. Anders als der generative Naive Bayes ist die Logistische Regression ein diskriminierender Klassifikator. Es werden also nur die Features gesucht, die zwischen den möglichen Klassen am besten unterscheiden können \cite[vgl.][57]{jurafskySpeechLanguageProcessing2020}.

\newpage

Dabei ist das grundlegende Ziel dasselbe. Die Berechnung der Klasse, in Abhängigkeit von einem Dokument $P(c\mid d)$, wobei das Dokument $d$ wieder als $n$ Features dargestellt werden kann $x\_{1}\cdots x\_{n}$. Im Gegensatz zum Naiven Bayes versucht die logistische Regression die Klasse eines Dokumentes direkt ohne die Berechnung aus Formel \textbf{\ref{Formel: Bayes ohne Nenner}} zu bestimmen \cite[vgl.][76]{jurafskySpeechLanguageProcessing2020}. Für die Bestimmung einer Klasse wird jedes Feature aus den Trainingsdaten $x\_{i}$ mit einem Gewicht $w\_{i}$ multipliziert. Anschließend wird ein sogenannter \textit{Biasterm} dazu addiert. Das Ergebnis ist die Summe der gewichteten Features $z$ \cite[vgl.][77]{jurafskySpeechLanguageProcessing2020}:

\begin{align}\label{Formel: LogReg Formel}

z = (\sum\_{i=1}^{n}w\_{i}x\_{i}) + b

\end{align}

Zur besseren Übersicht wird die Summe der Multiplikationen in Formel \textbf{\ref{Formel: LogReg Formel}} vereinfacht wie folgt dargestellt:

\begin{align}\label{Formel: LogReg Dotproduct}

z = w\_{i}\cdot x\_{i} + b

\end{align}

Diese Schreibweise wird auch \textit{Skalarprodukt} bezeichnet. Die Gewichte geben an, wie wichtig das Feature für die Klassifikation ist. Ein anderer Begriff für den Biasterm ist auch \textit{Intercept}. In der linearen Regression ist das der Punkt, an dem die Gerade die Y-Achse schneiden würde, wenn alle Werte der abhängigen Variablen gleich $0$ wären. Bei der Logistischen Regression beschreibt er eine Art Ausgangswahrscheinlichkeit der Klasse, wenn die Summe der Multiplikationen in Formel \textbf{\ref{Formel: LogReg Formel}} gleich $0$ wäre. Graphisch kann mit der Gewichtung $w$ der Features die Steigung der Kurve eingestellt werden. Während mit dem Biasterm $b$ die Kurve nach links oder rechts verschoben werden kann.

Beide Parameter nehmen reale Zahlenwerte an, sodass das Ergebnis $z$ theoretisch von $- \infty$ bis $+ \infty$ variieren kann \cite[vgl.][77]{jurafskySpeechLanguageProcessing2020}. Um $z$ in eine echte Wahrscheinlichkeit $P(y)$ umzuwandeln, wird eine spezielle Funktion angewendet. Diese nennt sich \textit{logistische} bzw. \textit{Sigmoid} Funktion ($\sigma$) und ist für die Regression namensgebend \cite[vgl.][77f]{jurafskySpeechLanguageProcessing2020}:

\begin{align}\label{Formel: Sigmoidfunktion}

\sigma(z) = \frac{1}{1+e^{-z}}

\end{align}

Durch diese Funktion wird $z$ in einen Wert zwischen $0$ und $1$ umgewandelt. Dieser ist dann die Wahrscheinlichkeit für die Zugehörigkeit zu einer Klasse. Es gilt also:

\begin{align}\label{Wahrscheinlichkeit = Sigma}

P(y=1) = \sigma(z) = \frac{1}{1+e^{-z}}

\end{align}

\newpage

Die Subtraktion dieses Wertes von $1$ ergibt die Wahrscheinlichkeit, dass ein Dokument nicht zur Klasse dazugehört ($P(y=0)$).

\begin{align}\label{Wahrscheinlichkeit nicht Klasse}

P(y=0) = 1 - \sigma(z) = 1 - \frac{1}{1+e^{-z}}

\end{align}

Das bereits bekannte Beispiel der Erkennung von Spam soll die Funktion veranschaulichen. Wieder werden die Termfrequenzen der Wörter aus Tabelle \textbf{\ref{tab: Spam or Ham}} als Features für die Klassifikation verwendet. Des Weiteren wird davon ausgegangen, dass die Gewichte bereits bekannt sind. Das Wort \textit{Bitcoin} hat eine Frequenz von $2$ und das Gewicht sei $3$. Das Wort \textit{Hallo} kommt ebenfalls zweimal vor, hat aber ein Gewicht von $-0,5$. Ein positiver Wert bedeutet einen positiven Einfluss auf die Klassenbestimmung, während ein negativer Wert negativen Einfluss hat. Zudem sei $ b = 0.1$. Enthält ein Testdokument nun diese beiden Wörter, berechnet sich die Wahrscheinlichkeit durch die logistische Regression wie folgt. Zunächst wird die Summe der gewichteten Features berechnet:

\begin{align}

z = (3,-0,5) \ast (2,2) + 0.1 = (3\ast2) + (-0,5\ast2) + 0,1 = \mathbf{5,1}

\end{align}

Das Ergebnis wird durch die Sigmoid-Funktion in Wahrscheinlichkeiten umgewandelt:

\begin{align}

P(+) = \frac{1}{1+e^{-5,1}} \approx \mathbf{0,99} \\

P(-) = 1 - \frac{1}{1+e^{-5,1}} \approx \mathbf{0,01}

\end{align}

In diesem stark vereinfachten Beispiel würde der Klassifikator das Dokument also mit sehr hoher Sicherheit der Klasse Spam (+) zuordnen. Den Ausschlag geben vor allem die Gewichte der Wörter. Hier wurden Gewichte und der Biasterm vorher bestimmt. In der Realität würde ein Klassifikator die optimalen Parameter iterativ bestimmen, um das tatsächliche Label möglichst genau zu treffen \cite[vgl.][80]{jurafskySpeechLanguageProcessing2020}. Dafür werden die Parameter so lange wiederholt angepasst, bis eine optimale Kurve gefunden wird, die die Datenpunkte am besten trifft.\\

Um die Funktionsweise dieser iterativen Anpassung zu verstehen, lohnt sich wieder ein Vergleich zur linearen Regression. Bei dieser werden die Standardabweichungen der Datenpunkte minimiert, um die optimale Gerade zu bestimmen. Im Gegensatz dazu wird bei der Logistischen Regression versucht, die sogenannte \textit{Kostenfunktion}, den \textit{cross-entropy loss}, zu minimieren \cite[vgl.][81]{jurafskySpeechLanguageProcessing2020}. Speziell wird für die Minimierung der Kostenfunktion das \textit{stochastische Gradientenverfahren} verwendet \cite[vgl.][82-85]{jurafskySpeechLanguageProcessing2020}. Dieser Algorithmus berechnet in kleinen Schritten, der Lernrate, immer wieder das Minimum der Kostenfunktion. Da diese nur ein Minimum hat, ist garantiert, dass der Algorithmus dieses auch findet \cite[vgl.][82]{jurafskySpeechLanguageProcessing2020}. Bei der Initialisierung einer Logistischen Regression kann die Lernrate eingestellt werden. Üblicherweise wird mit einer hohen Lernrate begonnen und diese anschließend stetig verkleinert \cite[vgl.][85]{jurafskySpeechLanguageProcessing2020}. Es wird nicht weiter darauf eingegangen, wie die Kostenfunktion durch das stochastische Gradientenverfahren berechnet wird. Zum Verständnis reicht es aus zu wissen, dass je kleiner das Ergebnis der Kostenfunktion ist, desto wahrscheinlich die vorhergesagte Klasse auch die echte Klasse darstellt.\\

Zusammenfassend wird zunächst eine Klasse für die Testdaten geschätzt. Auf Basis dieser Schätzung wird die Kostenfunktion mit Hilfe des stochastischen Gradientenverfahrens minimiert und letztlich die geschätzte Klasse zurückgegeben.\\

Im Fall, dass mehr als zwei Klassen vorhanden sind, würde sich grundsätzlich nicht viel ändern. Die Berechnung und Herleitung ist allerdings etwas komplexer \cite[vgl.][89ff]{jurafskySpeechLanguageProcessing2020}. Hierbei ist das Ergebnis wieder eine Wahrscheinlichkeit über die Zugehörigkeit für jede Klasse, die aufsummiert wieder $1$ ergeben muss. Für diesen Zweck kommen andere Kostenfunktionen und Minimierungsverfahren zum Einsatz. Die Kostenfunktion für solch eine multinomiale logistische Regression ist eine Generalisierung der Sigmoid-Funktion und heißt Softmax. Dieser wird in neuronalen Netzwerken als Aktivierungsfunktion eingesetzt.\\

Vielen Wissenschaftler:innen genügt es, wenn Klassifikatoren ihren Zweck erfüllen. Manche möchten aber zusätzlich verstehen, wie Klassifikatoren zu ihren Entscheidungen kommen bzw. wie wichtig die einzelnen Features für die Klassifikation sind. Beim Naiven Bayes erschwert die schlichte Multiplikation der Features die Interpretierbarkeit. Bei Modellen, die mit Termfrequenzen arbeiten, entspricht eine höhere Termfrequenz in der Regel auch einem höheren Gewicht. Die Logistische Regression vergibt dagegen einzelnen Featuren Gewichte. In Kombination mit statistischen Tests, kann so die Signifikanz eines Features auf den Klassifikationsprozess ermittelt werden \cite[vgl.][91]{jurafskySpeechLanguageProcessing2020}.\\

Neben dem Naiven Bayes gilt die Logistische Regression ebenfalls als Basismethode für die Textklassifikation. Besonders macht sie die Verwandtschaft zu neuronalen Netzwerken. Diese sind im Grunde genommen nichts anderes, als viele Logistische Regressionen hintereinander geschaltet \cite[vgl.][75]{jurafskySpeechLanguageProcessing2020}.

### Support Vektor Machine

Die bislang besprochenen Klassifikatoren werden nur mit Daten trainiert, die das korrekte Klassenlabel haben. Der Naive Bayes lernt aus den Daten die Features, um die Klasse am besten zu beschreiben. Die Logistische Regression versucht nur die Features zu lernen, die sie am besten von anderen Klassen unterscheidet. Im Gegensatz dazu nutzen \acsp\*{SVM} in der Trainingsphase sowohl Daten mit positiven Klassenlabeln, als auch mit negativen Labeln\footnote{Dabei meint \textit{positiv}, die Label der zu vorhersagenden Klasse und mit \textit{negativen} Labeln sind Daten gemeint, die einer anderen Klassen angehören.}. \acsp\*{SVM} versuchen die positiven und negativen Daten mit möglichst großen Abstand voneinander zu separieren \cite[vgl.][68]{kamathDeepLearningNLP2019}. Bei Support Vektor Maschinen bietet sich eine reine graphische Betrachtung der Funktionsweise an. Diese ist, im Gegensatz zur Mathematik dahinter, sehr einfach nachzuvollziehen. Anders als der Naive Bayes und die Logistische Regression, sind \acsp\*{SVM} keine reinen linearen Klassifikatoren. Graphisch würde im zweidimensionalen Raum ein linearer Klassifikator versuchen, die Punkte beider Klassen durch eine Gerade voneinander zu trennen \cite[vgl.][304]{manningIntroductionInformationRetrieval2008}. Der Name \textit{Support Vektoren} oder \textit{Stützvektoren} kommt daher, weil \acsp\*{SVM} nicht alle Datenpunkte respektive Vektoren für die Trennung nutzen. Sie verwenden nur diese, die am nächsten beieinander, aber in unterschiedlichen Klassen sind. Dies wird bei der Betrachtung von Abbildung \textbf{\ref{Figure: SVM Trennung durch Support Vektoren}} deutlich\footnote{Der Code aus Abbildung \textbf{\ref{Figure: SVM Trennung durch Support Vektoren}} basiert auf \href{https://scikit-learn.org/stable/auto\_examples/svm/plot\_separating\_hyperplane.html}{\underline{scikit-learn.org/plotHyperplane.html}}}:

\begin{figure}[h]%option kann noch anders gesetzt werden.

\center

\includegraphics[width=0.76\textwidth]{./Schaubild\_SVM.png}

\caption[Visualisierung von Support Vektor Maschinen]{SVM: Trennung der Daten mit Hilfe der Stützvektoren}

\label{Figure: SVM Trennung durch Support Vektoren}

\end{figure}

Die eingekreisten Punkte beider Farben sind die Stützvektoren, die zur Berechnung des Abstands in Betrachtung kommen. Die durchgehende graue Linie ist die Trennlinie mit dem größten Abstand zu beiden Punktewolken. Dieser Abstand soll bei \acsp\*{SVM} maximiert werden \cite[vgl.][30]{sebastianiMachineLearningAutomated2002a}. Dies wird deutlich, wenn sich folgendes Szenario vorgestellt wird: Angenommen die Trennlinie verläuft nicht wie in Abbildung \textbf{\ref{Figure: SVM Trennung durch Support Vektoren}}, sondern liegt nahe an der blauen Wolke. Ein neuer Punkt, der zur blauen Wolke gehört, aber auf der falschen Seite der Trennlinie liegt, würde missklassifiziert werden.\\

Auch wenn nach \citet[4]{joachimsTextCategorizationSupport1998} die meisten Daten bei der Textklassifikation linear trennbar sind, muss dies nicht immer der Fall sein. Die Aufgabe von \acsp\*{SVM} ist dann, eine lineare Trennung der Daten zu erreichen. Dazu werden sogenannte \textit{Kerneltricks} angewendet. Kerneltricks transformieren die Daten, um sie wieder linear trennbar zu machen. Dafür projizieren sie die Daten in eine oder mehrere höhere Dimensionen \cite[vgl.][115]{kirkThoughtfulMachineLearning2017}. Die Funktionsweise eines Kerneltricks wird in Abbildung \textbf{\ref{fig: Kerneltrick}} vereinfacht dargestellt.

\begin{figure}[h]

\begin{subfigure}[b]{0.49\textwidth}

\includegraphics[width=1\textwidth]{./SVM\_ohne\_Kernel.png}

\subcaption{Eindimensionale Datenpunkte}

\label{subfigure: SVM\_ohne\_Kernel}

\end{subfigure}

\hfill

\begin{subfigure}[b]{0.49\textwidth}

\includegraphics[width=1\textwidth]{./SVN\_Kernel.png}

\subcaption{Projektion der Daten auf zwei Dimensionen}

\label{subfigure: SVM\_mit\_Kernel}

\end{subfigure}

\caption[Vereinfachte Darstellung eines Kerneltricks]{Vereinfachte Darstellung eines Kerneltricks. Hier werden die Daten aus \textbf{\ref{subfigure: SVM\_ohne\_Kernel}} einfach quadriert, um die y-Koordinaten zu generieren.}

\label{fig: Kerneltrick}

\end{figure}

In Abbildung \textbf{\ref{subfigure: SVM\_ohne\_Kernel}}, befinden sich alle Punkte in einer Dimension, also auf einer Linie. Die lineare Trennung der Daten ist so nicht möglich. Egal wie die Trennlinie angesetzt wird, entsteht immer eine Schnittmenge aus Punkten beider Klassen. Um Daten in eine höhere Dimension zu projizieren, werden die Punkte in Abbildung \textbf{\ref{subfigure: SVM\_mit\_Kernel}} quadriert und die daraus resultierenden Werte als Y-Koordinaten verwendet \cite[vgl.][330ff]{manningIntroductionInformationRetrieval2008}. Nun lassen sich die Daten beider Klassen wieder linear voneinander trennen. In Räumen mit mehr als zwei Dimensionen kann die Trennung nicht mehr durch eine einfache Linie beschrieben werden. In diesen Fällen erfolgt die Trennung dann durch Ebenen bzw. Hyperebenen in hochdimensionalen Räumen.

\newpage

Im dreidimensionalen Raum könnte diese Ebene wie in Abbildung \textbf{\ref{Figure: Trennung durch Ebene}} aussehen\footnote{Code und Daten für Abbildung \textbf{\ref{Figure: Trennung durch Ebene}} von \href{https://towardsdatascience.com/support-vector-machines-svm-clearly-explained-a-python-tutorial-for-classification-problems-29c539f3ad8}{\underline{towardsdatascience.com/svm-clearly-explained }} übernommen und angepasst.}.

\begin{figure}[h]%option kann noch anders gesetzt werden.

\includegraphics[width=1\textwidth]{./SVM\_3D.png}

\caption[Trennung der Daten im dreidimensionalen Raum durch eine Ebene]{Trennung der Daten im dreidimensionalen Raum durch eine Ebene}

\label{Figure: Trennung durch Ebene}

\end{figure}

In Realität ist die Mathematik bei der Anwendung von Kerneln komplexer als eine einfache Quadrierung. Es gibt verschiedene Kernelfunktionen, die für in \acsp\*{SVM} zum Einsatz kommen können. Diese unterscheiden sich dabei in der Berechnung der Projektion auf höhere Ebenen. Gerne verwendete Kernel sind der polynomiale oder der \ac{rbf}-Kernel \cite[vgl.][333]{manningIntroductionInformationRetrieval2008}. Bei der praktischen Anwendung sei es nach \citet[2]{joachimsTextCategorizationSupport1998} zudem sinnvoll, unterschiedliche Kernel auszuprobieren. In seinen Versuchen ist die Leistung der oben genannten Kernel allerdings nahezu identisch \cite[vgl.][5]{joachimsTextCategorizationSupport1998}. Auch bei \citet{leopoldTextCategorizationSupport2002} fallen die Unterschiede nur gering aus. Ein Nachteil von Kernelfunktionen ist, dass sie sich manchmal zu gut an die Daten anpassen und somit zu sehr harten Entscheidungsgrenzen kommen \cite[vgl.][118]{kirkThoughtfulMachineLearning2017}. Wenn beispielsweise nur ein einzelner Punkt auf der falschen Seite der Hyperebene oder Linie liegt, sollte eine Missklassifikation erlaubt sein.

\newpage

Der "Härtegrad" der Entscheidung kann mit verschiedenen Parametern eingestellt werden. Diese unterscheiden sich je nach verwendeten Kernel. Der Parameter \textbf{C} kontrolliert den Kompromiss zwischen dem Grad der Missklassifikation der erlaubt wird und der optimalen Abgrenzung der Datenpunkte \cite[vgl.][2]{joachimsTextCategorizationSupport1998}. Wird der Parameter verringert, wird mehr Raum für Missklassifikation zugelassen und es wird eher der allgemeine Trend erfasst. Graphisch vergrößert sich der Trennbereich und es können Punkte, anders als in Abbildung \textbf{\ref{Figure: SVM Trennung durch Support Vektoren}} dargestellt, innerhalb des Trennbereiches liegen. Hohe Werte für den Parameter \textbf{C} sorgen im Umkehrschluss für gegenteilige Trends. Einzelne Punkte werden stärker gewichtet, um so Missklassifikationen zu verhindern. Dadurch kann der Abstand zwischen den Punktwolken unter Umständen geringer werden.\\

Die richtige Einstellung der Parameter hat bei \acsp\*{SVM} keine zu große Bedeutung, da diese laut \citet[6]{joachimsTextCategorizationSupport1998} automatisch schon relativ gut sei. Diese Einschätzung teilt auch \citet[30]{sebastianiMachineLearningAutomated2002a}. Darüber hinaus berichten \citet{zeheSentimentAnalysisGerman2017} bei der Verwendung von \acsp\*{SVM} für die Analyse des Sentiments deutscher Literatur, dass die Unterschiede bei der Verwendung verschiedener Werte für \textbf{C} gewöhnlich sehr klein waren. Lediglich bei \ac{rbf}-Kerneln war die Leistung der Klassifikation stärker von der Einstellung der Parameter betroffen, die Leistungsunterschiede waren aber trotzdem gering \cite[vgl.][5]{zeheSentimentAnalysisGerman2017}.\\

Support Vektor Maschinen haben gegenüber den anderen vorgestellten Klassifikatoren einige Vorteile. Die Reduktion der Features spielt für die Klassifikation keine große Rolle, da \acsp\*{SVM} auch mit hoch-dimensionalen Daten operieren können (\cite[vgl.][111]{kirkThoughtfulMachineLearning2017}, \cite[30]{sebastianiMachineLearningAutomated2002a}, \cite[6]{joachimsTextCategorizationSupport1998}). Dadurch skalieren sie sehr gut \cite[vgl.][30]{sebastianiMachineLearningAutomated2002a}. Unterschiede sind ähnlich wie bei der Parametereinstellung marginal. So berichten \citet[435ff]{leopoldTextCategorizationSupport2002} beispielsweise, dass eine Lemmatisierung bei \ac{TF-IDF}-gewichteten Daten zu leicht verbesserten Resultaten führten.\\

Im Vergleich zum Naiven Bayes ist die Trainingsphase für \acsp\*{SVM} länger \cite[vgl.][5]{joachimsTextCategorizationSupport1998}. Im Vergleich zur Logistischen Regression ist der rechentechnische Aufwand allerdings höher \cite[vgl.][15]{aurangzebReviewMachineLearning2010}. Sie erreichen aber generell höhere Genauigkeiten bei der Klassifikation als andere Klassifikatoren \cite[vgl.][423]{leopoldTextCategorizationSupport2002}, \cite[vgl.][38]{sebastianiMachineLearningAutomated2002a} und gelten deshalb als eine der effektivsten überwachten Lernmethoden für die Textklassifikation \cite[vgl.][1378]{wangOptimalSVMBasedText2006}.

### BERT / Deep Learning

### Evaluation von Modellen

### Warum Python

# Datengrundlage

## Struktur

Die Trainingsdaten stammen aus dem PAN und liegen im XML Format vor.

Ein Artikel hat drei Root-Elemente:

1. Austausch Steuerung
2. Metadaten
3. Inhalt

Jedes dieser Root-Elemente vereint weitere XML-Elemente unter sich. Die Metadaten enthalten beispielsweise folgende Elemente:

1. Doc-ID (Dokumenten-ID)
2. Dokumentationsstelle
3. Erscheinungsdatum
4. Erfassungsdatum und Zeit
5. Quelle
6. Klassifikationsliste
7. Personenliste
8. Deskriptorenliste
9. Präsentationsform
10. U.a.

Im Root-Element Inhalt ist selbsterklärend der Inhalt eines Dokumentes erfasst:

1. Titel mit Haupttitel, Sonstiger Titel und Seitentitel
2. Volltext
3. Anzahl der Worte und Volltextseiten

Beispielhaft sehen Sie in Abbildung XY den Baum eines XML-Dokuments:

## Statistiken

|  |  |  |
| --- | --- | --- |
| PAN-Kürzel | Präsentationsform | Anzahl Testdaten |
| BER | Bericht | 1.070.970 |
| INT | Interview | 115.759 |
| REZ | Rezension | 101.473 |
| GRF | Grafik | 55.767 |
| KOM | Kommentar | 53.888 |
| REP | Reportage | 13.584 |
| SER | Serie | 13.320 |
| TIT | Titelthema | 12.662 |
| ESS | Essay | 12.056 |
| KTE | Karte | 6.480 |
| PRM | Pressemitteilung | 5.081 |
| WRT | Wortlaut | 3.378 |
| CHR | Chronologie | 2.380 |
| ANA | Analysis (nur DW) | 1.837 |
| REST | Restliche P-Formen unter 1000 | 3.494 |



# Methodik

50 min Zeit warden benötigt, um das XML in einen Dataframe-Objekt umzuwundeln und anschließend als pkl.-Datei zu speichern. Die Tokenisierung und das Stemming dauern dann nochmal ungefähr eineinhalb Stunden. Die anschließende Lemmatisierung benötigt von alle Preprocessing-Maßnahmen mit XYZ am längsten.

## Warum Python

Es gibt eine riesige an Anzahl Programmiersprachen mit verschiedenen Architekturen und Anwendungsgebieten. Zu den Bekanntesten gehört die Sprache Python. Mit Hilfe von Frameworks lässt sich Python für nahezu jeden Verwendungszweck einsetzen. Python wird durch \textit{Python Software Foundation} betreut, aber von einer offenen Gemeinschaft entwickelt. Dadurch ist die Nutzung kostenfrei und auf allen Plattformen verfügbar. Durch den Einfluss von Forschung und Wirtschaft ist Python besonders beliebt im Bereich der \textit{Data Science} \cite[vgl.][4]{kamathDeepLearningNLP2019}.\\

Ein anderer Aspekt, der zur Popularität beiträgt, ist der einfache, gut strukturierte Aufbau der Sprache \cite[vgl.][Vorworf]{birdNaturalLanguageProcessing2009}. Der Code bleibt, im Vergleich zu anderen Programmiersprachen, sehr übersichtlich und ist einfach lesbar. Für den Bereich der Data Science gibt es eine Vielzahl von Programmbibliotheken, die auf Python basieren. Viele Deep Learning Frameworks stellen außerdem eine Python-Schnittestelle zur Verfügung \cite[vgl.][4]{kamathDeepLearningNLP2019}.\\

Aus diesen Gründen wird in dieser Arbeit Python in Verbindung mit diversen weiteren Programmbibliotheken für die Verarbeitung von natürlicher Sprache verwendet. Für die Verwaltung, Strukturierung und Berechnung von Vektoren eignen sich die Programmbibliotheken \textit{Pandas} (\cite{mckinneyDataStructuresStatistical2010}) und \textit{NumPy} (\cite{harrisArrayProgrammingNumPy2020}) besonders gut. \textit{SciKit-learn} oder kurz \textit{Sklearn} (\cite{JMLR:v12:pedregosa11a}) stellt eine große Anzahl an Werkzeugen für den Umgang mit Algorithmen des maschinellen Lernens bereit \cite[vgl.][37]{kirkThoughtfulMachineLearning2017}. In Sklearn sind alle gängigen Klassifikationsalgorithmen, sowie Vektorisierungsmethoden enthalten.\\

Das \textit{\ac{NLTK}} von \citet{birdNaturalLanguageProcessing2009} bietet spezielle Algorithmen für die Verarbeitung natürlicher Sprache. Da \ac\*{NLTK} nicht bei allen Methoden die deutsche Sprache unterstützt, kann zudem die Programmbibliothek \textit{Textblob} verwendet werden (\cite{loriaTextblobSimplePythonic}).\\

In einem Versuch wurden die später vorgestellten Daten mit beiden Bibliotheken tokenisiert. Obwohl \ac\*{NLTK} nicht für die deutsche Sprache konzipiert ist, liegen die Ergebnisse beider Methoden nahe beieinander. Textblob wird in dieser Arbeit für die Bestimmung der Tokens und \ac\*{POS}-Tags verwendet. Alle andere notwendigen Schritte erfolgen mit Hilfe des \ac{NLTK} und Sklearn.\\

Einen umfangreichen Überblick über Quellen und Methoden im Bereich Data Science bietet folgende kuratierte Liste: \href{https://github.com/krzjoa/awesome-python-data-science#readme}{\underline{github.com/awesome-python-data-science}}.\\

Die Programmbibliotheken sind in der Regel gut dokumentiert. Generell finden sich im World Wide Web viele Quellen und Anleitungen. Eine Einstiegshürde kann allerdings die Sprachbarriere sein. Die meisten Anleitungen sind nur auf Englisch verfasst.

\newpage

Außerdem funktionieren die meisten Programmbibliotheken für die Analyse von natürlicher Sprache häufig nur für wenige viel gesprochene Sprachen - die meisten werden nur auf Englisch optimiert. So können beispielsweise Unterschiede bei der Effektivität entstehen. \citet{zeheSentimentAnalysisGerman2017} untersuchen beispielswiese mit Hilfe von \acsp\*{SVM} und neuronalen Netzwerken das Sentiment von deutschen Novellen. Die Genauigkeit der Modelle fällt bei den deutschen Texten im Gegensatz zu ihren englischen Gegenstücken immer ab. Außerdem wird durch den Fokus auf viel gesprochene Sprachen die Nutzung der Bibliotheken für kleinere Sprachen erschwert.

## Pipeline

Über die in Abschnitt \textbf{\ref{section: Programmierschnittstelle}} beschriebene Programmierschnittstelle werden die Sendungen jeder Sendungsreihe heruntergeladen und in ein CSV-Format umgewandelt. Anschließend werden die Texte jeder Sendung mit einem Pythonskript vorverarbeitet (siehe Auszug des Skripts in \textbf{\ref{Code: function: Tokenize}}). Das verwendete Skript erfüllt mehrere Funktionen.\\

Im ersten Schritt werden die Daten aus der CSV-Datei geladen und die Texte der Sendungen einer Funktion übergeben, die sie tokenisiert. Die Funktion nutzt dabei Methoden der Programmbibliothek \ac{NLTK}. Zunächst wird der Funktion eine Liste an Texten übergeben. Hier besteht die Liste aus den Texten für jede Sendung. Jeder Text in dieser Liste wird in Sätze und in einzelne Wörter aufgeteilt. Danach werden die Satzzeichen entfernt. Die Wörter eines Textes werden am Ende wieder ohne Satzzeichen zusammengefügt. Darüber hinaus nutzt die Funktion den in Abschnitt \textbf{\ref{section: Preprocessing}} vorgestellten Snowballstemmer, um die Wörter auf ihren Stamm zu reduzieren. Die Funktion gibt als Ergebnis eine Liste aus zwei Listen zurück. Eine für die tokenisierten Daten, sowie eine für die stammformreduzierten Daten. Beide werden am Ende des Skripts in Spalten einer CSV-Datei umgewandelt.

\newpage

\begin{lstlisting}[caption=Funktion zur Tokenisierung, label=Code: function: Tokenize]

def tokenizeText(text\_list):

stripped = []

stemmed = []

# Initialisiert Snowballstemmer

stemmer = SnowballStemmer('german')

for sendung in text\_list:

sent\_of\_sendung = []

stem\_of\_sendung = []

# Zerlegt jeden Text in Saetze

tok\_sentences = sent\_tokenize(sendung, language='german')

# Tokenisiert jedes Wort in jedem Satz

for satz in tok\_sentences:

# Tokenisierung

words = word\_tokenize(satz)

# Entfernt Satzzeichen

new\_words= [word for word in words if word.isalnum()]

# Joined tokenisierte Woerter jedes Satzes

# einer Sendung

sent\_of\_sendung.append(" ".join(new\_words))

# Stemming

stemmed\_words = [stemmer.stem(word) for word

in new\_words]

stem\_of\_sendung.append(" ".join(stemmed\_words))

# Joined tokensierte Saetze einer Sendung

stripped.append(" ".join(sent\_of\_sendung))

# Joined gestemmte Saetze einer Sendung

stemmed.append(" ".join(stem\_of\_sendung))

return [

stripped,

stemmed

]

\end{lstlisting}

Für die Lemmatisierung der Texte müssen diese erst mit ihren POS-Tags ausgezeichnet werden. Da NLTK dies nicht direkt für die deutsche Sprache unterstützt, müssen weitere Abhängigkeiten installiert werden. Für die Lemmatisierung wird dem Tutorial von \citet{konradAccuratePartofSpeechTagging2016} gefolgt. Konrad entwickelte auch den Lemmatisierungsalgorithmus (\cite{konradGermalemmaLemmatizerGermana}). Für das POS-Tagging wird der Algorithmus aus \href{https://github.com/ptnplanet/NLTK-Contributions}{\underline{https://github.com/ptnplanet/NLTK-Contributions}} verwendet. Dieser ist auf deutschen Textdaten aus dem TIGER-Corpus trainiert (\cite{brantsTIGERLinguisticInterpretation2004}). Der Funktion (\textbf{\ref{Code: Function: POS-Tagging}}) wird die Liste von Wörtern übergeben, die von Satzzeichen entfernten wurden. Anschließend wird der trainierte Korpus genutzt, um die Texte mit ihren POS-Tags zu markieren.

\newpage

Die Funktion gibt letztlich eine Liste aus sogenannten Tuplen zurück. Jedes Tuple besteht aus einem Wort mit dem korrespondierden POS-Tag: \lstinline|("Professor", "NN")|.

\begin{lstlisting}[caption=Funktion zum Part-of-speech-Tagging, label=Code: Function: POS-Tagging]

# POS-Tagging

def TagPOS\_Text(text\_list):

pos\_list = []

for row in text\_list:

# Splittet Woerter am Leerzeichen

new\_row = row.split(' ')

# Markiert die POS-Tags

tagged\_sent = tagger.tag(new\_row)

# Joined markierte Woerter

pos\_list.append(tagged\_sent)

# returned Liste aus Tuplen

return pos\_list

\end{lstlisting}

Die Liste aus Tuplen nutzt letztlich eine Funktion zur Lemmatisierung als Input und liefert die lemmatisierten Texte zurück (siehe Auszug \textbf{\ref{Code: Function: Lemmatize}}).

\begin{lstlisting}[caption=Funktion zur Lemmatisierung, label=Code: Function: Lemmatize]

# Lemmatisierung

def lemmatizeText(text\_list):

lemmatized = []

for lists in text\_list:

lemmasOF = []

# Try\_Catch noetig, weil nicht alle POS-Tags

# unterstuetzt werden. Bei Fehler wird einfach

# das Wort zurueckgegeben.

for tuples in lists:

try:

lemma = lemmatizer.find\_lemma(tuples[0],

tuples[1])

lemmasOF.append(lemma)

except ValueError:

lemmasOF.append(tuples[0])

continue

lemmatized.append(" ".join(lemmasOF))

return lemmatized

\end{lstlisting}

Die lemmatisierten Texte werden anschließend ebenfalls in eine Spalte abgetragen und in der CSV-Datei gespeichert. Dieser Prozess wird für alle Sendungen jeder Sendungsreihe wiederholt. Dadurch entsteht eine einzige CSV-Datei, die alle Sendungen enthält. Diese Datei bildet die Basis für die Klassifikationsalgorithmen.

## Parameter

Neben den rohen Wörtern dienen die lemmatisierten und stammformreduzierten Wörter als Trainingsdaten. Dafür muss lediglich die Variable in Zeile \lstinline|2| angepasst werden, um die gewünschte Spalte aus der CSV-Datei in die Variable zu laden. Außer der Termfrequenz werden noch die One-Hot Enkodierung und das \ac{TF-IDF}-Maß als Vektorisierungsmethoden verwendet. Für die One-Hot Enkodierung wird dem \lstinline|CountVectorizer()| ein zusätzlicher Parameter in der Parameterliste übergeben (\lstinline |vect\_\_binary=True|). Dieser kappt die Termfrequenz bei $1$ wenn ein Wort im Text vorkommt und vergibt eine $0$ für alle nicht im Dokument, aber im Vokabular des Korpus vorkommenden Wörter. Für das TF-IDF-Maß wird statt dem \lstinline|CountVectorizer()|, der \lstinline|TfidfVectorizer()| verwendet.\\

Alle Klassifikatoren werden mit den im Codebeispiel \textbf{\ref{Code: Function: Classify}} beschriebenen Parametern \lstinline|vect\_\_lowercase| und \lstinline|vect\_\_stop\_words| für die Vektorisierungsmethoden getestet. Die Parameter \lstinline|n\_splits|, \lstinline|shuffle| und \lstinline|random\_state| der Crossvalidierung bleiben ebenfalls identisch.\\

Für die Klassifikatoren ist die Parametereinstellung unterschiedlich. Der Naive Bayes ist durch seinen einfachen Aufbau nicht so abhängig von der richtigen Parameterauswahl, wie die Logistische Regression oder die \acsp\*{SVM} mit rbf-Kernel. Wichtig ist der bereits erwähnte Parameter \lstinline|alpha|, um ein Smoothing durchzuführen. Dadurch werden Nullwahrscheinlichkeiten verhindert.\\

Bei der logistischen Regression gibt es mehrere optimierbare Parameter. Codebeispiel \textbf{\ref{Code: Parameter: LogReg}} zeigt, welche Parameter getestet werden.

\begin{lstlisting}[caption=Parametereinstellung für die Logistische Regression, label=Code: Parameter: LogReg]

params = {

"log\_\_max\_iter": [500, 1000, 1500, 2000],

"log\_\_solver": ['lbfgs', 'sag', 'saga'],

"log\_\_C": [0.001, 0.01, 0.1, 1],

}

\end{lstlisting}

Ziel der logistischen Regression ist die Minimierung der Kostenfunktion durch ein stochastisches Gradientverfahren. Dieses bestimmt das Minimum iterativ. Mit \lstinline|max\_iter| wird die Anzahl der Iterationen festgelegt, um das Minimum zu finden. Ist der Klassifikator zu komplex oder sind die Daten schlecht klassifizierbar, werden in der Regel mehr Iterationen benötigt, um den Klassifikator auf die Daten anzupassen. Der \lstinline|solver| spezifiziert das verwendete Gradientverfahren. Der Parameter \lstinline|C| reguliert die Klassifikation. Er steuert, wie im Kontext der \acsp\*{SVM} beschrieben (siehe Abschnitt \textbf{\ref{subsection: Support Vector Maschinen}}), den Grad der erlaubten Missklassifizierung, um eine zu starke Anpassung an die Daten zu verhindern. Dadurch würde der Klassifikator die Trainingsdaten zwar perfekt beschreiben, könnte aber neue Daten nur schlecht generalisieren.\\

In Auszug \textbf{\ref{Code: Parameter: SVM}} sind die Parameter für \acsp\*{SVM} zu sehen.

\begin{lstlisting}[caption=Parametereinstellung für Support Vektor Maschinen, label=Code: Parameter: SVM]

params = {

"svm\_\_C": [0.01, 0.1, 1.0],

"svm\_\_kernel": ['linear', 'poly', 'rbf']

}

\end{lstlisting}

Wie für die Logistische Regression wird für \acsp\*{SVM} der Parameter \lstinline|C| eingestellt. Dazu kommt noch die Auswahl eines Kernels.\\

Neben den hier beschriebenen Parametern bieten die Klassen und Methoden aus Sklearn noch diverse weitere einstellbare Optionen, um die Methoden der Programmbibliothek zu optimieren. Sind die Parameter nicht weiter spezifiziert, greift Sklearn auf die Default Einstellungen zurück. Diese sind in den meisten Fällen auch völlig ausreichend, um ein Klassifikationsergebnis zu erzielen.

Zusammenfassend wurden folgende mögliche Preprocessing und Modellkonfigurationen miteinander getestet:

1. Mit und ohne Stoppwortliste

2. Auf Erfahrungen basierend nur mit gestrippten Texten gearbeitet. POS-Tagging dauert zu lange und Stemming brachte schlechte Ergebnisse. Wenn nötig einfach faken, dass ich das getestet habe

3. Verschiedene Trainingsmengen. 5000 bis alle Dokumente. Es zeigte sich grob gesagt: je mehr desto besser, mindestens scheinen 5k Dokumente benötigt zu werden.

4. Verschiedene Klassifikatoren. Meistens aber SGD-SVM (linear) oder Logistische Regression. SGD linear SVM am effizientesten.

5. Verschiedene Klassengrößen. Nur mit den frequentesten oder nur mit den infrequentesten.

6. Mit Titeln und Seitentiteln, plus ersten 100 bzw. 300 Wörter

7. Nur die ersten 100 - 300 Wörter

8. Alle Klassen mit gleich vielen Trainingsdaten

9. Klassen automatisch balanciert.

# Ergebnisse

Inwieweit diese manuell klassifizierten Dokumente für ein Training konsistent genug sind, müsste im Rahmen einer ersten Iteration des PoCs geprüft werden. Zu klären wäre im Rahmen der Iteration auch, ob für alle Gattungen ausreichende Mengen an Dokumenten vorhanden sind (Überrepräsentanz, Unterrepräsentanz) und wie groß der Aufwand für eine entsprechende Aufbereitung der Trainingsdaten wäre. Im Rahmen des PoCs sollte ein erster Klassifikator trainiert werden, der eine automatisierte Zuordnung der Gattungen ermöglicht. Geklärt werden müsste:

 Welche Metriken zur Bewertung und Qualitätssicherung des Klassifizierens herangezogen werden können?

 Für welche Gattungen eine automatisierte Erkennung möglich ist bzw. für welche nicht?

 Wie hoch der Trainingsaufwand pro Gattung wäre?

 In Relation zum Nutzen (Häufigkeit / Prio): Wo sich eine automatisierte Erkennung lohnt, wo nicht?

## Proof of Concept

# Schluss

## Best practices

## Lessons learned

## Ausblick

# Projektbeschreibung und Einordnung

Dies ist eine Überschrift ohne Zahl – Ebene 1

Dies ist der Standardtext.

Auch das Inhaltsverzeichnis mit aktualisiert werden. Mit Klick auf „Felder aktualisieren“.

## Dies ist eine Überschrift x.x (z.B. 1.1.) – Ebene 2

So wird eine Beispielgrafik beschriftet (Rechtsklick und „Beschriftung einfügen „ auswählen:



Abbildung - Beispielgrafik - Überschriften Formatvorlagen [Quelle]

So sieht eine Beispieltabelle aus:

Rechtsklick und „Beschriftung einfügen „ auswählen, dann „Tabelle“ auswählen und den Text oben eingeben.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Name | Anzahl | xx | yy | zz | Bemerkung | Status |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

Tabelle – Beispieltabelle [Quelle]

## Zielsetzung und projektbezogene Ergebnistypen

## Konkrete Lieferobjekte

…

### Dies ist eine Überschrift n.n.n – Ebene 3

…

## Analyse, Ist-Zustand und Rahmenbedingungen

…

## Brauch ich nicht Vorgehensweise und Methoden zur Bedarfsanalyse

…

# Analyse- und Gestaltungsteil

…

## Stakeholderanalyse

…

## SWOT-Analyse

…

Quellen

Monografien

[M\_....] …

Sammelwerke

[S\_....]

Aufsätze

[A\_....] …

Elektronische Dokumente (aus gesicherten Quellen)

[E\_....]

Sonstige Quellen

[X\_....] …

Abbildungen

[Abbildung 1 - Beispielgrafik - Überschriften Formatvorlagen 1](#_Toc45521115)

Tabellen

[Tabelle 1 - Beispieltabelle 1](#_Toc45521377)

*Mit Klick auf „Felder aktualisieren“ aktualisiert sich die Ansicht entsprechend der Angaben weiter oben .*

Abkürzungen und Akronyme

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ABD | Archiv Bibliothek Dokumentation |  | MINT | Medien Integrierende Netzwerk Technologie (CMS zur Organisation aktueller Produktionsmaterialien) |
| BKN | Bürokommunikationsnetz |  | NiF | Nachricht im Film |
| CF | Clean Feed |  | PN | Produktionsnetz |
| DAS | Digitales Archivsystem |  | Prod.nr. | Produktionsnummer |
| EDL | Edit Decision List |  | RMZE | Rechtemanagement und Zentraleinkauf |
| FCC | File Conversion Center |  | RUKAS | Rechte- und Kostenauskunftssystem |
| FSDB | Fernsehdatenbank |  | SLA | Service-Level-Agreement |
| GF | Geschäftsfeld |  | ST | Sendeton |
| HiRes | High Resolution – Material in hochauflösender Qualität (Sende-Qualität) |  | TC | Timecode |
| IT | Internationaler Ton |  | USP | Unique Selling Proposition |
| LoRes | Low Resolution – Material mit geringer Auflösung (Ansichtsqualität) |  | vfm | Verein für Medieninformation und Mediendokumentare |
|  |  |  | Vgl. | Vergleiche |
|  |  |  | z.B. | Zum Beispiel |

Anhang

**…**

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig erstellt und keine anderen als die angegebenen Hilfsmittel und Quellen verwendet habe.

Soweit ich auf fremde Materialien, Texte oder Gedankengänge zurückgegriffen habe, enthalten meine Ausführungen vollständige und eindeutige Verweise auf die Urheber und Quellen.

Alle weiteren Inhalte der vorgelegten Arbeit stammen von mir im urheberrechtlichen Sinn, soweit keine Verweise und Zitate erfolgen.

Mir ist bekannt, dass ein Täuschungsversuch vorliegt, wenn die vorstehende Erklärung sich als unrichtig erweist.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
**Ort, Datum Unterschrift**

1. Mit diesen Verfahren beschäftigen sich beispielweise die Dieburg-Projekte von Constantin Förster (BR) und Jana Gierden (SWR) aus dem Jahr 2022. [↑](#footnote-ref-1)
2. Der Desk ist die zentrale Anlaufstelle für Recherchen und steht allen Redaktionen des SWRs zur Verfügung. [↑](#footnote-ref-2)
3. Mehr dazu in Kapitel XY Change Requests [↑](#footnote-ref-3)