HAW HAMBURG

BACHELORARBEIT

Entwicklung eines Algorithmus zur Klassifizierung von Twitter Beiträgen deutscher Politiker zur Parteiangehörigkeit

vorgelegt am 28.08.2023

Florian Weichbrodt

Erstprüferin: Prof. Dr.-Ing. Sabine Schumann
Zweitprüferin: Prof. Dr. Larissa Putzar

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Department Medientechnik
Finkenau 35
20081 Hamburg

Zusammenfassung

Die politische Landschaft auf Social Media Plattformen ist undurchschaubar, da Algorithmen Nutzenden nur Beiträge anzeigen, die seinen Präferenzen entsprechen. Dies kann zu eingeschränkter Meinungsbildung führen. Um dieses Problem anzugehen, wird in dieser Arbeit mittels deep learning Methoden aus dem Fachbereich Künstliche Intelligenz und neuronaler Netze ein Algorithmus trainiert, der Tweets deutscher PolitikerInnen verschiedener Parteien des Bundestags 2023 analysiert und diese Tweets der jeweiligen Partei zuordnet. Dies soll dem Nutzenden ermöglichen, ähnlich wie der Wahl-o-Mat, eigene Aussagen in Form von Tweets in das System zu speisen und systematisch zu testen, wie sie in den unterschiedlichen Parteien resonieren. Die dafür benötigten Daten werden von der Twitter API mithilfe der Python Bibliothek snscrape ausgelesen und für das Trainieren des Models, anhand der Schritte der Knowledge Discovery in Databases Methode, vorbereitet. Die Modellarchitektur setzt sich aus einem BERT-Modell zusammen, gefolgt von einer abgewandelten Version des KimCNNs [Ki 2014]. Das Ziel ist, eine Kombination aus kontextualer Informationsextraktion durch BERT und syntaktischer Informationsextraktion durch ein CNN (Convolutional Neural Network) herzustellen. Die anfängliche Genauigkeit von 41 % im ersten Trainingsdurchlauf ist unzureichend, weshalb anschließend unterschiedlichste Hyperparameter angepasst werden. Dazu zählen u. A. die Filteranzahl der convolutional-layers, die Regularisierung in Form von dropout, die Anzahl der Neuronen der dense-layers sowie die Anpassung der Modellarchitektur. Schließlich erzielt das Modell mit der höchsten Genauigkeit 46 %. Dies reicht allerdings nicht für zuverlässige Vorhersagen aus. Der Ursprung der niedrigen Genauigkeit wird letztlich auf die Art des neuronalen Netzwerks und der Daten selbst zurückgeführt.

Abstract

The political landscape on social media platforms is unfathomable, due to algorithms displaying the user only a selection of posts based on user preferences, leading to uninformed opinion forming. To combat this problem, this study focuses on training a neural network by using methods from the field of artificial intelligence, specifically deep learning, that classifies tweets from German politicians of various parties residing in the Bundestag 2023 to their respective party affiliation. This aims to enable systematic testing of personal statements to compare what party resonates with it best. The required data is sourced from the Twitter API using Python‘s library snscrape and prepared through Knowledge Discovery in Databases. The model consists of a BERT (Bidirectional Encoder Representation from Transformers) language model followed by an alteration of the KimCNN [Ki 2014]. This combines contextual information extraction through BERT and ssyntactical information through a CNN (Convolutional Neural Network). The initial training resulted in an insufficient accuracy of 41 %, which lead to adapting a multitude of hyperparameters such as the number of filters in a convolutional-layer, regularization by using dropout as well as changing the model architecture. The highest achieved accuracy is 46 %, which still is insufficient to rely on the predictions of the model. The origin of these results is attributed to the neural network type and the data itself.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API ..... *Application Programming Interface*
BERT ..... *Bidirectional Encoder Representations from Transformers*
BoW ..... *Bag of Words*
CBOW ..... *Continuous Bag of Words*
CL ..... *Convolutional-layer*
CNN ..... *Convolutional Neural Network*
DL ..... *Deep learning*
GLUE ..... *General Language Understanding Evaluation*
GRU ..... *Gated Recurrent Unit*
KDD ..... *Knowledge Discovery in Databases*
KI ..... *Künstliche Intelligenz*
LM ..... *Language Model*
LSTM ..... *Long Short-Term Memory Neural Network*
ML ..... *Machine learning*
MLM ..... *Masked Language Modeling*
NLP ..... *Natural Language Processing*
NN ..... *Neuronales Netz/Netzwerk*
NSP ..... *Next Sentence Prediction*
OOV ..... *Out of vocabulary*
RNN ..... *Recurrent Neural Network*
RoBERTa ..... *Robustly optimized BERT approach*
SA ..... *self-attention*
SG ..... *Skip-Gram*
SMOTE ..... *Synthetic Minority Oversampling Technique*
STM ..... *Short-Term Memory*
VSM ..... *Vector Space Model*

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Heutzutage findet Politik weniger in lokalen Gemeinden statt, sondern hauptsächlich über soziale Medien in Text-, Bild- oder Videoform. Dies resultiert in einer umfangreichen politischen Landschaft, nur einen Klick entfernt. In dieser politischen Landschaft kann es allerdings schwierig sein sich zurechtzufinden und zu verstehen, welche Aussagen und Meinungen am meisten mit den Eigenen resonieren. Durch Algorithmen, die den Nutzenden nur bestimmte Beiträge anzeigen, kann schnell der Überblick über das Gesamtgeschehen verloren gehen. Die sogenannte Filterbubble ist ein übliches Phänomen auf den meisten Social Media Plattformen und wirkt sich als eine Art Zensur der Informationen aus, die die Nutzenden erhalten. Die Beiträge und Informationen, die den Nutzenden angezeigt werden, sind stark zugeschnitten, um die durch unbekannte Algorithmen ermittelten Interessen der Nutzenden akkurat darzustellen, um sie so lange wie möglich auf der Plattform zu halten. Dadurch gehen wichtige Informationen verloren, die für eine Meinungsbildung ausschlaggebend sein könnten [StMaGe 2020]. Im Umkehrschluss kann dadurch eine falsche Wahrnehmung des aktuellen politischen Geschehens entstehen, was dazu führen könnte, dass die Nutzenden Personen Parteien und PolitikerInnen unterstützen, die deren Interessen nicht bestmöglich widerspiegeln. Zwar können alle Beiträge öffentlich eingesehen werden, aber nur auf den jeweiligen Profilen der PolitikerInnen oder über die Suchfunktion, was eine weitere nötige Interaktion für den Nutzenden bedeutet. Falls eine PolitikerIn über eine Partei behauptet, sie vertrete eine bestimmte Meinung zu einem Thema, obwohl dies nicht der Fall ist, weiß der Nutzende außerdem nicht, ob diese Aussage stimmt. Stattdessen müsste sich der Nutzende durch weitere Quellen informieren und sich durch kritisches Denken einen eigenen Standpunkt bilden. Wenn dies nicht durchgeführt wird, bleibt dem Nutzenden lediglich der ursprüngliche Beitrag im Sinn und dieser wird häufig als Wahrheit eingestuft, da in diesem Moment eventuell keine Gegenargumente existieren.

Durch die Verfügbarkeit dieser Daten auf öffentlichen Plattformen eröffnen sich allerdings Möglichkeiten Systeme zu entwickeln, die diese politische Landschaft, in Form von gesammelten Daten, analysieren und dessen Ergebnisse schlussendlich interpretiert werden können. Diese Systeme entspringen dem Bereich *Künstliche Intelligenz* (KI) der Informatik, welcher seit mehreren Jahrzenten stark beforscht wird und weitere Aufgabenbereiche, wie *neuronale Netze* (NN), *deep learning* (DL) und *text classification* umfasst. Für die Beschaffung der benötigten Daten eignet sich die Social Media Plattform Twitter, da sie für die textliche Interaktion zwischen Usern gedacht ist und ursprünglich, anders als Instagram und TikTok, nur Text- und nicht Bild- oder Videodaten unterstützt hat. Dementsprechend beinhaltet der Großteil der Beiträge auch heute noch Text, was dem politischen Dialog der Parteien ähnelt.

1.2 Zielsetzung

Um zu versuchen die angesprochenen Probleme zu lösen, die durch den politischen Dialog und Filterbubbles auf Social Media Plattformen entstehen, soll durch Verwendung von *Natural Language Processing* (NLP) mindestens ein Modell mithilfe NNs trainiert werden, das Tweets deutscher PolitikerInnen zu ihrer Parteiangehörigkeit klassifiziert. Ein Modell im Kontext des zu entwickelnden Produktes dieser Arbeit, ist ein System bzw. ein Algorithmus, der nach Training mit einem Datensatz bestimmte Strukturen und Muster erkennt [Br 2020a]. Allerdings wird auch häufig bereits die Architektur eines Modells als solches bezeichnet. Mithilfe dieses Systems soll ermöglicht werden einen Vergleich eigener Aussagen und Meinungen zur Menge an Tweets deutscher PolitikerInnen aufzustellen, um zu untersuchen, welche den jeweiligen Parteien und PolitikerInnen entsprechen. Somit könnte Nutzenden bei der Entscheidung geholfen werden, welche Partei sie unterstützen möchten, ähnlich wie beim „Wahl-o-Mat“ [BPB 2023]. Nicht nur Privatpersonen, sondern auch PolitikerInnen sowie Parteien wird damit ermöglicht zu überprüfen, ob ihre verfassten Beiträge mit der Ausrichtung der eigenen Partei übereinstimmen, bevor sie veröffentlicht werden. Zwar kann mit solch einem Modell nicht ermittelt werden, ob die Aussagen eines Beitrags wahr sind, allerdings sollte es in der Lage sein, die unterschiedlichen Standpunkte der Parteien zu erfassen und somit eine informierte Aussage über die Resonanz der unterschiedlichen Parteien zu einer Aussage zu treffen.

Um einen größeren Teil des Potentials der Datenmenge auszuschöpfen, können zusätzlich Altersgruppen, Geschlechter und Zeitabschnitte in Betracht gezogen werden. Dadurch können unterschiedliche Systeme trainiert werden, um sie letztendlich miteinander zu vergleichen und zu analysieren, wie gleiche Aussagen in den unterschiedlichen Systemen klassifiziert werden. So könnte durch systematisches Testen ein Verständnis über die Unterschiede der Aussagen und Meinungen der PolitikerInnen der Parteien gewonnen werden.

1.3 Aufbau der Arbeit

Um ein oder mehrere Modelle zu erstellen, die die Klassifizierung vornehmen, wird erstens eine Auswahl der Grundlagen erläutert, die dafür benötigt wird. Anschließend wird anhand dieser ein grobes Konzept entwickelt, durch das die Umsetzung erfolgen soll. Hierbei werden die Gründe erläutert, warum sich für oder gegen die jeweiligen behandelten Methoden, Algorithmen oder Arbeitsabläufe entschieden wird. Bei der Umsetzung im vorletzten Kapitel dieser Arbeit ist zu erwarten, dass einige der festgelegten Methoden in der Praxis angepasst werden müssen, weswegen das ursprüngliche Konzept eventuell nicht exakt übernommen wird. Die Kapitel des Hauptteils werden wesentlich an den Aufbau der Schritte von *Knowledge Discovery in Databases* (KDD) angelehnt, welche in Kapitel 2.2 erläutert werden.

2 Grundlagen

Um ein Modell zu entwickeln, das Tweets klassifiziert, muss dieses ein Verständnis der Trainingsdaten gewinnen. Dieses und weitere Ziele werden im Aufgabenbereich NLP mithilfe von Algorithmen aus dem Bereich *machine learning* (ML), wie *Naive Bayes* und *logistische Regression* und NNs, aus dem Unterbereich DL, angestrebt. Im Bereich NLP soll natürliche, menschliche Sprache für den Computer verständlich gemacht werden, um am Ende eine Kommunikation zwischen Menschen und Maschinen zu ermöglichen [Ml 2020].

In diesem Kapitel werden daher die Grundlagen, die in dieser Arbeit verwendet werden, systematisch anhand der Struktur von KDD erläutert sowie ähnliche Arbeiten aufgezeigt.

2.1 Vergleichbare Arbeiten

Viele Arbeiten, die mit Tweets arbeiten, verwenden *sentiment analysis*, um Tweets auf bestimmte Aspekte zu analysieren. Arbeiten wie „*Classifying political tweets using naïve bayes and support vector machines"* [HaAlRoBi 2018] und „*What's in your tweets? I know who you supported in the UK 2010 general election*" [BoKiYo 2021] die text classification für Tweets verwenden, behandeln meistens englische oder fremdsprachige Tweets. Allerdings verfolgt keine Arbeit das Ziel, Tweets zur Parteizugehörigkeit zu klassifizieren. Dementsprechend kann sich eine Arbeit im Bereich deutscher Sprache, text classification und politische Tweets als Bereicherung für die Forschung herausstellen.

2.2 Knowledge Discovery in Databases

Der Prozess der Verarbeitung der Rohdaten, bis zum Erschließen neuen Wissens über diese, ist durch KDD definiert [Fa 1997]. Es besteht aus mehreren Teilschritten, die durchlaufen werden und in denen jeweils unterschiedliche Methoden zum Erreichen des jeweiligen Ziels Anwendung finden.

KDD umfasst fünf Schritte, die im Idealfall als Endergebnis neues Wissen über die verwendeten Daten entstehen lassen. Als erstes wird aus einer vorhandenen Datenmenge eruiert, welche Teildatensätze für den Anwendungsfall notwendig sind, und der Rest entfernt (siehe Abbildung 1, Selection). Im Schritt der *Vorverarbeitung* (siehe Abbildung 1, Preprocessing) wird die Datenmenge bereinigt, gesäubert, umstrukturiert und durch weitere Methoden bearbeitet, um möglichst viele Störfaktoren zu entfernen. Im Anschluss wird der Datensatz durch *Transformation* (siehe Abbildung 1, Transformation) in ein geeignetes Eingabeformat für den Trainingsalgorithmus gebracht, während weitere Methoden verwendet werden, um Daten zu komprimieren und wichtige *features* aus ihnen zu extrahieren. Mittels *Data-Mining* wird eine Analyse der Daten durchgeführt (siehe Abbildung 1, Data Mining), durch die die unterliegenden Muster der Daten erkannt werden sollen. Abschließend werden die Ergebnisse ausgewertet und dargestellt, wodurch neues Wissen über die Daten entsteht (siehe Abbildung 1, Interpretation/Evaluation). Im Normalfall werden diese Schritte mehrfach durchlaufen, um Optimierungen durchzuführen.

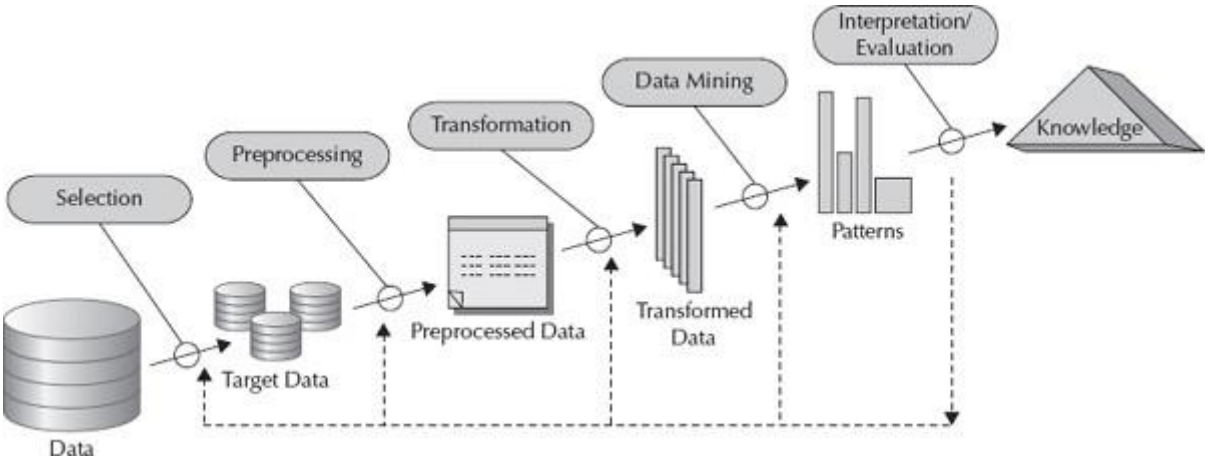

Abbildung 1: Knowledge Discovery in Databases Schritte [Va 2018]

Anhand der KDD-Schritte wird in den folgenden Unterkapiteln eine Auswahl unterschiedlicher Methoden erläutert, die in dem jeweiligen Schritt verwendet werden und für die Entwicklung eines Modells, für die Klassifizierung von Tweets deutscher PolitikerInnen, angewendet werden können. Hierbei ist zu beachten, dass KDD nicht die Auswahl an Methoden definiert, sondern einen Rahmen für mögliche Methoden darstellt, die im jeweiligen Schritt Verwendung finden. Die Methoden variieren ebenfalls je nach Anwendungsfall. KDD wird somit in diesem Kapitel sowie im weiteren Verlauf der Arbeit zur Einordnung der Methoden verwendet. Da allerdings nicht für jeden Schritt Grundlagen benötigt werden, sind in diesem Kapitel nicht alle KDD-Teilschritte vorhanden.

2.3 Datenvorverarbeitung

2.3.1 Tokenizing

Mittels *tokenizing* werden Textdaten in Bruchteile bzw. *Tokens* unterteilt. Hierbei wird ein Text nach vordefinierten Trennzeichen durchsucht oder ein bereits trainiertes Modell angewendet, um die Tokens zu erstellen. Dies ist wichtig für die Anwendung weiterer Methoden, wie *PoS-tagging*, da der Text bereits granular vorliegt, aber auch für die Eingabe der Trainingsdaten in den Klassifizierungsalgorithmus, der im Modell verwendet wird.

Es existieren unterschiedliche Arten des tokenizing. *Word tokenizing* unterteilt die Wörter eines Satzes. Die Tokens, die während des Trainings des Algorithmus vorkommen, werden zu einem Vokabular hinzugefügt und mit einem Index versehen, da der Trainingsalgorithmus meistens eine numerische Eingabe verlangt. Ein Problem, das sich aus dieser Methode ergibt, ist der Umgang mit unbekannten Tokens. Dies kann passieren, wenn nach Abschluss des Trainings Textdaten außerhalb der Trainingsdaten verwendet werden, die unbekannte Tokens bzw. Wörter beinhalten. Sie werden als *out of vocabulary* (OOV) Wörter bezeichnet. Eine mögliche Lösung hierfür ist, OOV-Wörter mit einem spezifischen Token zu ersetzen, der für alle OOV-Wörter verwendet wird. Dadurch entstehen allerdings weitere Probleme, da alle OOV-Wörter die gleiche Darstellung erhalten, obwohl sie vorher eventuell unterschiedlich

oder gleich gewesen sind. Somit gehen Informationen über diese Wörter verloren, die allerdings durch die Verwendung von *character tokenizing* erhalten bleiben [Bu 2022].

Durch character tokenizing werden keine Tokens durch Wörter erstellt, sondern anhand der Zeichen eines Wortes. Dadurch besteht das Vokabular nur aus der Anzahl aller Buchstaben und Sonderzeichen des jeweiligen Alphabets, wodurch Informationen über OOV-Wörter erhalten bleiben, statt sie durch spezielle Tokens zu substituieren. Allerdings steigt hierdurch die Anzahl der Tokens pro Eingabe maßgeblich und die Zusammenhänge sowie die Bedeutungen der ehemaligen Wörter gehen großteilig verloren [Bu 2022]. Um das OOV-Problem zu lösen, existiert eine Kombination aus *character* und *word tokenizing*, namentlich *sub*-word tokenizing, die in Kapitel 2.4.2.3 erläutert wird.

2.3.2 Ausbalancieren des Datensatzes

Da ein Datensatz eventuell unausgeglichen ist, was bedeutet, dass die Klassen untereinander ungleiche Verhältnisse an Daten aufweisen, gibt es Methoden wie under- und oversampling, um die Verteilung der Daten pro Klasse anzugleichen. Diese sind nötig, da die Vorhersagen des trainierten Modells ansonsten zur dominierenden Klasse tendieren könnten.

Undersampling beschreibt das zufällige Entfernen von Daten der jeweiligen Klassen, bis sie die gleiche oder eine ähnliche Menge an Daten ausmachen wie die Minderheitenklasse. Oversampling repliziert stattdessen zufällig Daten, bis alle Klassen die gleiche oder eine ähnliche Menge an Daten aufweisen, wie die dominierende Klasse [Br 2021a].

Beide Methoden haben allerdings Vor- und Nachteile. Durch undersampling gehen wichtige Informationen durch das Löschen von Daten verloren, die für die Generalisierfähigkeit des Modells hilfreich sein könnten. Da es sich bei den Daten um Tweets handelt, könnten außerdem zufällig hauptsächlich neue Tweets entfernt werden, wodurch der zeitliche Bereich, den das Modell abdecken kann, verkleinert wird. Allerdings sinkt die Trainingsdauer durch den Verlust der Daten, was je nach Anwendungsfall vorteilhaft sein könnte. Durch oversampling gehen zwar keine Daten verloren, allerdings kann *overfitting* entstehen, falls das Modell die mehrfach auftretenden Daten erkennt. Dieses Problem kann jedoch im Idealfall durch die *Synthetic Minority Oversampling Technique* (SMOTE) [ChBoHaKe 2002] gelöst werden. Hierbei werden keine Daten repliziert, sondern neue Daten, durch Interpolation der Daten der Minderheitsklassen, synthetisch hergestellt. Somit unterscheiden sie sich minimal voneinander, wodurch overfitting verhindert werden kann. Das Verwenden von oversampling bedeutet allerdings auch, dass die Datenmenge und somit die Trainingsdauer ansteigt.

2.4 Datentransformation

2.4.1 Feature extraction

Durch *feature extraction* werden die betrachteten Informationen der Daten in reduzierter, komprimierter und bei *text mining* meist numerischer Form dargestellt. Ein feature beschreibt dabei eine messbare Eigenschaft der vorhandenen Daten [DataRobot 2022]. Hierbei gehen im Idealfall wenig bis keine Informationen über die originalen Daten verloren [Sa 2016; Ms 2021].

Eine hierfür häufig verwendete Methode ist *Bag of Words* (BoW). Sie wird meistens bei *document-* oder text classification eingesetzt und analysiert die Frequenz von Wörtern eines Textes. Textdaten werden zu einer Menge aller einzigartigen Wörter des jeweiligen Texts oder Dokuments reduziert, wodurch ein Vokabular entsteht. Zusätzlich wird die Frequenz der Wörter aufgrund der Theorie ermittelt, dass Texte ähnlich sind, wenn sie vergleichbare Wortfrequenzen haben [Br 2017a].

Eine Repräsentationsmöglichkeit der daraus resultierenden features stellt das *Vector Space Model* (VSM) [SaWoYa 1975] dar. Hierbei handelt es sich um eine mathematische Repräsentation von features als Vektoren [Ta 2021a]. Dadurch kann mit arithmetischen Operationen, wie der *Kosinus-Ähnlichkeit*, die Ähnlichkeit der Vektoren, die aus den Texten resultieren, in Form der Abstände dieser ermittelt werden. Um die dafür benötigten Vektoren zu erstellen, wird das durch BoW erstellte Vokabular mittels *one-hot encoding* (siehe Abbildung 2) dargestellt. Dabei wird jedes Wort zu einem einzigartigen Nullvektor mit einer eins umgewandelt. Somit würde der Satz „the cat sat on the mat“ zum Vektor [2, 1, 1, 1, 1], wobei jede Stelle des Vektors zu dem jeweiligen Wort im Vokabular zugeordnet werden kann und die Frequenz des Wortes durch die Zahl an der jeweiligen Stelle dargestellt wird. Hierbei ist die Länge des Vektors von der Länge des Vokabulars abhängig.

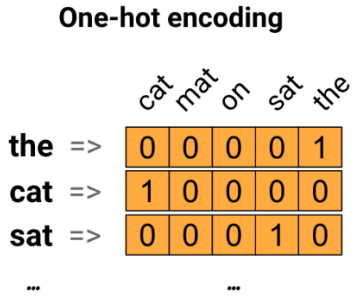

Abbildung 2: One-hot encoding Vektor [TF o.D.]

Dadurch entstehen allerdings lange und *spärliche Vektoren*, da sie eine geringe Anzahl an nicht-Null Werten beinhalten [IBM 2021] und stets die Länge des Vokabulars besitzen, was bei einem großen Vokabular von mehreren tausend Wörtern die Effizienz des Trainings reduziert [Pa 2021]. Eine weitere Möglichkeit Worte in numerische Form zu bringen ist *integer encoding* (siehe Abbildung 3).

I	ate	an	apple	and	played	the	piano
1	2	3	4	5	6	7	8

Abbildung 3: Integer encoding [Pa 2021]

Durch die Nummerierung anhand des Auftretens der Wörter in einem Text entstehen zwar dichtere Vektoren, allerdings gehen hierbei sowie beim one-hot encoding Informationen des Kontexts verloren, in dem die Wörter auftauchen [AIML 2017]. Um stattdessen die Bedeutung von Wörtern zu erfassen, ohne ausschließlich auf die Frequenz zu achten, können die Vektoren dieser Wörter durch w*ord embeddings* repräsentiert und im VSM dargestellt werden.

2.4.2 Word embeddings

Word embeddings stellen Wörter mit ähnlicher Bedeutung als ähnliche Vektoren dar. Diese Ähnlichkeit wird anhand des Kontexts ermittelt, in dem sie auftreten. So können auch unterschiedliche Wörter, die im gleichen Kontext verwendet werden, anders als bei der Verwendung von BoW, einer ähnlichen Bedeutung zugeordnet werden [Br 2017b]. Einige der bekanntesten word embedding Algorithmen sind *Word2Vec* [MiChCoDe 2013], *GloVe* [PeSoMa 2014] und *fastText* [BoGrJoMi 2017]. Bei ihnen handelt es sich um statische word embeddings, da sich das embedding nach dem Trainieren nicht mehr verändert und die Repräsentation eines Wortes somit in jedem Kontext gleichbleibt. Diese Algorithmen sind bereits in trainierter Form verfügbar und müssen im Idealfall nicht angepasst werden, weswegen hierfür keine weitere Trainingszeit anfällt.

2.4.2.1 Word2Vec

Word2Vec erstellt die embeddings, indem entweder das *Continuous Bag of Words* (CBOW) [MiChCoDe 2013, Seite 4] oder das *Skip-Gram* (SG) Modell [MiChCoDe 2013, Seiten 4-5] verwendet wird. Durch CBOW wird der Vektor oder das embedding eines Wortes durch den Kontext angrenzender Wörter erlernt. Als Eingabewerte erhält das CBOW-Modell eine durch die *Fenstergröße* festgelegte Anzahl an Vektoren, die die Eingabesequenz darstellen. Sollte es noch keine Vektorrepräsentation für ein Wort geben, wird dessen Vektor zufällig initialisiert. Bei einer Fenstergröße $x_F = 1$ und dem Satz „I like eating delicious apples“ würde das vorherige Wort „like“ und das nachfolgende Wort „delicious“ dazu verwendet werden, das mittlere Wort „eating“ zu projizieren. Dies wird durch die Summierung der Vektorrepräsentationen der umliegenden Wörter erreicht, wie in Abbildung 4 erkennbar ist. Durch das Verwenden des Mittelwerts der umliegenden Wörter haben ihre Positionen keinen Einfluss auf die Projektion. So könnten die Wörter vor und hinter dem vorherzusagenden Wort vertauscht sein, ohne das Ergebnis zu verändern.

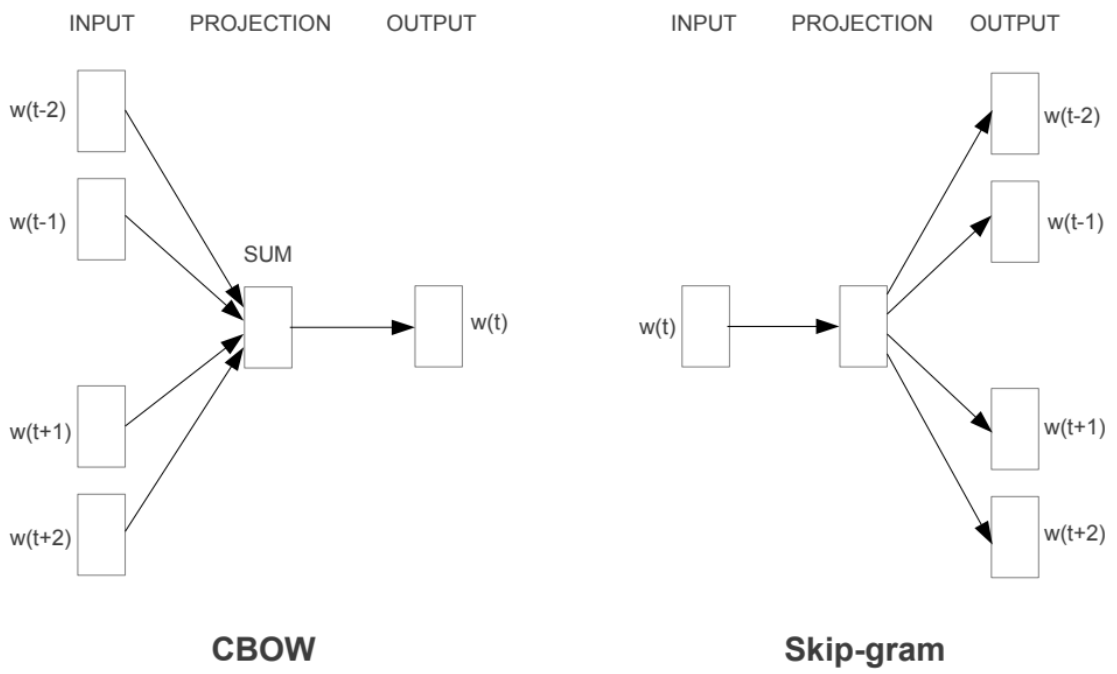

Abbildung 4: Architekturen der CBOW- und SG-Modelle [MiChCoDe 2013, Seite 5]

Im Gegenzug zu CBOW verwendet SG nur ein Wort als Eingabe und erlernt die word embeddings durch die Vorhersage der durch die Fenstergröße festgelegten benachbarten Wörter (siehe Abbildung 4). Da Wörter, die weit vom Eingabewort entfernt liegen, häufig weniger ausschlaggebende Informationen über die Beziehung zu diesem liefern, werden sie bei SG mit Gewichten behaftet, um dieses Verhalten widerzuspiegeln und ihren Einfluss auf das Ergebnis zu reduzieren [MiChCoDe 2013, Seite 4].

2.4.2.2 GloVe

Statt lediglich den lokalen Kontext in Form von benachbarten Wörtern zu verwenden, wird durch GloVe eine globale *co-occurence* Matrix erstellt. Jedes Matrixelement gibt die Häufigkeit an, zu der ein Wort, in Kombination mit einem anderen, im betrachteten Textkorpus vorkommt. Hieraus kann die Wahrscheinlichkeit des gemeinsamen Auftretens dieser Wörter ermittelt werden. Damit selten und häufig auftretende Wörter nicht zu viel Einfluss auf die Ergebnisse erhalten, werden sie mit einer Gewichtungsfunktion anhand ihrer co-occurence Wahrscheinlichkeit faktorisiert.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Abbildung 5: Beispiel einer GloVe co-occurence Matrix [PeSoMa 2014, Seite 3]

Die Wahrscheinlichkeiten aus Abbildung 5, zu der ein Wort j im Kontext von Wort k vorkommt, sind durch $P_{kj} = P(j|k) = X_{kj}/X_k$ definiert. Hierbei steht X_{kj} für die Anzahl, zu der Wort j im selben Kontext wie Wort k steht und X_k für die Anzahl, zu der Wort k im gesamten Korpus vorkommt. Anhand dieser Berechnungen wird die *loss-function* erstellt, um das Trainieren des Modells zu ermöglichen, welches das word embedding generiert.

2.4.2.3 FastText

Ein weiteres word embedding Modell, das auf Word2Vec bzw. SG basiert, ist fastText. Es stellt nicht wie Word2Vec und GloVe lediglich Wörter als Vektoren dar, sondern verwendet zusätzlich die Informationen von sub-words. FastText setzt dafür einen eigenen tokenizer ein, der das sub-word tokenizing vornimmt. Hierbei werden *n-grams* verwendet, um ein Wort in jegliche Kombinationen an n Buchstaben zu unterteilen. Aus der Summe der Vektoren der einzelnen n-grams für ein Wort ergibt sich wieder der ursprüngliche Wortvektor. Mit Hilfe dieser Informationen können ähnliche Wörter von Sprachen mit starker Morphologie, wie Deutsch, besser repräsentiert werden. Die Vektoren für Wörter wie „Apfelbaum“ und „Apfel“ würden in der Theorie einen geringeren Abstand haben als in anderen embeddings mit word tokenizing, da ihre n-grams für das Teilwort „Apfel“ überlappen und somit eine hohe morphologische Ähnlichkeit aufweisen [Ga 2021]. Durch das Verwenden von sub-word tokenizing können außerdem OOV-Wörter repräsentiert werden. Hierfür werden bereits bestehende Repräsentationen der n-grams des unbekannten Wortes verwendet und ein Mittelwert dieser gebildet oder andernfalls zufällig initialisiert.

2.4.3 Language Models

Einen anderen Ansatz embeddings zu erstellen, bieten Language Models (LMs) wie BERT [DeChLeTo 2018], RoBERTa [LiOtGoDu 2019] und XLNet [YaDaYaCaSaLe 2019]. Sie basieren auf der Transformer-Architektur [VaShPaUsJoGoKaiPo 2017], die eine Art NN ist, das mit *self-attention* (SA) Mechanismen Eingabesequenzen verarbeitet. Diese LMs können, wie NNs, auch für unterschiedliche NLP-Aufgaben wie sentiment analysis und text classification genutzt werden. Die dafür erlernten embeddings können von den trainierten LMs extrahiert und ebenso wie die statischen word embeddings verwendet werden. Allerdings erstellen die erwähnten LMs dynamische Vektorrepräsentationen. Dies bedeutet, dass gleiche Wörter in unterschiedlichem Kontext zu unterschiedlichen Vektoren werden, um semantische Informationen besser abzubilden [Me 2021]. Um dies zu erreichen, muss das jeweilige LM vor den Input des Klassifizierungsalgorithmus des Modells geschaltet werden. Somit kann das LM je nach Eingabesequenz die word embeddings anhand des Kontexts anpassen und diese dem Lernalgorithmus als Input übergeben. Hieraus resultiert allerdings eine höhere Trainingsdauer.

2.4.3.1 Transformer-Architektur

Ein Transformer besteht aus zwei Teilen, Encoder und Decoder [VaShPaUsJoGoKaiPo 2017, Seite 3]. Der Encoder stellt die gelernte Repräsentation der Eingabesequenz dar und der Decoder verwendet diese Informationen, um einen Output zu generieren. Durch SA, das von *attention heads* berechnet wird, besitzt der Transformer ein stark ausgeprägtes Langzeitgedächtnis und kann ermitteln, welche Worte der Eingabesequenz von größter Bedeutung für den Output des Modells sind, um diesen eine entsprechende Gewichtung zu erteilen. Durch den SA-Mechanismus besitzt der Transformer ein großes Fenster, in dem eine Eingabesequenz gespeichert werden kann. Somit kann ein Transformer auch frühere Teile der Eingabe für die Berechnung des Outputs verwenden. Über diese Menge berechnet der Encoder mit SA, welche Wörter am meisten *attention* benötigen [Ph 2021]. Durch die Berechnung kann der Decoder ein passendes Wort, das auf die Eingabesequenz folgt, als Output generieren. Danach verwendet ein *Feed-Forward Neural Network* die Ergebnisse von SA, um eine potenziell genauere Repräsentation der berechneten attention als Output zu berechnen [VaShPaUsJoGoKaiPo 2017, Seite 5; Ph 2021]. Hierbei wird eine *softmax* Funktion zur Normalisierung der Werte verwendet. Sie verringert bereits kleine und erhöht hohe Werte, was dazu führt, dass Wörter mit hoher *attention-score* noch stärker fokussiert werden und erst im Anschluss Wörter mit geringeren attention-*scores*. Der Output des Decoders wird wieder als Input für die Generierung des nächsten Outputs verwendet, um auch diese Informationen in die Berechnung zu inkludieren, womit es sich um eine sequenzielle Berechnung handelt [VaShPaUsJoGoKaiPo 2017, Seite 3; Ph 2021].

2.4.3.2 BERT

Das *Bidirectional Encoder Representations from Transformers* (BERT) [DeChLeTo 2018] Modell stellt eine weiterentwickelte Form der Transformer-Architektur dar. Insgesamt existieren zwei unterschiedliche BERT-Modelle, BERT-Base und BERT-Large, die sich in der Größe des Netzwerks unterscheiden, da eine höhere Anzahl trainierbarer Parameter einen positiven Einfluss auf die Genauigkeit des Modells hat [DeChLeTo 2018, Seite 8].

Grundsätzlich wird das embedding mit BERT durch den Einsatz von zwei Aufgabentypen im Bereich *unsupervised learning* erlernt, was eine von mehreren Lernarten im Themengebiet machine learning ist, die auch bei DL verwendet werden kann. Hiermit erlernt das System ein Verhalten ohne Hinzugabe des richtigen Ergebnisses bzw. Labels, anhand dessen überprüft werden kann, ob die Vorhersage korrekt ist. Konträr dazu existiert *supervised learning*, wobei das System durch die Hinzugabe des korrekten Ergebnisses trainiert wird [Co 2022].

Die erste Aufgabe von BERT, im Bereich unsupervised learning, lautet *Masked Language Modeling* (MLM). Die Eingaben, die von BERT verarbeitet werden sollen, werden zuvor mit maskierten Wörtern versehen, die durch BERT anhand des Kontextes der anderen Wörter erlernt und vorhergesagt werden. Diese Wörter werden mit einem „[MASK]“ Token ersetzt. Standardmäßig sagt BERT in einem Satz

15 % der Wörter vorher, von denen 80 % maskiert, 10 % mit zufälligen Vektorrepräsentationen eines Wortes ersetzt werden und 10 % unverändert bleiben [DeChLeTo 2018, Seite 4]. Die zweite Aufgabe ist *Next Sentence Prediction* (NSP) und besteht daraus, zwei Eingabesätze getrennt mit einem „[SEP]“ Token miteinander zu vergleichen und vorherzusagen, ob einer auf den anderen folgt oder es sich um einen zufälligen Satz aus dem Korpus handelt. Somit soll BERT die Zusammenhänge zwischen Sätzen verstehen und den bidirektionalen Kontext von Wörtern simultan erlernen, statt wie bei der ursprünglichen Transformer-Architektur sequenziell [DeChLeTo 2018, Seiten 4-5]. BERT verwendet sub-word tokenizing, weswegen OOV-Wörter wie auch bei fastText dargestellt werden können.

2.4.3.3 RoBERTa

Robustly optimized BERT approach (RoBERTa) [LiOtGoDu 2019] ist eine Variante von BERT, mit dem Ziel, die Genauigkeit von BERT zu erhöhen. Dies wird durch das Trainieren mit einem größeren Datensatz von 160 GB erreicht [LiOtGoDu 2019, Seite 3]. Der Trainingsdatensatz von BERT umfasst im Vergleich lediglich 16 GB. Zusätzlich werden längere Eingabesequenzen und Trainingszeiten verwendet [LiOtGoDu 2019, Seite 3]. Ebenso wie BERT verwendet RoBERTa MLM zum Trainieren, allerdings wird der Ansatz von NSP verworfen [LiOtGoDu 2019, Seite 5] und die Art der Maskierung von *static* zu *dynamic masking* geändert. Mit static masking werden die zu maskierenden Worte initial und einmalig gesetzt, wodurch sie jede Trainingsepoche gleichbleiben. Durch dynamic masking werden die Trainingsdaten zehnmal dupliziert, um unterschiedliche Maskierungen derselben Daten zu ermöglichen. Dies trainiert das Modell im Umkehrschluss darauf, möglichst unterschiedliche Wörter derselben Sequenz vorhersagen zu können [LiOtGoDu 2019, Seite 4]. Zusätzlich hat der verwendete Transformer mehr Schichten und *attention heads*, wodurch dem Modell mehr Kapazitäten zur Verfügung stehen, mit denen es die Leistung von BERT übertreffen kann [LiOtGoDu 2019, Seite 6].

2.4.3.4 XLNet

Ähnlich wie RoBERTa erhöht XLNet [YaDaYaCaSaLe 2019] die Menge an Trainingsdaten, verwendet höhere Computerleistung, aber behält NSP. Da BERT die maskierten Wörter parallel vorhersagt, kann es keine Zusammenhänge zwischen diesen herstellen. Aus diesem Grund wird mit XLNet der MLM-Task angepasst, indem nicht nur 15 % der Tokens vorhergesagt werden. Stattdessen werden alle Worte sequenziell mit Tokens in zufälliger Reihenfolge und anhand der Permutationen des Eingabesatzes ausgetauscht. Somit werden die Zusammenhänge zwischen allen Tokens besser erlernt [YaDaYaCaSaLe 2019, Seite 3]. Außerdem wird die *Transformer-eXtended Length* Architektur verwendet, bei der es sich um eine Variante der Transformer-Architektur handelt, die größere Eingabesequenzen verarbeiten kann [DaYaYaCaLeSa 2019]. Zusätzlich wird mit ihr eine neue Komponente *memory* eingeführt, durch die Informationen aus früheren Segmenten der Eingabesequenz abgespeichert werden können, um sie im späteren Verlauf der Bearbeitung wiederzuverwenden. Somit kann das Modell die Abhängigkeiten von

längeren Eingabesequenzen auch bei geringerer Rechenleistung besser erlernen [DaYaYaCaLeSa 2019, Seiten 1-2].

2.5 Data-Mining

Um Texte klassifizieren zu können, wird ein Algorithmus verwendet, der die Vorhersage für die jeweilige Eingabe tätigt. Dieser wird im KDD-Teilschritt Data-Mining (siehe Abbildung 1, Data Mining) benötigt, um die Architektur des Modells zu entwickeln. Hierbei kann grundsätzlich zwischen ML-Methoden wie Naive Bayes, Support Vector Machines, Linearer Regression sowie DL-Algorithmen wie neuronale Netzwerke unterschieden werden. Statistische Modelle aus ML sind im Vergleich zu NNs einfacher zu implementieren, da sie weniger Parameter haben, die angepasst werden können, um die Qualität der Vorhersage zu verbessern. Zusätzlich müssen sie tendenziell nicht im selben Umfang trainiert werden, wie die Gewichte an den Neuronen eines NN, was die Veränderung von Parametern des Algorithmus und das daraus resultierende erneute Trainieren vereinfacht [Dl 2022]. Andererseits können NNs anhand ihrer höheren Kapazität, die aus ihren Architekturen resultiert, auch mit komplexeren Datensätzen genauere Ergebnisse erzielen [Mo o.D.].

Es gibt bereits eine vielfältige Auswahl unterschiedlichster Arten NNs, die jeweils für bestimmte Aufgabentypen, im Vergleich zu anderen, besser geeignet sind. Die meisten dieser Netze basieren auf einer Handvoll stark beforschter Algorithmen, die sich von anderen absetzen, da sie ihre Aufgaben besonders gut erfüllen können. Darunter fallen für Textklassifizierung *Convolutional Neural Networks* (CNNs), *Recurrent Neural Networks* (RNNs), *Gated Recurrent Units* (GRUs) und *Long Short-Term Memory Neural Networks* (LSTMs) [Mo o.D.; Br 2022].

2.5.1 Recurrent Neural Network

RNNs [RuHiWi 1986] sind entwickelt worden, um Eingabesequenzen verarbeiten zu können, während Informationen von vorherigen Eingaben bei zukünftigen Berechnungen verwendet werden. Somit werden die Eingaben nicht unabhängig voneinander betrachtet, sondern mit Wissen der Vorherigen. Die Architektur des RNNs verwendet *loops*, um die Ergebnisse vorheriger Berechnungen bei der Aktuellen zu verwenden. Anhand Abbildung 6 ist zu erkennen, dass das Ergebnis der *hidden-layer* der nächsten Schicht im Netz übergeben wird, allerdings auch der Schicht selbst als Eingabe, zusätzlich zum nächsten Output der *Input-layer*. Dadurch werden Eingabesequenzen sequenziell bearbeitet.

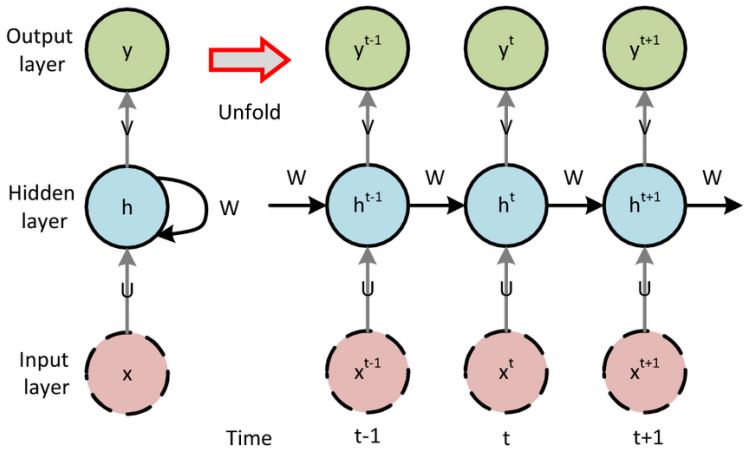

Abbildung 6: Architektur eines RNNs [ZaPaKa 2021]

Eine große Schwachstelle von RNNs stellt dessen schlechtes *Short-Term Memory* (STM) dar, das sich durch das *vanishing gradient* Problem äußert. Es tritt durch *backpropagation* beim Training des RNNs auf und verursacht, dass dem RNN pro Eingabe, die es in einer Sequenz verarbeitet, weniger Informationen von den vorherigen Inputs erhalten bleiben. Je weiter die Schicht von der Output-layer entfernt liegt, desto geringer die Anpassungen der Gewichte und umso weniger können diese Schichten lernen. Dies liegt an der Multiplikation des Gradienten pro Schicht, in Kombination mit der Verwendung einer sigmoid oder tanh Aktivierungsfunktion an den Neuronen, die üblicherweise in RNNs verwendet werden. Der Gradient beinhaltet die Informationen über den Einfluss von jedem Neuron im neuronalen Netzwerk auf die Veränderung der loss-function. Die Aktivierungsfunktionen sorgen dafür, dass der Output in einen Zahlenbereich von jeweils -1 bis 1 und 0 bis 1 normalisiert wird. Dies führt im Umkehrschluss dazu, dass eine große Veränderung am Input eine kleine Auswirkung auf den Output hat. Da der Gradient pro Schicht in Richtung Eingabeschicht mit den Gradienten der Neuronen vorheriger Schichten multipliziert wird, verringert sich der Gradient pro Schicht durch die Multiplikation von normalisierten Outputs kleiner eins [Gu 2021; Hi 2022].

Um die Auswirkungen des STMs bzw. des vanishing gradients zu verringern, sind LSTMs und GRUs entwickelt worden, die auf RNNs basieren.

2.5.2 Convolutional Neural Network

CNNs [Lc 1998] sind für den Bereich *Computer Vision* entwickelt worden, um Eigenschaften von Bildern zu erlernen. Insbesondere bei Bildklassifikation können CNNs deshalb im Vergleich zu anderen Algorithmen hohe Genauigkeiten erzielen, da sie mittels Methoden wie convolution und pooling die Muster und Strukturen eines Bildes besonders gut erkennen. Zusätzlich werden die Berechnungen des CNNs auf der Grafikkarte des Computers ausgeführt, statt auf dem Prozessor, was den Trainingsprozess beschleunigt. Allerdings werden CNNs ebenfalls im Bereich text classification verwendet, was auf ihre Fähigkeit zurückzuführen ist, Muster und Strukturen in Daten besonders gut zu erkennen.

2.5.2.1 Convolution

Durch die convolutional-layer (CL) wird eine Art feature extraction der numerischen Daten vorgenommen. So werden auf die als Matrix vorliegenden Pixeldaten der Bilder Filter angewendet, die ebenfalls Matrizen mit zuvor festgelegter Größe $m * m$ sind. Die Filter werden über die Matrix gelegt und multiplizieren den aktuellen Pixelwert mit dem korrespondierenden Wert im Filter und summieren im Anschluss die Ergebnisse. Danach werden die Filter einen *stride* weitergeschoben, bis sie alle Werte multipliziert haben. Das Ergebnis der Konvolution pro Filter ist eine *feature map* bzw. eine Matrix mit geringerer Dimension, die der nächsten Schicht übergeben wird. Wenn die Verringerung der Dimension unerwünscht ist, kann es mit padding, was das Einfügen von Nullwerten um die Matrix herum beschreibt, verhindert werden. Somit gehen keine Randinformationen verloren und der globale Kontext wird beibehalten. Andernfalls wird der Fokus auf die wichtigsten Eigenschaften der Eingabewerte gelegt und somit der lokale Kontext besser extrahiert. Durch das Verwenden mehrerer Filter mit unterschiedlicher Initialisierung der Gewichte wird ermöglicht, dass pro Filter unterschiedliche Aspekte und Muster der Daten erlernt werden, da die Änderungen der Gewichte unabhängig von anderen Filtern vorgenommen werden [Vo 2023]. Standardmäßig verwenden CLs die ReLu Aktivierungsfunktion, um Probleme wie vanishing gradient zu umgehen und eine höhere Generalisierfähigkeit des Modells zu ermöglichen, da, durch das Ausschalten aller Werte kleiner null, viele Neuronen ausgeschaltet und nur die wichtigsten erlernten features verwendet werden [Br 2020b].

2.5.2.2 Pooling

Die feature maps der CLs werden im Anschluss mit einem weiteren Filter versehen, der ebenso eine Matrix der Größe $m * m$ ist und das pooling durchführt. Die unterschiedlichen pooling Arten sind max-pooling und average-pooling. Statt die Pixelwerte zu multiplizieren, ermittelt max-pooling den höchsten und average-pooling den Durchschnitt der Werte pro stride. Durch diese Operationen wird die Anzahl der Werte reduziert, was das Trainieren des Netzwerks effizienter gestaltet, während die wichtigsten Informationen der jeweiligen Bereiche beibehalten werden [Wa 2020].

3 Konzeption anhand KDD

Dieses Kapitel befasst sich mit dem Konzept für die Entwicklung des Klassifizierungsalgorithmus und ist, wie das zweite Kapitel, anhand der KDD-Schritte gegliedert, um eine klare Einordnung in die unterschiedlichen Teilbereiche zu ermöglichen. Für jeden Teilschritt werden die gewählten Methoden für die Entwicklung des Produkts erläutert und begründet.

3.1 Programmierumgebung

Die einzelnen Schritte und das Trainieren eines Modells werden mit der Programmiersprache Python vollzogen. Sie ist mathematisch ausgelegt und die meisten der benötigten Bibliotheken für die Programmierung von KI-Systemen sind speziell für Python entwickelt worden. Zusätzlich wird für das Erstellen und Trainieren der Modelle das Framework TensorFlow [TF 2023a] verwendet. Ein alternatives Framework hierfür ist PyTorch [Pt 2023], welches sich, im Gegensatz zu TensorFlow, durch ein effizienteres Training von Modellen auszeichnet. Allerdings hat TensorFlow eine ausgeprägtere Dokumentation und bietet mehr Möglichkeiten für das Bereitstellen von Modellen für den produktiven Bereich, weswegen sich hierfür entschieden wird.

3.2 Datensammlung

Das Unternehmen dpa-infocom pflegt eine Sammlung an Politikerdaten, die in Form einer Datenbank strukturiert vorliegt. Sie bietet sich dementsprechend an, für das Entwickeln des Modells verwendet zu werden. Diese Daten umfassen Informationen wie Alter, Parteizugehörigkeit, Name und auch Social Media Accounts wie Twitter, Instagram und Facebook der jeweiligen PolitikerIn. Zu diesem Zeitpunkt beinhaltet sie 5210 deutsche PolitikerInnen.

Die Datenbank, in Kombination mit dem Twitter *Application Programming Interface* (API), ermöglicht ein automatisiertes Auslesen der Tweets deutscher PolitikerInnen, um mit ihnen einen Klassifikationsalgorithmus zu trainieren. Es ist auch möglich diese Daten manuell zu sammeln. Da allerdings für das Entwickeln des Modells im Idealfall eine große Datenmenge vorhanden sein sollte, um hohe Genauigkeiten der Vorhersagen zu erzielen, würde dies einen zu hohen Zeitaufwand bedeuten.

Da das Ziel der Arbeit darin besteht, Tweets von PolitikerInnen deutscher Parteien zur jeweiligen Parteizugehörigkeit zu klassifizieren, ist dafür nicht nur insgesamt eine große Datenmenge von Nöten, sondern auch eine ausgewogene Verteilung der Daten pro Partei. Dadurch fallen viele der kleinen Parteien heraus, da sie meistens aus weniger Mitgliedern bestehen bzw. wenige dieser Mitglieder einen Eintrag in der Datenbank haben und von ihnen dementsprechend nur eine geringe Menge an Tweets gesammelt werden kann.

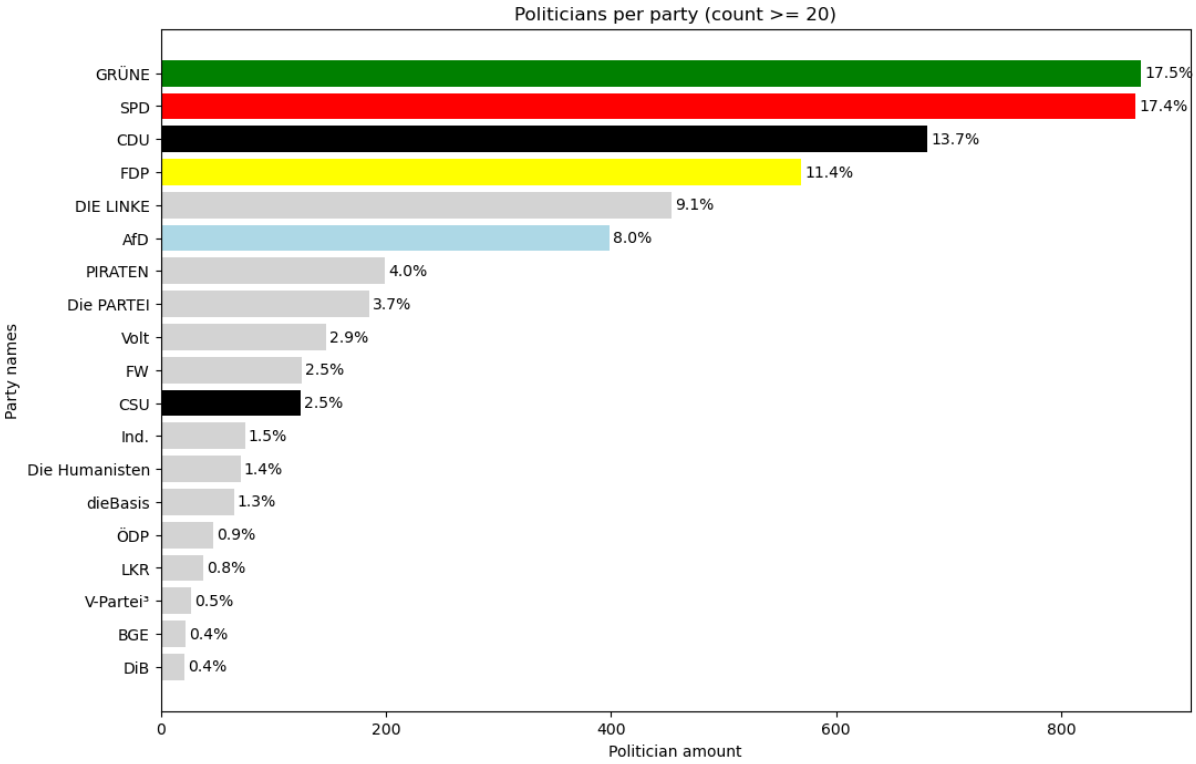

Abbildung 7: Prozentuale Verteilung von PolitikerInnen pro Partei in der Datenbank [eigene Darstellung]

Anhand Abbildung 7 ist die prozentuale Verteilung an PolitikerInnen pro Partei innerhalb der Datenbank dargestellt. Die CDU/CSU, Grünen, SPD, AfD, LINKE und FDP setzen sich von den restlichen Parteien ab. Die Partei DIE LINKE besitzt zwar mehr PolitikerInnen in der Datenbank als die AfD, allerdings nicht im Bundestag. Dort haben sie die Hälfte der Sitze der AfD. Zusätzlich haben sie die fünf Prozent Hürde nicht erreicht und sind durch die Grundmandatsklausel als Fraktion in den Bundestag eingezogen. Aus diesen Gründen wird sich gegen die Verwendung von Tweets von PolitikerInnen der Partei DIE LINKE entschieden.

Um Tweets und deren Metadaten von der Twitter API abzufragen gibt es Python Bibliotheken, wie Tweepy [Tp 2023] und snscrape [JAA 2023]. Tweepy legt den Fokus spezifisch auf die Plattform Twitter, wobei snscrape viele Methoden für unterschiedliche Social Media Webseiten anbietet. Für die Datensammlung benötigt Tweepy eine Authentifizierung bei der Twitter API, wofür ein Account im Twitter Entwicklerportal notwendig ist. Nach Authentifizierung können mittels Tweepy bis zu 3200 Tweets pro Twitter Account abgefragt werden. Allerdings gibt es ein Abfragelimit von 900 Tweets pro 15 Minuten.

Im Gegensatz zu Tweepy benötigt snscrape keine Authentifizierung, da es nicht über die API, sondern direkt über die Website Daten ausliest. Somit kann snscrape das Limit von 3200 Tweets pro Account sowie das Abfragelimit umgehen. Dementsprechend erleichtert snscrape den Programmieraufwand deutlich, da nicht wie bei Tweepy auf das Abfragelimit geachtet werden muss. Außerdem liefert snscrape mehr Metadaten der Tweets und deren Autor, wie die Anzahl an Likes eines Tweets oder die Anzahl an Followern des Autors.

Diese zusätzlichen Metadaten können sich nützlich erweisen, um die Auswirkung von Tweets auf das Modell von einem Account mit wenig Reichweite durch Gewichtung anzupassen, da diese unter Umständen weniger repräsentativ für die Meinungen und Aussagen sind, die die Partei vertritt. Da das Ziel der Arbeit allerdings darin liegt, das Problem der Filterbubbles zu bekämpfen, um den Nutzenden einen besseren Überblick der politischen Landschaft zu bieten, stellt sich hier die Frage, ob es unter dieser Prämisse gerechtfertigt ist, Tweets eine Wichtigkeit zu erteilen, wie es ein Algorithmus einer Social Media Website machen würde. Da die Algorithmen der jeweiligen Plattform nicht öffentlich sind und somit kein Verständnis über sie gewonnen werden kann, ist es nicht möglich dies einzuschätzen. Dementsprechend wird die Gewichtung nicht verwendet.

Ein Nachteil an snscrape ist allerdings die benötigte Zeit für das Abfragen der Tweets. Um die gleiche Menge an Tweets zu erhalten, braucht snscrape ungefähr drei Mal länger als Tweepy. Da mit snscrape aber die maximale Menge an Tweets ausgelesen werden kann und dies nur einmalig geschehen muss, wird für die Datensammlung snscrape verwendet.

Alle Bearbeitungen, die an den gesammelten Daten vorgenommen werden, werden mit der Python Bibliothek Pandas [Pa 2023] durchgeführt. Mit ihr können CSV-Daten ausgelesen, iteriert und mittels unterschiedlicher Funktionen bearbeitet werden.

3.3 Datenselektion

Beim Selektieren der Daten (siehe Abbildung 1, Selection) ist bei diesem Anwendungsfall hauptsächlich auf zwei Aspekte zu achten. Unter der Menge an Tweets werden sich viele Beiträge mit wenig Wörtern befinden. Diese beinhalten im Verhältnis zu längeren Tweets wenig Informationen, die zum Erlernen von Mustern hilfreich sind. Außerdem handelt es sich bei ihnen meistens um kurze Antworten oder Kommentare, die sich auf einen anderen Tweet beziehen. Einige der PolitikerInnen, die in der Datenbank vorhanden sind, sind ebenfalls auf europäischer Ebene politisch tätig, weswegen sie Tweets in anderen Sprachen verfassen. Diese müssen ebenfalls herausgefiltert werden, was mit der Python Bibliothek *langdetect* [La 2021] möglich ist. Sie analysiert einen Eingabetext und berechnet eine Wahrscheinlichkeitsverteilung der Zugehörigkeit zu unterschiedlichen Sprachen. Sollte die Sprache mit der höchsten Wahrscheinlichkeit Deutsch sein und diese Vorhersage eine Genauigkeit von über 66 % erreichen, wird der Tweet weiterverwendet und andernfalls entfernt. Dieser Prozentwert soll sicherstellen, dass ein Tweet eindeutig Deutsch ist und nicht nur wenige Prozente über einer anderen Sprache liegt.

Da es keine Datensätze gibt, die fehlende Werte beinhalten, weil nur der Tweet und die Parteizugehörigkeit verwendet werden, muss auf diese Fehlerquelle nicht geachtet werden.

3.4 Datenvorverarbeitung

3.4.1 Textbearbeitung

Da Tweets im Normalfall mit Hashtags, URLs und Sonderzeichen behaftet sind, ist preprocessing nötig (siehe Abbildung 1, Preprocessing). Die verwendete Textbearbeitung beinhaltet das Austauschen von Nutzernamen mit einem „@user“ Token, da der Nutzername keine semantischen Informationen beinhaltet, allerdings noch für die Struktur des Satzes erhalten bleiben soll. Ähnlich wird mit URLs verfahren, die mit einem „URL“ Token ersetzt werden. Hashtags werden in vielen Tweets als normale Wörter in der Struktur eines Satzes verwendet. Außerdem können sie aus mehreren Wörtern mit unterschiedlicher Syntax bestehen, weswegen nicht für jeden Einzelfall garantiert werden kann, dass sie richtig aufgeteilt werden. Dementsprechend bleiben sie unbehandelt, mit Ausnahme des Hashtag Zeichens [TUHH o.D.]. Allerdings können Hashtags zu leicht auf die eigentliche Klassenzugehörigkeit des Tweets hindeuten. Würde von Parteiangehörigen der SPD in den meisten Tweets das Hashtag „#SPD“ vorkommen, könnte dieser Token einen zu großen Einfluss auf die Klassifizierung haben, weswegen die Hashtags in Tweets von PolitikerInnen derselben Partei entfernt werden. Zahlen werden beibehalten, da sie Daten darstellen könnten und sie entweder vom tokenizer unterteilt werden oder als einzelnes Token verwendet werden können.

Da Teile der Bearbeitung abhängig vom gewählten Algorithmus sind, der die word embeddings erstellt, wird hier auf das Kapitel 3.5.2 vorgegriffen, in dem BERT als Algorithmus gewählt wird. Für das BERT-Modell sind noch spezielle Tokens im Eingabetext nötig. Zum einen der „[CLS]“ Token, der den Anfang eines Eingabetexts signalisiert und zum anderen der „[SEP]“ Token, der ein Satzende signalisiert. Dementsprechend werden alle Sonderzeichen entfernt, die kein Satzende markieren, damit der tokenizer vom gewählten BERT-Modell die speziellen Tokens setzen kann.

Um diese Bearbeitungen vorzunehmen, wird die RegEx-Bibliothek verwendet. Mithilfe regulärer Ausdrücke kann ein Eingabetext auf die definierten Kriterien geprüft werden, um sie anschließend zu entfernen.

3.4.2 Ausbalancieren des Datensatzes

Des Weiteren kann der Datensatz unausgeglichen sein, wofür Methoden wie undersampling oder oversampling aus Kapitel 2.3.2 verwendet werden müssen. Welche der beiden Methoden angewendet wird, ist abhängig vom Datensatz, der im vierten Kapitel erstellt wird.

3.4.3 Aufteilen der Daten

Schließlich müssen die Daten vor der Konvertierung zu numerischen Werten der tokenizer des word embeddings und der Eingabe in das NN in Trainings-, Validierungs- und Testdaten unterteilt werden.

Die Trainingsdaten werden als Eingabewerte für das Training des Modells benötigt. Die Validierungsdaten werden verwendet, um nach jeder Epoche die Leistung des Modells bei Daten außerhalb der Trainingsdaten zu messen und die Gewichte dementsprechend anzupassen. Somit kann das Modell mit ungesehenen Daten getestet werden, um dessen Generalisierungsfähigkeit zu ermitteln. Die Testdaten stellen ebenfalls ungesehene Daten dar und werden nach Abschluss des Trainings gebraucht, um die Leistung des Modells abschließend zu bewerten. Um diese Aufteilung durchzuführen, wird die train-test-split Methode der Python Bibliothek sklearn [SK o.D.] verwendet. Hierfür gibt es mehrere häufig verwendete prozentuale Aufteilungen (splits), wie 80-10-10, 70-15-15 und 60-20-20. Im weiteren Verlauf wird der 80-10-10 split verwendet, allerdings können zur Optimierung des Modells auch die anderen Methoden getestet und verglichen werden, da keine der Möglichkeiten pauschal für den jeweiligen Anwendungsfall als geeignet deklariert werden kann [Br 2020c; Dr 2019]. Alternativ könnte die Aufteilung der Daten auch manuell vorgenommen werden, woraus allerdings kein Vorteil entsteht. Das einzige Kriterium ist, dass die Daten letztendlich aufgeteilt sind, worin sich verschiedene etablierte Methoden nicht stark unterscheiden können.

3.5 Datentransformation

Um die Daten zum Trainieren eines Modells zu verwenden, müssen sie in einem geeigneten Format vorliegen, was durch die Transformation der Daten in Schritt 3 aus Abbildung 1 vorgesehen ist. Hierfür werden die Teildatensätze verwendet, um durch word embeddings feature extraction zu betreiben und sie somit zu Vektoren zu konvertieren. Um zu entscheiden, welcher Algorithmus für die Erstellung der word embeddings in Betracht gezogen wird, werden in diesem Kapitel die Leistungen der unterschiedlichen Algorithmen verglichen.

3.5.1 Statische embeddings

3.5.1.1 Word2Vec

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days]
			Semantic	Syntactic	Total	
3 epoch CBOW	300	783M	15.5	53.1	36.1	1
3 epoch Skip-gram	300	783M	50.0	55.9	53.3	3
1 epoch CBOW	300	783M	13.8	49.9	33.6	0.3
1 epoch CBOW	300	1.6B	16.1	52.6	36.1	0.6
1 epoch CBOW	600	783M	15.4	53.3	36.2	0.7
1 epoch Skip-gram	300	783M	45.6	52.2	49.2	1
1 epoch Skip-gram	300	1.6B	52.2	55.1	53.8	2
1 epoch Skip-gram	600	783M	56.7	54.5	55.5	2.5

Abbildung 8: Genauigkeiten der CBOW- und SG-Modelle bei gleichen Parametern [MiChCoDe 2013, Seite 8]

Entsprechend der Vergleiche aus Abbildung 8 eignet sich das SG-Modell von Word2Vec aus Kapitel 2.4.2.1 für Aufgaben, die Semantik behandeln, da es für diese eine höhere Genauigkeit aufweist als das CBOW-Modell. Diese Tatsache ist bereits an den unterschiedlichen Architekturen erkennbar. CBOW verwendet die Wortvektoren umliegender Wörter, also Strukturen, um ein Wort vorherzusagen, was bereits der Definition von Syntax nahekommt. Im Gegenzug dazu erlernt SG das Embedding lediglich durch isolierte Wortvektoren als Eingabe, wodurch auf natürliche Weise die Semantik der Wörter erfasst werden muss, um den Kontext des Wortes vorherzusagen.

3.5.1.2 GloVe

Model	Dim.	Size	Sem.	Syn.	Tot.
ivLBL	100	1.5B	55.9	50.1	53.2
HPCA	100	1.6B	4.2	16.4	10.8
GloVe	100	1.6B	67.5	54.3	60.3
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	64.8	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	80.8	61.5	70.3
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW†	300	6B	63.6	67.4	65.7
SG†	300	6B	73.0	66.0	69.1
GloVe	300	6B	77.4	67.0	71.7
CBOW	1000	6B	57.3	68.9	63.7
SG	1000	6B	66.1	65.1	65.6
SVD-L	300	42B	38.4	58.2	49.2
GloVe	300	42B	**81.9**	**69.3**	**75.0**

Abbildung 9: Ergebnisse unterschiedlicher word embeddings bei Wort-Analogie Aufgaben in Prozent [PeSoMa 2014, Seite 6]

Wie in Abbildung 9 zu erkennen ist, weist GloVe durch die Verwendung der globalen co-occurence Matrix eine deutlich höhere Genauigkeit bei Aufgaben zu Wort-Analogien auf als CBOW oder SG und eignet sich demnach besser als Word2Vec für diesen Aufgabentypen.

3.5.1.3 FastText

In Abbildung 10 ist erkennbar, dass fastText bei Aufgaben im Bereich Syntax im Vergleich zu CBOW und SG in jeder Sprache, außer Englisch, höhere Genauigkeiten aufweist. Die Repräsentation der Semantik der Wörter ist in Deutsch und Englisch allerdings weniger genau. Für die Darstellung der syntaktischen Ähnlichkeiten von Wörtern eignet sich fastText durch die morphologischen Informationen der Teilwörter besser als Word2Vec, während Semantik genauer durch SG dargestellt wird [BoGrJoMi 2017, Seite 5].

		sg	cbow	sisg
CS	Semantic	25.7	27.6	27.5
	Syntactic	52.8	55.0	77.8
DE	Semantic	66.5	66.8	62.3
	Syntactic	44.5	45.0	56.4
EN	Semantic	78.5	78.2	77.8
	Syntactic	70.1	69.9	74.9
IT	Semantic	52.3	54.7	52.3
	Syntactic	51.5	51.8	62.7

Abbildung 10: Genauigkeiten der Modelle SG, CBOW und fastText im Vergleich mit unterschiedlichen Sprachen in Bezug auf Semantik und Syntax [BoGrJoMi 2017, Seite 5]

3.5.1.4 Evaluierung der statischen Embeddings

Grundsätzlich variieren die Genauigkeiten der drei word embedding Algorithmen je nach Anwendungsfall, wie in [DhGaWaSo 2022, Seite 357] erläutert wird, und dementsprechend kann keine eindeutige Aussage darüber getroffen werden, welcher Algorithmus für den Anwendungsfall der Klassifizierung von Tweets am besten geeignet ist. Diese These wird von den Ergebnissen der Vergleichsarbeit *Comparative Analysis of Word Embeddings for Capturing Word Similarities* [ToStoKa 2020] unterstützt. Hierbei erreichen GloVe und fastText bei Aufgaben im Bereich Wortähnlichkeit ähnliche Genauigkeiten.

Die höchste Genauigkeit, beim Vergleich der word embeddings mit einem englischen Datensatz in der ursprünglichen Arbeit [DhGaWaSo 2022], hat allerdings fastText (siehe Abbildung 11), was aus dessen Fähigkeit OOV-Worte darzustellen abzuleiten ist [DhGaWaSo 2022, Seite 356]. Deshalb ist es wahrscheinlicher, dass fastText auch bei diesem Anwendungsfall eine höhere Genauigkeit als Word2Vec und GloVe erzielt. Deshalb werden Word2Vec und GloVe nicht verwendet. Allerdings können LMs durch die Transformer-Architektur und die Menge trainierbarer Parameter noch bessere Leistungen erzielen, weswegen sich bei der initialen Architektur auf LMs beschränkt wird.

Word Embedding	Accuracy (%)
Word2Vec	92.5
GloVe	95.8
FastText	**97.2**

Abbildung 11: Vergleich der Genauigkeiten von Word2Vec, GloVe und fastText bei Tests mit einem Datensatz von 19,977 Nachrichtenartikeln [DhGaWaSo 2022, Seite 356]

3.5.2 Evaluierung der Language Models

Während der Entwicklung unterschiedlicher LMs ist aufgefallen, dass ein Modell mit mehr trainierbaren Parametern tendenziell bessere Ergebnisse liefern kann [De o.D., Folien 23 und 35], weshalb sich dieser Trend bei der Entwicklung neuer Modelle stets durchsetzt. Durch die Menge dieser Parameter und der

Einführung der Transformer-Architektur haben LMs typischerweise mehr trainierbare Parameter als Algorithmen, die statische word embeddings generieren, und können dadurch höhere Genauigkeiten erzielen.

Insgesamt eignet sich jedes der erwähnten LMs für die Erstellung von embeddings für diesen Anwendungsfall, aber RoBERTa liefert beim *General Language Understanding Evaluation* (GLUE) Benchmark in Abbildung 12 im Durchschnitt eine etwas höhere Genauigkeit als BERT und XLNet.

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
Single-task single models on dev										
$BERT_{LARGE}$	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
$XLNet_{LARGE}$	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	**90.2/90.2**	**94.7**	**92.2**	**86.6**	**96.4**	**90.9**	**68.0**	**92.4**	**91.3**	-
Ensembles on test (from leaderboard as of July 25, 2019)										
ALICE	88.2/87.9	95.7	**90.7**	83.5	95.2	92.6	**68.6**	91.1	80.8	86.3
MT-DNN	87.9/87.4	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0	87.6
XLNet	90.2/89.8	98.6	90.3	86.3	**96.8**	**93.0**	67.8	91.6	**90.4**	88.4
RoBERTa	**90.8/90.2**	**98.9**	90.2	**88.2**	96.7	92.3	67.8	**92.2**	89.0	**88.5**

Abbildung 12: Genauigkeiten der drei Modelle in neun Aufgaben anhand des GLUE-Benchmarks [LiOtGoDu 2019, Seite 8]

Allerdings gibt es keine umfangreichen, in deutscher Sprache trainierten Modelle für XLNet und RoBERTa [Hu o.D. a], sondern nur multilinguale, wodurch Genauigkeitseinbußen zu erwarten sind. Im Gegenzug hat BERT mehrere deutsche Modelle vorzuweisen. Hierbei stehen die dbmdz-BERT [Hu o.D. b] und German-BERT [ChMöPiSo 2019] Modelle zur Auswahl. In der Arbeit [AßCoHe 2021, Seiten 5-7] sind deren Leistungen beim „GermanEval17"-Test ermittelt worden, wobei sich herausgestellt hat, dass dbmdz-BERT in allen sechs Aufgabenbereichen die höchste Genauigkeit erzielt.

Da es für die multilingualen XLNet- und RoBERTa-Modelle keine direkten Vergleiche untereinander und zum dbmdz-BERT gibt, müssten die Ergebnisse der Modelle bei der Verwendung verglichen werden, um zu erfassen, welches die höchste Genauigkeit erzielt. Aus zeitlichen Gründen wird allerdings dbmdz-BERT verwendet, da es auf BERT basiert, was die Grundlage der weiteren behandelten LMs darstellt und somit eine ausreichend hohe Genauigkeit aufweisen sollte. Es steht auf der Huggingface Webseite zur Verfügung und kann mit dessen Python Bibliothek eingebunden werden. Falls Probleme bei der Umsetzung auftreten, kann als Alternative von den statischen Modellen fastText verwendet werden, da es, wie LMs, sub-word tokenizing verwendet und es in den Vergleichsarbeiten jeweils eine der höchsten und die höchste Genauigkeit erzielt hat (siehe Kapitel 3.5.1.4).

3.6 Data-Mining

Data-Mining (siehe Abbildung 1, Data Mining) beschreibt das Erkennen von Mustern eines Datensatzes, in der Hoffnung neue Erkenntnisse über diese Daten zu gewinnen. Hierfür wird ein für den Anwendungsfall spezifischer Algorithmus verwendet. Durch die in Kapitel 2.5 erläuterten Vorteile von NNs,

wird sich für die Verwendung dieser entschieden. Das gewählte NN wird nach dem Training bewertet und, je nachdem wie gut es die in der Motivation erläuterte Problemstellung lösen kann, angepasst, um bessere Ergebnisse zu erzielen.

3.6.1 Evaluierung der Klassifizierungsalgorithmen

Im Vergleich zu RNNs bzw. LSTMs und GRUs eignen sich CNNs [WeZhLuWa 2016, Seite 3], eine *gated*-CNN Ausführung [DaFaAuGr 2016, Seite 4] sowie ein *attention-based* CNN [YiScXiZh 2015, Seite 2] besser für Aufgaben im Bereich NLP. Im Gegensatz dazu erzielt ein GRU bei russischen Texten im Bereich sentiment analysis eine höhere Genauigkeit als ein CNN [YiKaYuSc 2017, Seite 2]. Auf Basis dieser Arbeiten existiert die Vergleichsarbeit „*Comparative Study of CNN and RNN for Natural Language Processing*“ [YiKaYuSc 2017], die für CNNs, LSTMs und GRUs ermittelt, welche der Architektur bei welchem Aufgabentyp das beste Ergebnis erzielen kann. GRUs eignen sich für die meisten Aufgaben am besten, außer bei Aufgaben, bei denen bestimmte Kombinationen von Schlagworten für die Lösung von Wichtigkeit ist [YiKaYuSc 2017, Seite 5]. Allerdings liefern CNNs und GRUs bei einer Satzanzahl des Eingabetexts von bis zu zehn Sätzen ähnliche Ergebnisse, jedoch mit einer etwas höheren Genauigkeit des CNNs [YiKaYuSc 2017, Seite 6]. Da Tweets zum Zeitpunkt der Datensammlung eine Zeichenbegrenzung von 280 haben, wird diese Anzahl an Sätzen nur in seltenen Fällen erreicht. Die trainierten Modelle der Arbeit verwenden zusätzlich keine vortrainierten embeddings, die die Semantik der Worte abbilden, weshalb die Ergebnisse mit bereits trainierten embeddings deutlich variieren können [YiKaYuSc 2017, Seite 3]. Da die Semantik bereits durch das Embedding abgebildet wird, könnte das CNN beispielsweise die Vorhersage durch das Erkennen der Muster mit syntaktischer Informationsextraktion bereichern und so, im Vergleich zu GRUs, besser abschneiden. Außerdem basieren GRUs und LSTMs auf RNNs und sind für das Verarbeiten sequenzieller Informationen entwickelt worden, was bei der Klassifikation von Tweets nicht benötigt wird. Aus diesen Gründen wird für die Umsetzung ein CNN verwendet.

3.6.2 Architektur des neuronalen Netzwerks

In diesem Unterkapitel wird die generelle Architektur des verwendeten CNNs erläutert. Hierbei existiert eine Vielzahl einstellbarer Parameter an den einzelnen Schichten des Netzwerks, welche ebenfalls großteilig aufgegriffen werden.

Um die Architektur eines NNs zu definieren, nutzt TensorFlow die *Keras* Bibliothek. Sie stellt Methoden zur Verfügung, mit denen die einzelnen Schichten einer großen Auswahl unterschiedlicher NNs definiert und mit Hyperparametern angepasst werden können. Für das BERT-Modell ist allerdings die Huggingface Bibliothek zuständig. Sie ermöglicht die Konvertierung des heruntergeladenen Modells in ein geeignetes TensorFlow Format, welches in die Architektur eingegliedert werden kann, die mit Keras erstellt wird.

Die grundsätzliche Strukturierung des CNN (KimCNN) ist aus der Arbeit „*Convolutional Neural Networks for Sentence Classification"* [Ki 2014] abgeleitet. Hierbei wird ein word embedding Algorithmus vor die CL geschaltet, worauf eine max-pooling Schicht folgt. Vor einer dense-layer als Output-layer wird zusätzlich eine dropout-layer eingefügt. Somit handelt es sich um eine grundlegende Architektur, die durch das Hinzufügen weiterer CLs leicht erweitert werden kann, falls die Leistung des Modells für den aktuellen Anwendungsfall nicht ausreicht. Es gibt allerdings andere Ansätze, wie CNNs, die auf Zeichenbasis arbeiten [JoZh 2017], und Kombinationen von RNNs bzw. LSTMs und CNNs [WaLiCaChWa 2019]. Diese werden im Rahmen dieser Arbeit allerdings nicht verwendet, da deren Anpassung und Erweiterung, im Vergleich zur vorher erwähnten Architektur, durch deren spezielle Umsetzung einen zu hohen Zeitaufwand bedeutet.

Das KimCNN wird allerdings angepasst, indem die Anzahl an convolutional- und pooling-layers auf drei erhöht und im Anschluss jeweils eine dropout-layer eingefügt wird. Außerdem werden zwei dense-layers verwendet. Diese Anpassungen sollen dem Modell mehr Kapazitäten bieten, da durch das Hinzufügen weiterer Schichten eventuell eine höhere Abstraktions- und Verständnisebene der Daten erzielt wird.

3.6.2.1 Hyperparameter

Bei der Einstellung der Filtergröße der CLs und anderer Parameter, die sich auf das Training des Modells auswirken, wird generell von Hyperparametern gesprochen. An manchen Stellen sind Empfehlungen für Spielräume bei Parametern vorhanden, allerdings werden die verwendeten Werte im Normalfall mehrmals verändert und das Modell erneut trainiert, um den Einfluss dieser Parameter auf die Vorhersage zu ermitteln, da es für sie meistens keine Faustregel gibt.

3.6.2.2 BERT-Modell

Wie in Kapitel 2.4.3 erläutert, muss das BERT-Modell vor das CNN geschaltet werden, um die embeddings zu generieren. Hierfür müssen Eingabetensoren [TF 2023b] in der Länge, die der Anzahl der Worte des Tweets mit den meisten Worten im Datensatz entspricht, definiert werden.

3.6.2.3 Convolutional-layer

Auf die Eingabeschicht mit den definierten Tensoren folgen, wie in Kapitel 3.6.2 festgelegt, drei Kombinationen von convolutional-, pooling- und dropout-layers.

Die CLs verwenden alle eine Filtergröße von drei, da es sich bei den Eingabedaten um kurze Texte handelt. Dementsprechend spielt lokaler Kontext eine wichtigere Rolle als globaler Kontext, wie bei längeren Paragrafen oder Dokumenten. Der Einfluss unterschiedlicher Filtergrößen auf die Genauigkeit des Modells ist von Anwendungsfall zu Anwendungsfall allerdings unterschiedlich und kann nur durch das Testen der verschiedenen Werte beim Trainieren des Modells ermittelt werden. Solange allerdings

im üblichen Bereich von zwei bis fünf gearbeitet wird, steht der lokale Kontext im Fokus, was bei dieser Problemstellung oberste Priorität hat [Vo 2023].

Außerdem wird kein padding verwendet, damit durch die Konvolutionen die Dimensionen der Eingabetensoren verkleinert und die generalisierbaren features der Tweets erfasst werden (siehe Kapitel 2.5.2.1). Wenn padding verwendet wird, schrumpft der Eingabetensor nicht mehr und somit könnte sich das Modell auf spezifische Elemente der einzelnen Eingabewerte fokussieren, die sich beispielsweise am Anfang des Textes befinden, statt die generalisierbaren features der Tweets zu erfassen. Die Auswirkungen vom sogenannten „Zero-Padding“ sind außerdem laut [Ch 2020, Seite 26] vernachlässigbar, da es hauptsächlich für Probleme, die durch die Reduzierung der Größe von Bilddaten entstehen, gedacht ist und verwendet wird. Unterschiedliche padding Methoden weisen ebenfalls keine große Veränderung bei Verwendung von max-pooling auf (siehe Kapitel 3.6.2.4) und die daraus resultierenden Werte befinden sich normalerweise in einem ähnlichen Zahlenbereich.

Zusätzlich wird die Anzahl der Filter pro CL verdoppelt, beginnend mit 16. Dies hat das Ziel pro Konvolution eine höhere Abstraktionsebene der ursprünglichen Daten zu erreichen. Initial beinhalten die Rohdaten viel *noise*, was grundsätzlich Daten bezeichnet, aus denen wenig Informationen gewonnen werden können [Dc 2021]. Pro Konvolution werden die features der Daten extrahiert und somit noise Schritt für Schritt herausgefiltert. Je weniger noise vorhanden, desto mehr Filter werden verwendet, um unterschiedliche, komplexere Muster der bereits extrahierten, primitiveren features zu erkennen [Br 2020d].

3.6.2.4 Pooling-layer

Da der Effekt von „Zero-Padding“ bei max-pooling vernachlässigbar ist, und weil es den höchsten, wichtigsten Wert pro feature map extrahiert, wird max-pooling verwendet. Der höchste Wert stellt nämlich ein ausschlaggebendes Kriterium für die Klassifikation dar. Beim average-pooling kann hingegen, durch das Verwenden des Durchschnitts der Werte, eine unzureichende Erkennung der wichtigen features entstehen, da sie eventuell viel noise beinhalten [Sh 2020]. Beim pooling muss eine Fenstergröße, ähnlich wie bei den Filtern einer CL, definiert werden. Die standardmäßig von Keras eingestellte Größe beträgt hierbei zwei. Sie wird üblicherweise bei der Größe belassen, wodurch die Eingabematrix um die Hälfte schrumpft. In Ausnahmefällen werden höhere Fenstergrößen verwendet, aber im Normalfall wird die Matrix dadurch zu stark verkleinert und Informationen können verloren gehen.

Nach der letzten CL wird global-max-pooling verwendet, das im Vergleich zu max-pooling pro feature-map den höchsten Wert extrahiert. Dies reduziert die Dimensionen der feature-maps und des dementsprechenden Outputs der letzten CL, während die wichtigste Information pro feature-map behalten wird [Ol 2022].

3.6.2.5 Dropout-layer

Da CNNs üblicherweise durch die große Menge an trainierbaren Parametern im Verhältnis zu anderen NNs Anzeichen von overfitting aufweisen, werden nach den pooling-layers dropout-layers eingesetzt. Diese Schichten setzen einen definierten Prozentsatz des Outputs der vorherigen Schicht auf null. Durch das Ausschalten bestimmter Neuronen bzw. Filter durch die dropout-layer, erlernt das Modell viele unterschiedliche Muster der Trainingsdaten, die auch auf Daten außerhalb des Trainings angewendet werden können. Dies liegt daran, dass Neuronen bzw. Filter durch das zufällige Ausschalten unabhängiger voneinander trainiert werden und somit verstärkt generalisierbare Muster erlernen. Der dropout Grenzwert wird auf 20 % festgelegt, da dies ein typischer Wert für reelle Zahlenwerte ist und die Verwendung von hohen dropout Werten zu einem langsameren Trainingsprozess und eventuell underfitting führen kann [SrHiKrSa 2014, Seite 1953].

3.6.2.6 Dense-layer

Nach den drei Kombinationen von convolutional-, pooling- und dropout-layers werden zwei dense-layers verwendet. Durch die zufällige Initialisierung der Gewichte dieser Neuronen, ähnlich zu den Filtern der CLs, kann jedes Neuron mittels backpropagation unterschiedliche Aspekte der Daten erlernen. Im Gegensatz zu einer CL werden allerdings keine features extrahiert, sondern es wird anhand der bereits extrahierten features eine Vorhersage getroffen. Das Ergebnis eines Neurons dieser Schicht basiert auf allen Werten der vorherigen Schicht, statt wie beim Filter der CL, auf einen Teilbereich der Eingabedaten.

Die erste dense-layer besteht aus 128 Neuronen, in Kombination mit der Aktivierungsfunktion ReLu. Auf sie folgt eine dropout-layer, wie bei den pooling-layers, allerdings mit einem dropout Wert von 50 %, da eine dense-layer mit hohem dropout Wert typischerweise zu underfitting und mit geringem Wert zu overfitting führt [SrHiKrSa 2014, Seite 1953]. Die Wahl ist allerdings auch abhängig von der Anzahl an Neuronen der jeweiligen Schicht, weshalb für ideale Leistung des Modells ein Gleichgewicht dieser beiden Parameter gefunden werden muss. Die letzte Schicht des NNs ist eine weitere dense-layer, die aus fünf Neuronen besteht, was der Anzahl an Klassen bzw. der Parteien entspricht. Hier wird als Aktivierungsfunktion softmax verwendet, da sie die Werte in einen Zahlenbereich von null bis eins normalisiert, sodass sie als Summe eins ergeben. Somit ist der Output der letzten Schicht im Netzwerk eine prozentuale Verteilung der Vorhersage der unterschiedlichen Klassenzugehörigkeiten.

3.6.2.7 Optimierungsalgorithmen

Bei einem der wichtigsten Hyperparameter handelt es sich um die Lernrate des NNs. Sobald die loss-function keine großen Verbesserungen mehr aufweist und sich stets in einem ähnlichen Zahlenbereich aufhält, konvergiert die loss-function.

Falls die Lernrate zu hoch eingestellt ist, konvergiert das Modell schneller. Sollte sie allerdings zu hoch gewählt sein, kann es dazu führen, dass sie nicht konvergiert. Falls die Lernrate zu niedrig gewählt ist, konvergiert sie langsamer und das Training dauert dementsprechend länger, aber führt eventuell zu einer besseren Generalisierung der Muster der Daten. Allerdings kann die loss-function hierbei in einem lokalen Minimum konvergieren, statt am absoluten Minimum, was die bestmögliche Anpassung des Modells in der jeweiligen Konfiguration darstellt [Br 2020e]. Um die Wahl der bestmöglichen Lernrate zu vereinfachen, sind unterschiedliche Optimierungsalgorithmen entwickelt worden, die die Lernrate während des Trainings flexibel anpassen. Einer der am häufigsten verwendeten Algorithmen ist der Adam-optimizer, welcher auf den AdaGrad und RMSProp Algorithmen basiert. In Abbildung 13 weist dieser einen niedrigeren Wert der loss-function auf als AdaGrad, RMSProp, SGDNesterov und AdaDelta, weshalb er auch für dieses CNN verwendet wird. Die standardmäßig eingestellte und empfohlene initiale Lernrate entspricht hierbei $lr = 0.001$ [Br 2021b].

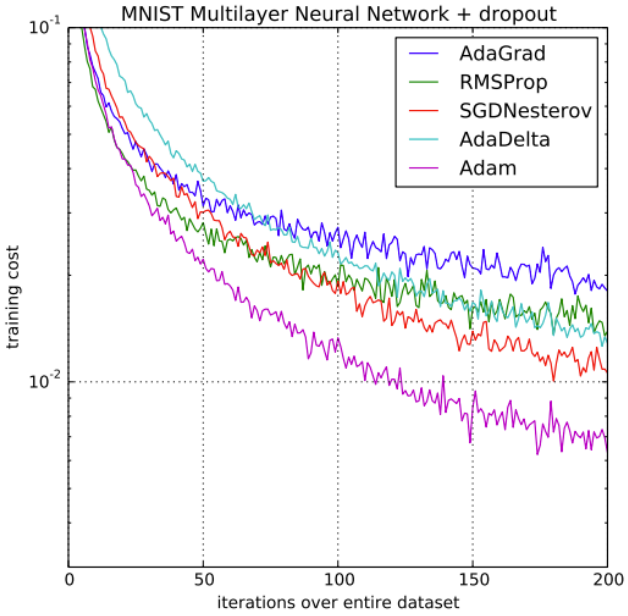

Abbildung 13: Loss-functions unterschiedlicher Optimierungsalgorithmen [KiBa 2015]

4 Entwicklung des Modells

Dieses Kapitel befasst sich mit der Umsetzung des Konzepts aus Kapitel 3. Dazu werden die KDD-Schritte durchlaufen und die festgelegten Methoden verwendet, um ein Modell zu entwickeln, welches die Klassifikation vornimmt.

4.1 Datensammlung

Das Entfernen der Parteien wird mit der Pandas Funktion *apply* durchgeführt. Diese Funktion erhält eine Bedingung in Form einer weiteren Funktion. Sollte die Bedingung zutreffen, werden die Indizes der entsprechenden Werte zurückgegeben. Diese Indizes werden wiederum verwendet, um einen neuen Datensatz mit dem für den jeweiligen Index korrespondierenden Wert zu speichern. Letztendlich sind noch 3379 PolitikerInnen in der Datensammlung vorhanden.

Nach Entfernen der Parteien, die nicht im Bundestag residieren (siehe Kapitel 3.2), ist anhand der Verteilung der Twitter-Accounts pro Partei in Abbildung 14 zu erkennen, dass die SPD, Grünen und CDU/CSU jeweils ungefähr ein Viertel der Accounts ausmachen, die FDP und AfD hingegen deutlich weniger. Dementsprechend könnte die Klassifizierung der FDP und AfD qualitativ von weniger Wert sein, da durch die geringere Anzahl an Accounts ebenso eine geringere Vielfalt an unterschiedlichen Tweets vorhanden sein könnte.

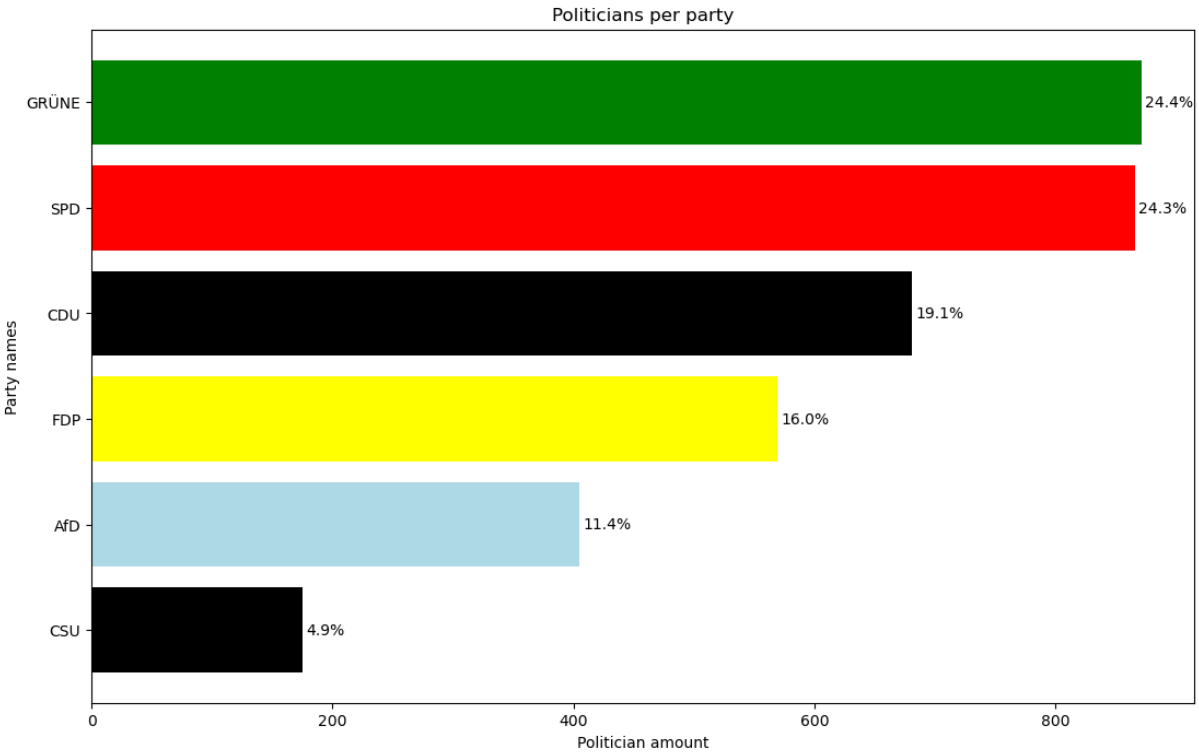

Abbildung 14: Verteilung der Twitter-Accounts nach dem Entfernen der Parteien außerhalb des Bundestags und der Partei DIE LINKE [eigene Darstellung]

Um die Tweets dieser PolitikerInnen automatisiert zu sammeln, werden die Twitter-Accounts aus der Datenbank verwendet. Diese Daten liegen als CSV-Datei vor, womit für jeden Eintrag, mit Hilfe der Twitter URL des jeweiligen Accounts, die Tweets von der Twitter API abgefragt werden können. Somit kann mit Pandas über den Datensatz iteriert werden und es können pro Eintrag die Tweets des Twitter-Accounts mittels snscrape (siehe Kapitel 3.2) abgerufen werden. Die dafür benötigte Methode ist die *TwitterUserScraper* Funktion. Pandas wird im Anschluss dafür verwendet die gesammelten Daten in einer CSV-Datei abzuspeichern, um sie im weiteren Verlauf verwenden zu können. Die Anzahl der abgefragten Tweets beträgt 6,2 Millionen und dessen Verteilung ist anhand Abbildung 15 dargestellt.

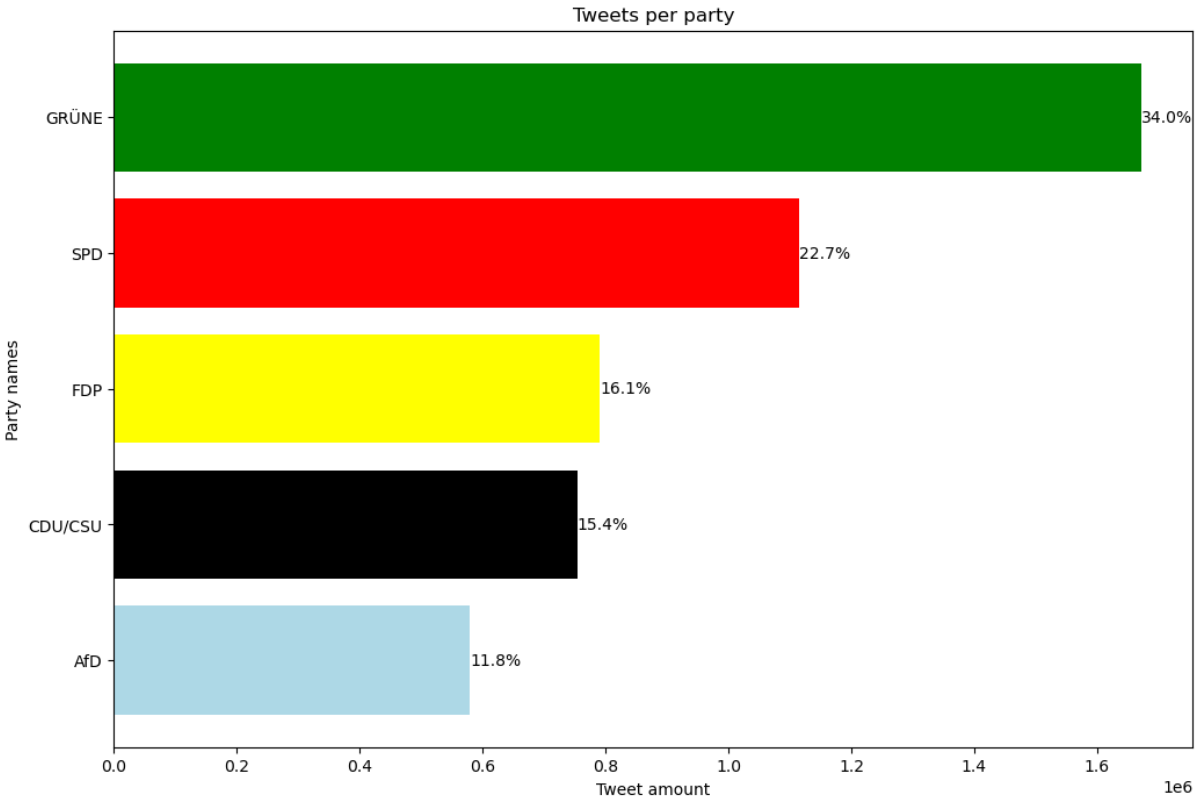

Abbildung 15: Tweets pro Klasse in Prozent [eigene Darstellung]

4.2 Datenselektion

Die Verteilung kurzer Tweets, wie in Kapitel 3.3 festgelegt, ist anhand Abbildung 16 dargestellt. Die Menge an Tweets mit maximal fünf Wörtern ist im Vergleich zum Durchschnitt gering. Ab sechs Wörtern pro Tweet steigt die Anzahl allerdings stark an. Dementsprechend werden Tweets entfernt, die eine maximale Wortanzahl von fünf haben, was sich auf ungefähr 50.000 beläuft.

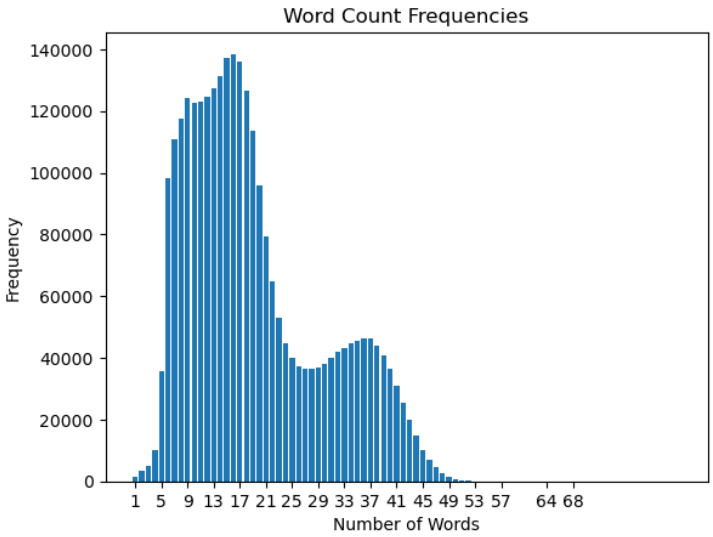

Abbildung 16: Anzahl der Tweets pro Wortanzahl [eigene Darstellung]

Zusätzlich müssen nicht deutschsprachige Tweets entfernt werden. Um dies zu erreichen, wird die CSV-Datei mit Pandas ausgelesen und es werden einzelne Dateneinträge, wie beim Entfernen der Parteien, mit der *apply* Funktion in Kombination mit der langdetect Bibliothek herausgefiltert.

4.3 Datenvorverarbeitung

4.3.1 Textbearbeitung

Um die in Kapitel 3.4.1 festgelegte Textbearbeitung durchzuführen, wird über den zuvor mittels Datenselektion bearbeiteten und abgespeicherten Datensatz iteriert. Gleichermaßen wird auch die apply Funktion verwendet, um jeden Tweet zu bearbeiten. Zuvor wird eine Funktion definiert, die alle spezifizierten Textbearbeitungen an einem Eingabetext durchführt. Sie wird in die apply Funktion gegeben, welche über die Daten iteriert und die Funktion auf jeden Tweet anwendet (siehe Anhang A für Beispieltweets).

4.3.2 Ausbalancieren der Daten

Anhand Abbildung 15 ist zu erkennen, dass die Verteilung der Tweets pro Partei unausgeglichen ist, weswegen der Datensatz mit Methoden aus Kapitel 2.2.2 ausbalanciert werden muss. Da es sich bei den vorhandenen Daten allerdings um einen umfangreichen und somit komplexen Datensatz handelt, ist dies eventuell nicht nötig, da trotz der Unausgeglichenheit genug Informationen für die Klassifizierung der Daten vorhanden sein könnten. Bevor diese Methoden angewendet werden, wird deshalb zuerst ein Trainingsdurchlauf initiiert und eine *confusion*-Matrix erstellt. Anhand dieser Matrix werden die Vorhersagen des Modells in Zusammenhang mit den richtigen Ergebnissen der einzelnen Tweets pro Klasse gebracht, wodurch ermittelt werden kann, ob eine Anpassung der Verteilung der Daten nötig ist.

4.3.3 Aufteilen der Daten

Damit die Verhältnisse der Klassen in den aufgeteilten Datensätzen dem ursprünglichen Verhältnis gleichen, wird der *stratify* Parameter bei der Funktion train-test-split (siehe Kapitel 3.4.3) gesetzt. Dieser erhält alle Labels des aufzuteilenden Datensatzes und teilt ihn anhand dieser gleichmäßig auf. Somit bleibt das Verhältnis der Klassen des originalen Datensatzes erhalten. Damit die Aufteilung bei jedem Aufruf der Funktion mit denselben Daten reproduzierbar ist, wird derselbe *random state* Parameter gesetzt. Die Aufteilung wird zweimal durchgeführt, um zuerst die Testdaten und im Anschluss die Validierungsdaten abzuspalten (siehe Anhang B, Codeblock 1). Danach werden die Labels der Daten in numerische Form umgewandelt und eine map angelegt, damit die Ziffern ihren ursprünglichen Labels zugeordnet werden können (siehe Anhang B, Codeblock 2).

4.4 Datentransformation

Um die embeddings aus BERT zu erhalten, müssen die Daten zuerst durch den tokenizer tokenisiert werden. Hierfür ist die Funktion *encode_plus* zuständig. Diese wird in der eigenen *bert_encode* Funktion verwendet (siehe Codeblock 1), die den mit Pandas eingelesenen Datensatz iteriert und den Tweet pro Iterationsschritt tokenisiert. Hierfür wird die höchste Anzahl an Tokens pro Tweet im Datensatz benötigt, damit alle kürzeren Tweets durch den tokenizer mit Nullwerten aufgefüllt werden können. Dies wird durch die Parameter *max_length* und *padding* ermöglicht. Um die speziellen BERT-Tokens zu setzen, wird der Parameter *add_special_tokens* eingeschaltet.

Codeblock 1: Funktion, um die Eingabewerte für das BERT-Modell zu erhalten [eigene Darstellung]

```
def bert_encode(texts, tokenizer, max_length=100):
    all_tokens = []
    all_masks = []

    for text in texts:
        text = tokenizer.encode_plus(text, add_special_tokens=True,
max_length=max_length, return_attention_mask=True, return_tensors='tf',
padding='max_length', truncation=True)
        input_ids = text['input_ids'][0]
        attention_mask = text['attention_mask'][0]
        all_tokens.append(input_ids)
        all_masks.append(attention_mask)

    return [tf.convert_to_tensor(all_tokens), tf.convert_to_ten-
sor(all_masks)]
```

Bei den resultierenden Werten handelt es sich um die *input_ids*, die eine numerische Darstellung der tokenisierten Eingabesequenz sind und die *attention_mask*, welche die Tokens als Liste binärer Werte darstellt, die signalisiert, welche Tokens Teil der Eingabesequenz sind und welche Tokens das Padding

sind. Diese Werte werden vom BERT-Modell im NN verwendet, um die embeddings zu generieren. Die Ergebnisse werden in Form eines TensorFlow Tensors zurückgegeben. Ein Tensor ähnelt einer Matrix mit dem Unterschied, dass er mehr als zwei Dimensionen unterstützen kann [Gr 2021] und hat den Vorteil, dass er, kombiniert mit GPUs, schnellere Trainingszeiten aufweisen kann. Somit befinden sich die Daten in einem geeigneten Format für die Eingabe in das NN.

4.5 Data-Mining

4.5.1 Implementierung des NNs

4.5.1.1 BERT-Modell

Um die Tensoren mit vorbestimmter Größe zu definieren, wird mit Pandas über den Datensatz iteriert und mit der apply Funktion die Länge der jeweiligen Tweets ermittelt. Der Maximalwert wird abgespeichert und in die Tensoren übergeben, welche mit Keras erstellt werden. Es wird ein Tensor für jeweils die tokenisierte Eingabesequenz und die attention mask initialisiert. Im Anschluss wird das BERT-Modell mit der Huggingface Bibliothek von Huggingface heruntergeladen und diesem die definierten Eingabetensoren übergeben. Sie dienen allerdings nur zur Definition des Modells und dessen Eingabeparameter und sind noch nicht mit Werten befüllt. Um die Trainingszeit zu reduzieren, wird das BERT-Modell so eingestellt, dass es während des Trainings nicht weiter lernt, indem es sich auf die Trainingsdaten anpasst (siehe Anhang B, Codeblock 3). Das bereits trainierte Modell sollte ausreichen, um die embeddings zu generieren, da es bereits mit einem umfangreichen Datensatz trainiert wurde, der 2.350.234,427 Tokens umfasst [Hu o.D. b]. Falls es jedoch zu geringer Leistung des NNs kommt, kann dieser Parameter noch eingeschaltet werden.

4.5.1.2 CNN

Das Ergebnis des BERT-Modells wird in die erste CL gegeben (siehe Codeblock 2). Um die definierten Hyperparameter einzustellen, können der mit Keras definierten CL Optionen als Parameter übergeben werden. Um die Filtergröße auf drei einzustellen, wird der „kernel_size“ Parameter gesetzt. Für die Filteranzahl wird der „filters“ Parameter und für das padding der „padding“ Parameter gesetzt. Hierbei steht „same“ für das Verwenden und „valid“ für das Weglassen von padding. Danach muss noch die Aktivierungsfunktion ausgewählt werden, was mit dem „activation“ Parameter ermöglicht wird.

Codeblock 2: Implementation einer Kombination von convolutional-, dropout- und pooling-layer [eigene Darstellung]

```
conv1d = tf.keras.layers.Conv1D(filters=16, kernel_size=3, padding='valid',
                                activation='relu')(bert_output)
pool1d = tf.keras.layers.MaxPooling1D(pool_size=2)(conv1d)
dropout1 = tf.keras.layers.Dropout(0.2)(pool1d, training=True)
```

Im Anschluss folgt die max-pooling Schicht, dessen Fenstergröße durch den „pool_size“ Parameter auf zwei eingestellt wird. Die global-max-pooling Schicht nach der dritten pooling-layer benötigt keine Fenstergröße.

Die darauffolgende dropout-layer erhält ebenfalls Parameter. Hierbei ist allerdings hauptsächlich die dropout-Schwelle von Wichtigkeit, welche als Dezimalwert übergeben wird. Damit dropout nur während des Trainings im Modell verwendet wird, aber nicht mehr beim trainierten Modell, wird allerdings noch der „training“ Parameter beim Übergeben der vorherigen Schicht in die aktuelle dropout-layer eingeschaltet.

Im Anschluss werden die beiden dense-layers definiert. Sie benötigen die Anzahl an Neuronen, die mit dem Parameter „units“ eingestellt werden, sowie die jeweilige Aktivierungsfunktion.

Um den Optimierungsalgorithmus Adam einzustellen, wird die dementsprechende Funktion von Keras verwendet und die Lernrate auf 0.001 gesetzt. Das Modell muss abschließend kompiliert werden, was mit der *compile* Funktion von Keras ermöglicht wird. Sie erhält den Optimierungsalgorithmus sowie die loss-function, die für eine Multiklassifikationsaufgabe eine *sparse categorical crossentropy* ist.

4.5.2 Hardwarelimitierungen

Da die Leistung des Privatcomputers des Autors nicht für das Trainieren mit einem Datensatz dieses Umfangs ausreicht, muss zuerst eine alternative Lösung gefunden werden, um das Training durchzuführen.

Durch die batch-size kann eine Epoche in mehrere Iterationen aufgeteilt werden. Statt alle Trainingsdaten auf einmal pro Iteration zu verwenden und somit den Zwischenspeicher zu überlasten, können kleinere Batchgrößen angegeben werden, wie beispielsweise vier oder acht. Hierdurch erhöht sich die Trainingszeit allerdings, da nach jedem Batch die Gewichte des Modells angepasst werden. Je größer die Batchgröße, desto geringer die Trainingszeit. Hier muss allerdings ein Gleichgewicht gefunden werden, da die Gewichte in einem gewissen Abstand aktualisiert werden sollten, damit das Erlernte auf die nächsten Daten angewendet werden kann. Ein weiteres Problem bei einer geringen Batchgröße ist, dass pro Aktualisierung der Gewichte eventuell nur Daten einer Politikerin oder eines Politikers oder einer Partei verwendet wurden, was zu einem ungleichmäßigen Training führen kann, wodurch sich die Zeit zum Konvergieren erhöht. Durch diese Probleme ist das Wählen einer kleinen batch-size keine Alternative. Stattdessen gibt es Cloud-Anbieter, die Server mit mehr Rechenleistung zur Verfügung stellen. Einer dieser Anbieter ist AWS, welcher bei der dpa-infocom bereits genutzt wird. Dementsprechend ist es naheliegend, den SageMaker Service von AWS [AWS 2023a] für das Trainieren des NNs zu verwenden. Andere Möglichkeiten wären Cloud AI von Google [GC 2023] und Azure Machine Learning von Microsoft [AML 2023]. Die Einrichtung einer Entwicklungsumgebung ist in AWS durch viele Tutorials ausreichend dokumentiert, um die in Kapitel 3.1 definierte Entwicklungsumgebung einzurichten [AWS 2023b]. SageMaker ermöglicht zusätzlich das Bereitstellen von Modellen. Somit kann ein trainiertes

Modell über eine API angesprochen und Vorhersagen erstellt werden. Dadurch muss das Modell nicht auf einer Website eingebunden sein, sondern kann über die Schnittstelle angesprochen werden, was das Erstellen einer solchen Website vereinfacht.

4.5.3 Trainingsumgebung

Das Modell wird initial mit einer batch-size von 32 trainiert, da diese in den meisten Anwendungsfällen die besten Ergebnisse erzielt [Br 2020f]. Das Training findet auf einer G4dn EC2 Instanz von AWS statt. Es umfasst eine NVIDIA T4 Tensor Core GPU mit 16 GB Speicher, 32 GB Arbeitsspeicher und acht CPU-Kerne.

4.5.4 Erstellen der Trainingsfunktion

Um das Training zu starten, muss die *fit* Funktion von Keras auf das Modell angewendet werden. Sie benötigt minimal die Trainings- und Validierungsdaten und die dazugehörigen Labels. Damit die Daten im richtigen Format für das Training sind, wird die in Kapitel 4.4 definierte bert_encode Funktion zu einem Generator umgewandelt. Dieser wird von der fit Funktion aufgerufen und bringt die Daten in ein geeignetes Format. Des Weiteren erhält die Funktion die batch-size als Parameter, welche auf 32 eingestellt wird.

Während des Trainings werden primär die Metriken accuracy und loss des Trainings- und Validierungsdatensatzes beobachtet, um eine Aussage über die Leistung des Modells zu treffen. Sollte eine angemessene Kombination der Modellarchitektur und Hyperparameter gefunden werden, werden ebenfalls Metriken wie F1-Score anhand des Testdatensatzes ermittelt. Hierfür können ebenfalls Methoden von Keras verwendet werden, die mit dem *metrics* Parameter der fit Funktion eingestellt werden.

4.5.5 Erster Trainingsdurchlauf

Der erste Trainingsdurchlauf ist abgebrochen worden, da die Menge an Daten dazu geführt hat, dass eine Epoche mehrere Tage zum Trainieren benötigt. Da erst nach jeder Epoche mit dem Validierungsset geprüft wird, wie genau das Modell Daten außerhalb des Trainingssets vorhersagt und anhand dieser Metrik overfitting erkannt wird, ist es bis zu diesem Zeitpunkt eventuell schon zu spät, um das Training abzubrechen. Damit der validation-loss, der den Wert der loss-function für die Validierungsdaten darstellt, besser überwacht werden kann, wird der Datensatz in mehrere Teildatensätze unterteilt. Somit wird die benötigte Zeit für eine Epoche verringert und es kann auf eventuelles overfitting, durch das Anpassen der Hyperparameter oder Stoppen des Trainings reagiert werden. Der Prozess des iterativen Trainings mit unterschiedlichen Teildatensätzen wird ebenfalls bei *cross-validation* verwendet. Hierbei werden typischerweise unterschiedliche Modellarchitekturen mit gleichen Trainingsdaten trainiert und im Anschluss mit einem gleichen Validierungsset überprüft. Das Modell mit den besten Ergebnissen wird für die weiteren Trainingsiterationen verwendet und mit dem Testset evaluiert [Ta 2021b].

Der Datensatz wird somit in 20 Teile mit jeweils fünf Prozent der Gesamtdaten aufgespalten. Jeder Teil wird erneut zu 90 % Trainings- und zehn Prozent Validierungsdaten umgewandelt, wodurch die 80-10-10 Aufteilung beibehalten wird. Zusätzlich zu den bereits erläuterten Hyperparametern wird für das Training noch eine *early-stopping* Funktion in die Trainingsfunktion übergeben, die den validation-loss überwacht. Sollte sich dieser innerhalb von drei Epochen nicht verbessern, wird das Training abgebrochen und die besten Gewichte wiederhergestellt. Somit wird das Trainieren von Gewichten mit overfitting verhindert.

4.6 Iterative Evaluierungen und Anpassungen

Nach der ersten Iteration über einen Teildatensatz, wird, wie in Kapitel 4.3.2 festgelegt, eine confusion-Matrix erstellt, um zu überprüfen, ob der unausgeglichene Datensatz ein Problem für die Vorhersagen des Modells darstellt (siehe Abbildung 17) [Br 2020g; La 2022].

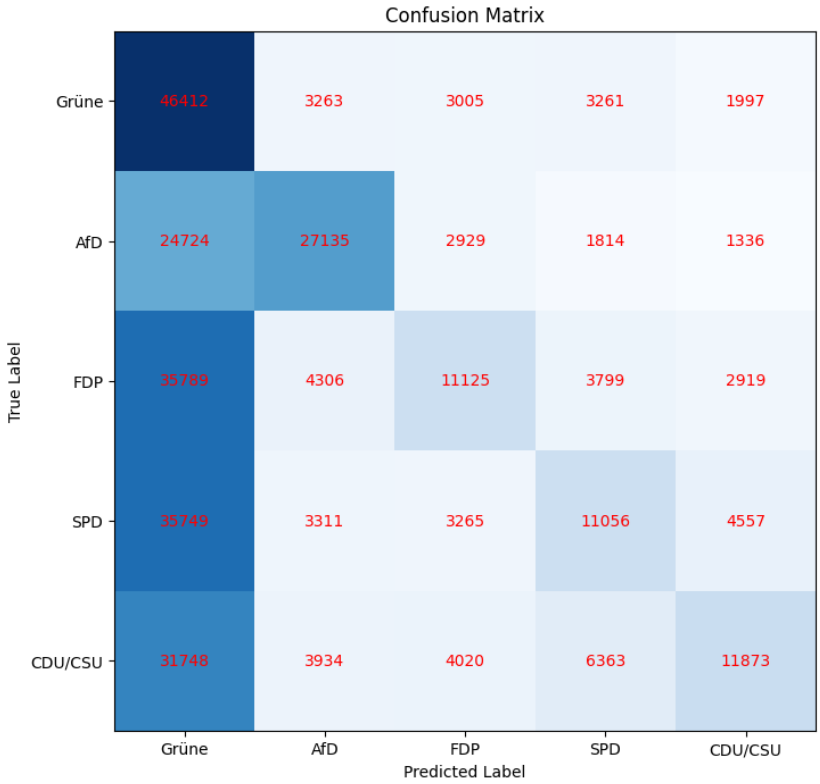

Abbildung 17: Confusion-Matrix der ersten Trainingsiteration [eigene Darstellung]

In Abbildung 17 sind die Vorhersagen des Modells an der x-Achse und die richtigen Klassen an der y-Achse dargestellt. Anhand dieser Vorhersagen ist deutlich zu erkennen, dass die Tweets in den meisten Fällen den Grünen zugeordnet werden, obwohl diese von einer anderen Partei stammen. Hier ist die Tendenz des Modells zur dominierenden Klasse eindeutig erkennbar. Da allerdings nur die ersten fünf Prozent der Gesamtdaten in dieser Iteration verwendet worden sind, ist es schwierig vorherzusagen, ob sich dieser Trend fortführt. Aus zeitlichen Gründen wird sich trotzdem dafür entschieden, den Datensatz vorsorglich auszubalancieren.

Das Verwenden von oversampling bedeutet einen Anstieg der Datenmenge. Da die erste Iteration auf der EC2 Instanz von AWS bereits acht Stunden gedauert hat und dies noch 19-mal wiederholt werden müsste, ergäben sich 160 Stunden Trainingszeit für den gesamten Datensatz. Da die Kosten hierfür bis zu einem gewissen Grad von der dpa-infocom gedeckt werden, stellt dies zwar kein finanzielles Problem dar, allerdings würde die Dauer durch oversampling noch vervielfacht werden. Aus diesem Grund wird undersampling verwendet. Dadurch ergibt sich potenziell ein Nachteil für die Leistung des Modells, allerdings verringert sich die Trainingsdauer dadurch und die Kapazitäten im Rahmen dieser Arbeit werden nicht weiter ausgedehnt. Nach Anwendung sind noch 2.906.895 Tweets vorhanden.

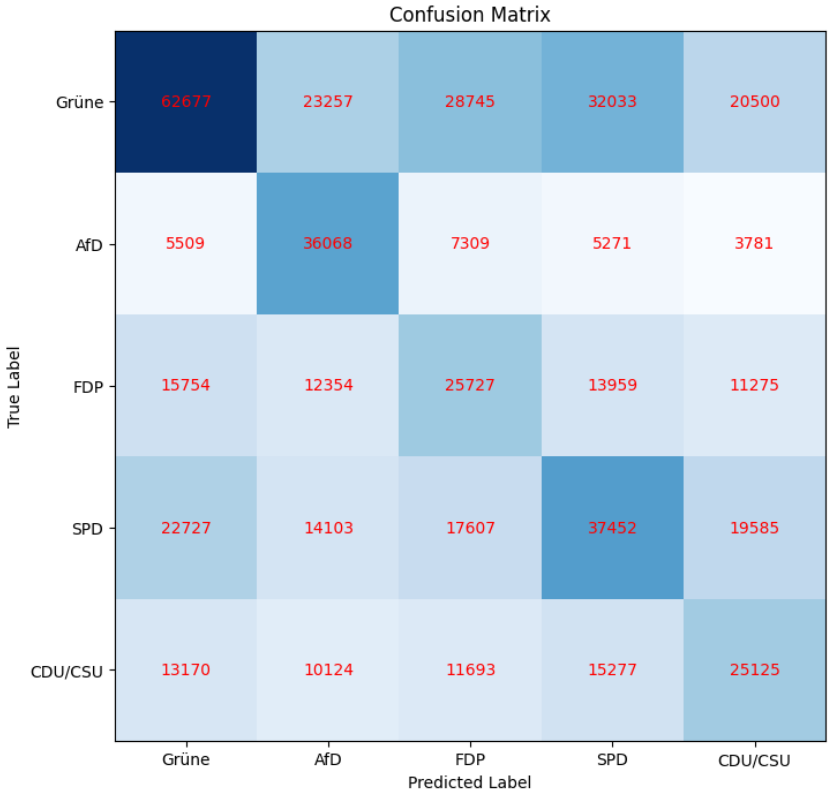

Abbildung 18: Confusion-Matrix der ersten Trainingsiteration mit undersampling [eigene Darstellung]

Anhand Abbildung 18 ist bereits nach einer Iteration zu erkennen, dass das Modell keine eindeutige Tendenz in Richtung der Grünen aufweist. Allerdings ist eine leichte Tendenz in Richtung der restlichen Klassen aufgetreten, da andere Parteien häufig mit den Grünen verwechselt werden. Insgesamt liegt trotzdem eine deutlich ausgeglichenere Verteilung vor. Außerdem ist die in Kapitel 4.1 vermutete geringere Qualität bei Vorhersagen der AfD und FDP nicht stark im Vergleich zu den anderen Klassen zu erkennen, obwohl das Model nur mit fünf Prozent der Daten trainiert ist.

4.6.1 Fortführung des Trainings

Während des ersten Versuchs eines vollständigen Trainingsdurchlaufs mit allen Daten ist nach der dritten Iteration, also 15 % der Daten, festgestellt worden, dass sich die Genauigkeit des Modells nicht

verbessert und in der zweiten Iteration maximal 41 % erreicht. Dabei sind keine Anzeichen von overfitting aufgetreten, da die Genauigkeit und der loss des Trainings- und Validierungssets im gleichen Bereich liegen (siehe Abbildungen 19 bis 21). Da während des Trainings eines NNs das Auftreten von Problemen üblich ist, werden zuerst einige Hyperparameter der bestehenden Modellarchitektur angepasst. Dies allerdings Parameter für Parameter, damit der ausschlaggebende Faktor isoliert werden kann (siehe Kapitel 3.6.2.1).

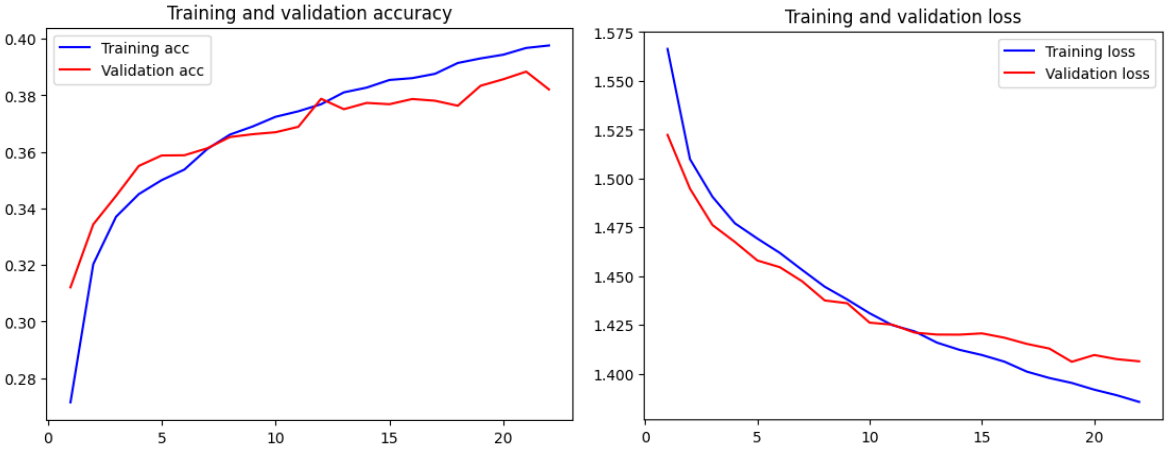

Abbildung 19: Zweite Iteration, Genauigkeit (links), erste Iteration, loss (rechts) [eigene Darstellung]

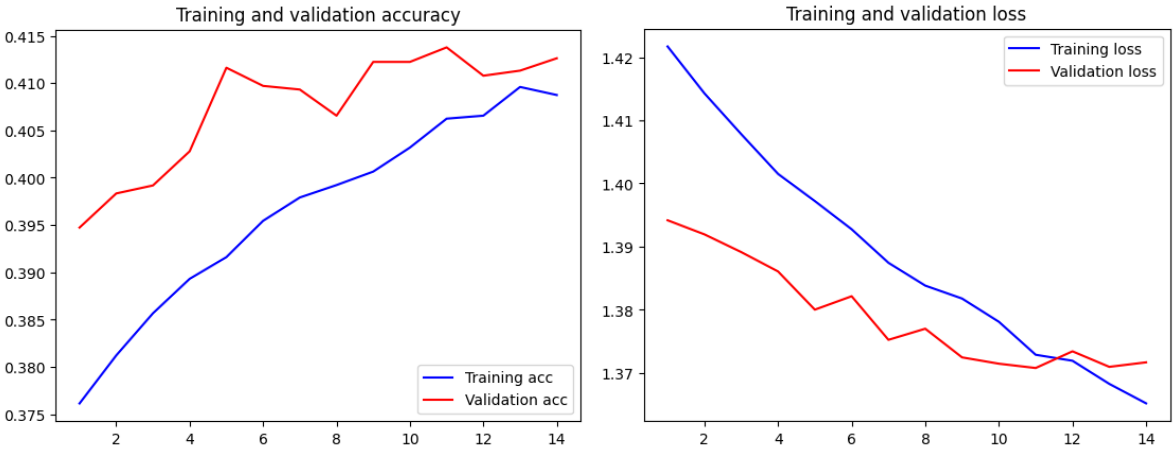

Abbildung 20: Zweite Iteration, Genauigkeit (links), zweite Iteration, loss (rechts) [eigene Darstellung]

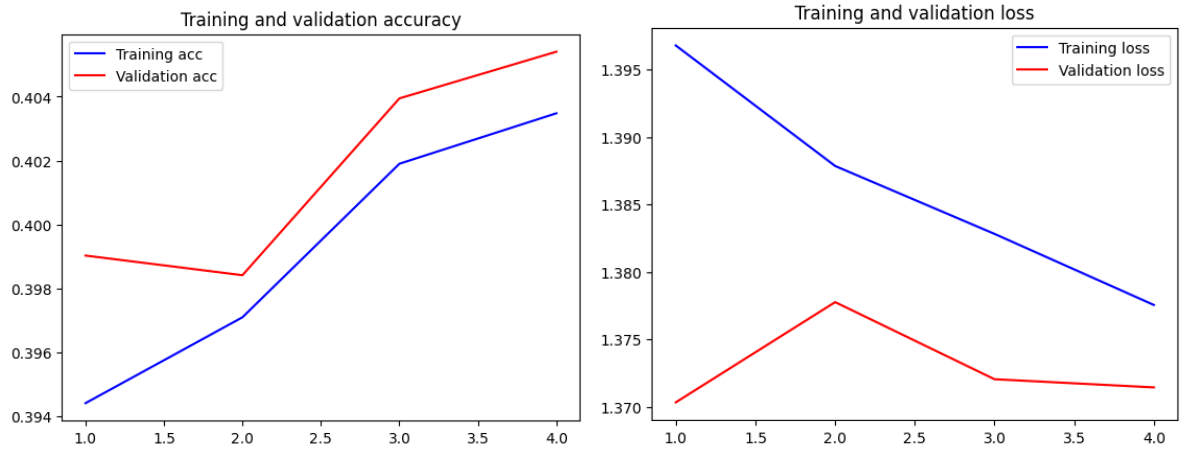

Abbildung 21: Dritte Iteration, Genauigkeit (links), dritte Iteration, loss (rechts) [eigene Darstellung]

4.6.2 Anpassungen der Hyperparameter

Zuerst wird die batch-size erhöht, um auszuschließen, dass zu wenig Informationen innerhalb eines batchs zur Verfügung stehen, um das Modell zu verbessern. Allerdings hat sich die Genauigkeit, bei einer batch-size von jeweils 64 (siehe Abbildung 22) und 128 (siehe Abbildung 23) nach einer Iteration, nicht verbessert. Somit wird bei einer Batch Size von 32 verblieben und andere Parameter angepasst.

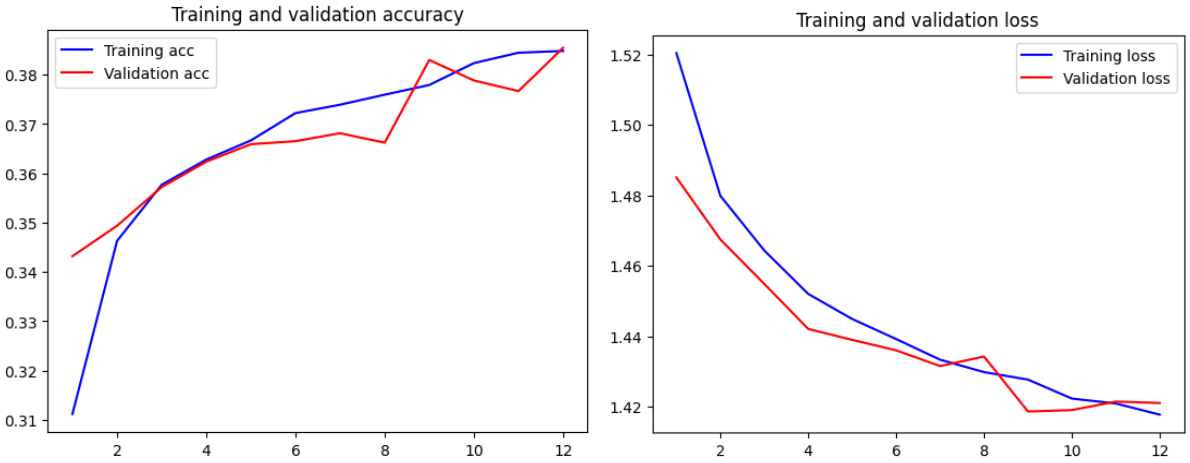

Abbildung 22: Erste Iteration, Genauigkeit bei 64 batch size (links), erste Iteration, loss bei 64 batch size (rechts) [eigene Darstellung]

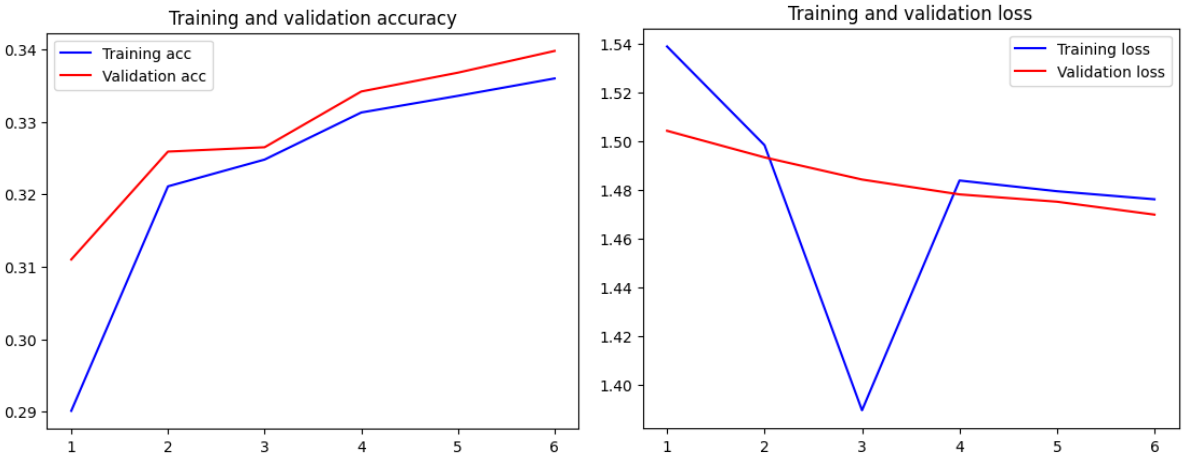

Abbildung 23: Erste Iteration, Genauigkeit bei 128 batch size, (links), erste Iteration, loss bei 128 batch size (rechts) [eigene Darstellung]

Danach wird eine weitere CL hinzugefügt, damit das Modell mehr Kapazitäten zur Verfügung hat, die Muster der Daten zu erkennen und somit underfitting zu vermeiden. Ebenso wie nach der Erhöhung der batch-size, hat dies keine Verbesserung erzielt, sondern eine starke Verringerung der Genauigkeit (siehe Abbildung 24).

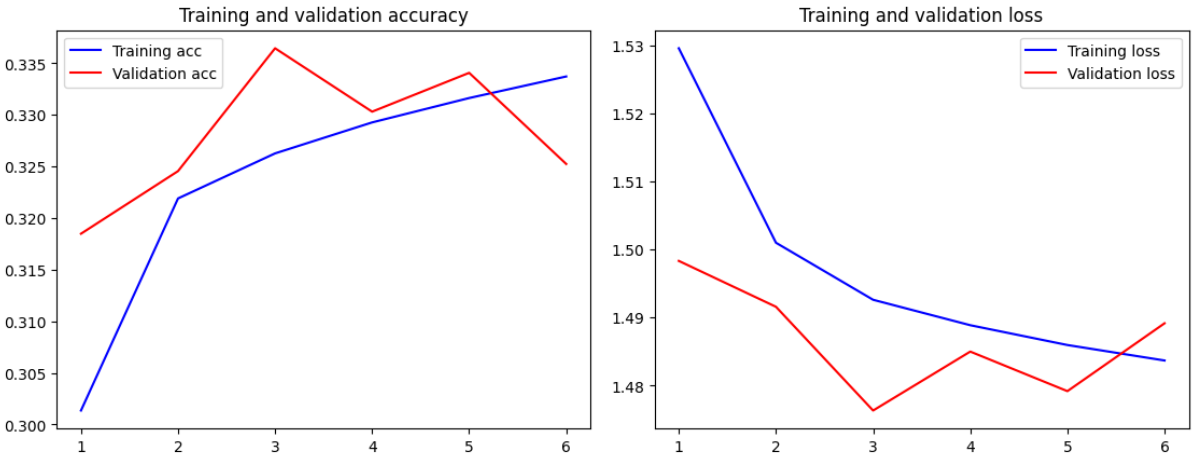

Abbildung 24: Erste Iteration, Genauigkeit bei 4 CLs (links), erste Iteration, loss bei 4 CLs (rechts) [eigene Darstellung]

Da der Datensatz verhältnismäßig groß ist und dadurch wahrscheinlich viel Varianz und Komplexität aufweist, werden die dropout-layers entfernt, um zu überprüfen, ob zu viele Informationen verloren gehen, wodurch das Modell zu underfitting neigt. Hierbei ist die Trainingszeit deutlich gesunken und das Modell erreicht eine Genauigkeit von 42 %. Allerdings ist der validation-loss im Verhältnis zum training-loss angestiegen (siehe Abbildungen 25 und 26), was ein Anzeichen von overfitting ist. Trotzdem werden die dropout-layers für die weiteren Versuche entfernt, da der primäre Fokus aktuell auf der Erhöhung der Genauigkeit liegt und Maßnahmen gegen overfitting im Anschluss erneut eingebaut werden können, sollte es ein zu großes Problem darstellen.

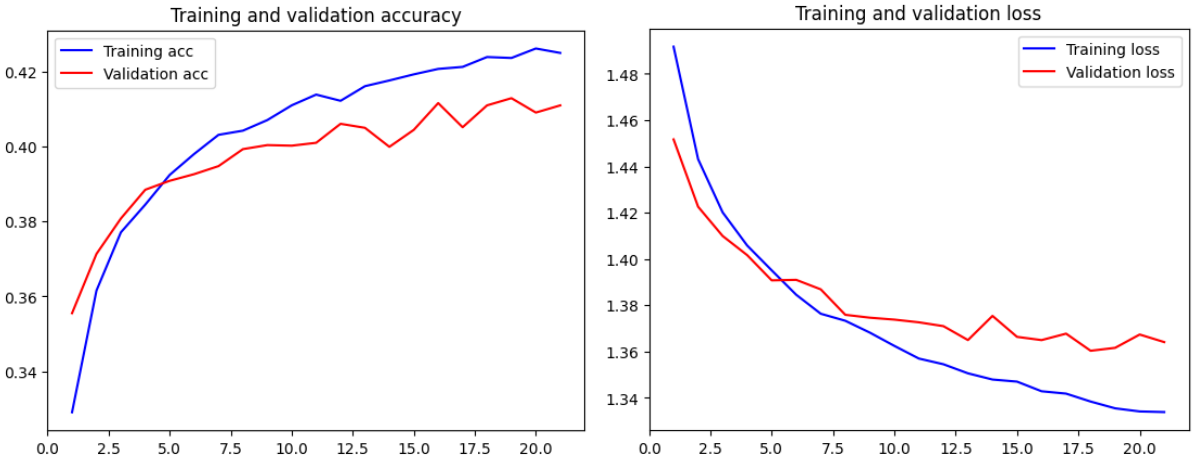

Abbildung 25: Erste Iteration, Genauigkeit ohne dropout (links), erste Iteration, loss ohne dropout (rechts) [eigene Darstellung]

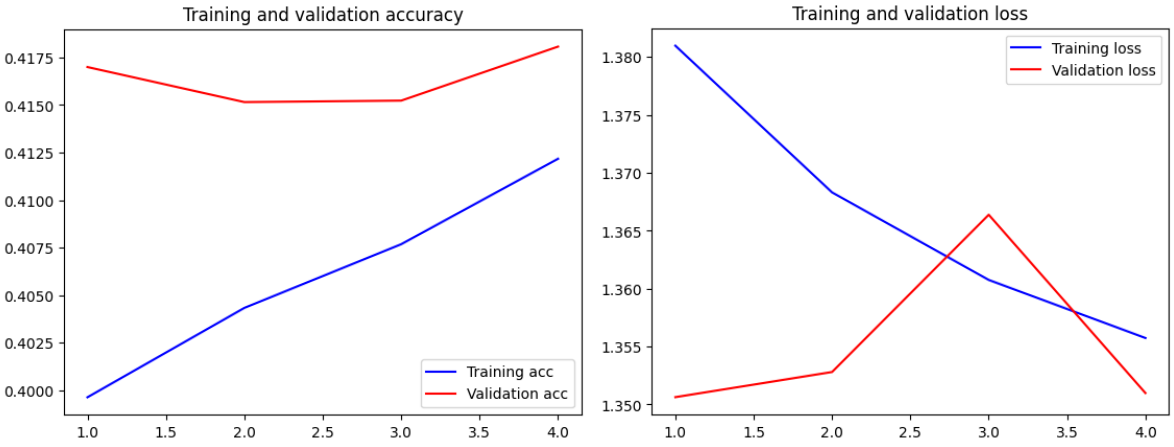

Abbildung 26: Zweite Iteration, Genauigkeit ohne dropout (links), zweite Iteration, loss ohne dropout (rechts) [eigene Darstellung]

Da sich die Genauigkeit nach mehreren Anpassungen nicht stark verändert hat, handelt es sich eventuell um ein fundamentaleres Problem, weswegen die Architektur vom Modell weiter verändert wird. Das erneute Hinzufügen einer CL (siehe Abbildung 27) und das erstmalige einer weiteren dense-layer hat wie vor dem Entfernen der dropout-layers keine Verbesserung bewirkt.

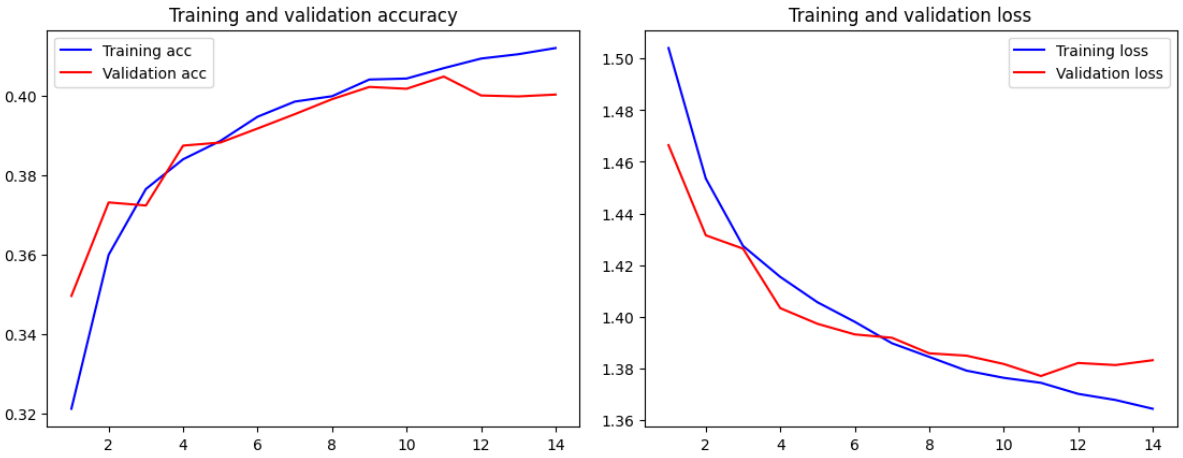

Abbildung 27: Erste Iteration, Genauigkeit bei 4 CL (links), erste Iteration, loss bei 4 CL (rechts) [eigene Darstellung]

Dies lässt darauf schließen, dass bereits in den vorherigen CLs die Extraktion von features ungenügend betrieben wird, wodurch die folgenden Schichten zu wenig Informationen erhalten, um an ihnen verlässlich Muster zu erkennen. Da es sich wie bereits erwähnt um einen komplexen Datensatz handelt, reicht die Filtergröße von 16 in der ersten Schicht eventuell nicht aus, um die Muster des Datensatzes zu erkennen. Hierfür werden statt 16, 32 und 64 Filtern je CL, 64, 96 und 128 verwendet. Die Genauigkeit ist dadurch auf 44,5 % angestiegen, wie in Abbildung 28 (links) zu erkennen ist.

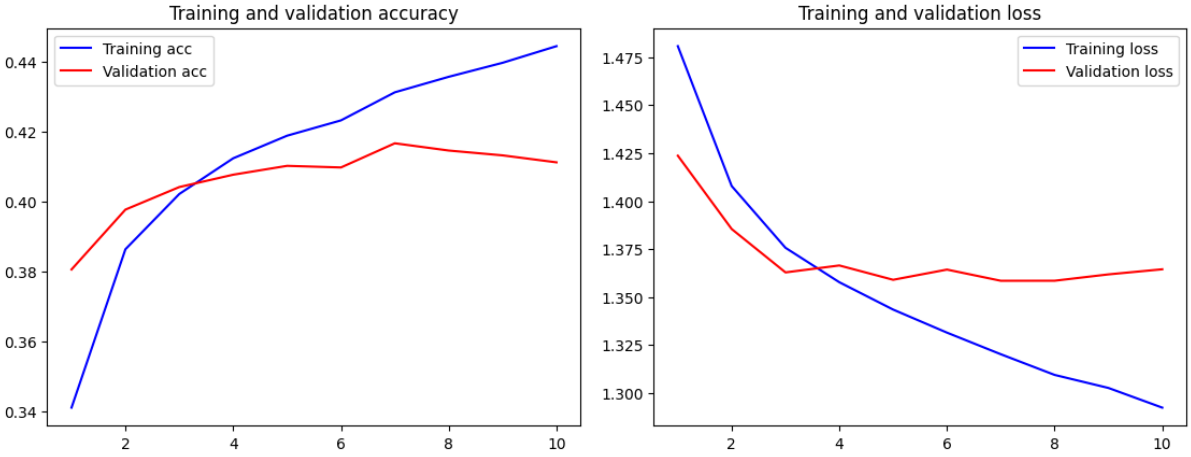

Abbildung 28: Erste Iteration, Genauigkeit bei 64 / 96 / 128 Filtern (links), erste Iteration, loss bei 64 / 96 / 128 Filtern (rechts) [eigene Darstellung]

Nach einer weiteren Erhöhung der Filterzahl auf 128, 160 und 192 hat sich die Genauigkeit auf 46 % erhöht, wie in Abbildung 29 zu sehen ist. Allerdings hat sich der validation-loss zur vorherigen Filterzahl nicht verbessert. Sollte ein Modell mit hoher Genauigkeit bei den Trainingsdaten entwickelt werden

können, muss dementsprechend erneut Regularisierung in Form von dropout eingebaut werden. Eine weitere Erhöhung der Filter hat allerdings keine Fortschritte erzielt.

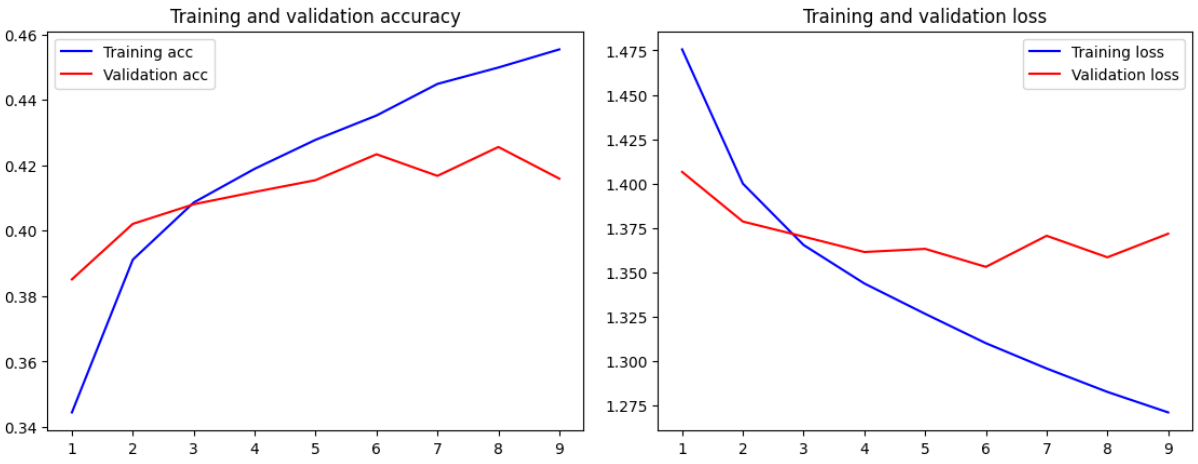

Abbildung 29: Erste Iteration, Genauigkeit bei 128 / 160 / 192 Filtern (links), erste Iteration, loss bei 128 / 160 / 192 Filtern (rechts) [eigene Darstellung]

Da die meisten Parameter des Modells bereits auf Verbesserungen getestet worden sind, kann noch das BERT-Modell während des Trainings angepasst werden, damit das Embedding mit den verwendeten Daten weiter trainiert wird und somit auf sie angepasst wird. Dadurch hat sich die Genauigkeit allerdings verringert. Ebenso hat es nach der Erhöhung der Anzahl der Neuronen der vorletzten dense-layer auf 256 und 512, der initialen Lernrate des Adam Optimizers auf $lr = 0.01$, der Filtergröße auf fünf, dem Hinzufügen von padding und der Erhöhung Anzahl an Daten pro Iteration auf 20% des Datensatzes keine Verbesserung gegeben. Dies lässt darauf schließen, dass die maximale Leistung des Modells in dieser Form erreicht worden ist.

4.6.3 FastText als word embedding

Um deshalb auszuschließen, dass die von BERT entwickelten, dynamischen embeddings zu viel Varianz und Komplexität in das Modell schleusen, wird stattdessen fastText verwendet. Somit wird jedes Wort in jedem Kontext durch den gleichen Vektor repräsentiert und das Modell kann eventuell Muster besser erlernen. Wie in Abbildung 30 und 31 zu erkennen, hat sich die Genauigkeit des Modells dadurch nicht verbessert, was die ursprüngliche These bestätigt, dass die maximale Leistung des Modells erreicht wurde. Allerdings hat sich die Dauer des Trainingsprozesses um das 30-fache verringert, da die embeddings nicht mehr während des Trainings dynamisch generiert werden.

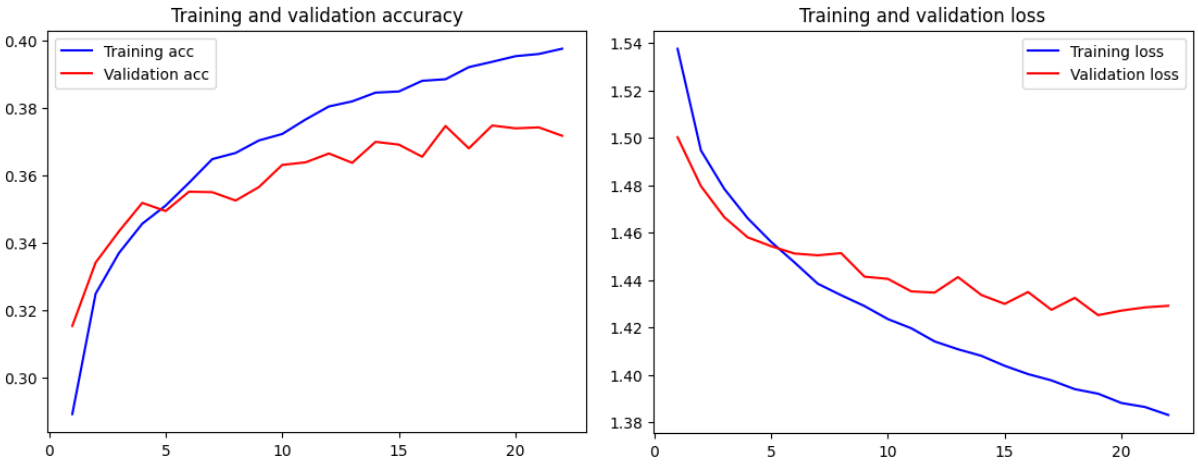

Abbildung 30: fastText Embedding, erste Iteration, Genauigkeit (links), erste Iteration, loss (rechts) [eigene Darstellung]

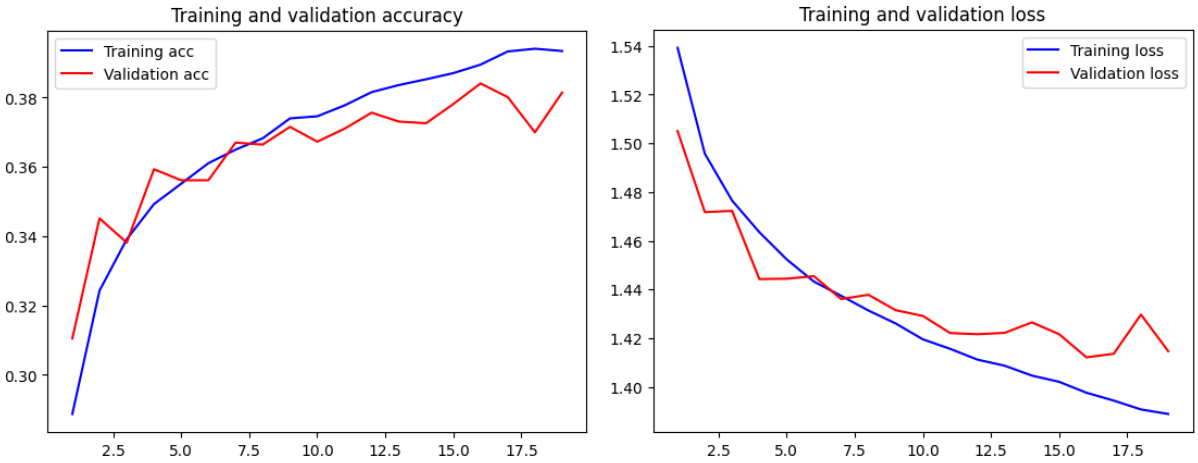

Abbildung 31: fastText Embedding, zweite Iteration, Genauigkeit (links), zweite Iteration, loss (rechts) [eigene Darstellung]

Da der Unterschied der Leistung des BERT-Modells im Vergleich zu fastText nur wenige Prozente ausmacht, fastText eine deutlich schnellere Trainingsdauer aufweist und noch kein Modell mit dem gesamten Datensatz trainiert worden ist, wird fastText verwendet, um mit dem gesamten Datensatz zu trainieren. Sollte sich die Genauigkeit maßgeblich verbessern, wird dasselbe ebenfalls mit BERT durchgeführt.

Anhand Abbildung 32 ist zu erkennen, dass nach einer Trainingsiteration über den gesamten Datensatz sich die Genauigkeit nicht verbessert und eine untypisch höhere Genauigkeit und geringerer loss beim Validierungsset auftritt. Dementsprechend wird das Modell mit BERT nicht mit dem gesamten Datensatz trainiert.

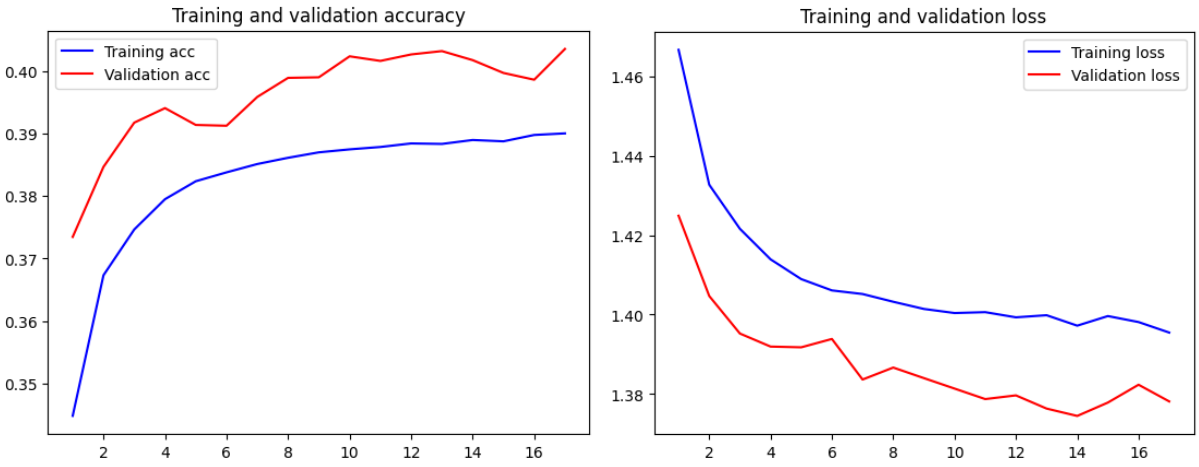

Abbildung 32: fastText Embedding, erste Iteration, Genauigkeit (links), erste Iteration, loss (rechts) [eigene Darstellung]

4.6.4 Anpassung der Datenselektion

Eine weitere Ursache für die geringe Genauigkeit des Modells könnte aus den Daten selbst hervorgehen. In Kapitel 4.2 wird die minimale Anzahl Wörter pro Tweet auf fünf festgelegt. Stattdessen wird sie bei diesem Versuch erhöht. Tweets mit einer Wortanzahl bis kleiner elf nehmen verhältnismäßig einen deutlich höheren Anteil des gesamten Datensatzes ein als die bereits Entfernten (siehe Abbildung 16). Allerdings beinhalten diese eventuell stets zu wenige Informationen. Deshalb werden Tweets mit maximal zehn Wörtern entfernt, was ungefähr 400.000 Tweets ausmacht. Dies kann allerdings Vor- und Nachteile mit sich bringen. Somit könnte einerseits durch weniger Tweets mit wenig Informationen Genauigkeit gewonnen, andererseits durch weniger Tweets insgesamt, Genauigkeit verloren werden.

Wie in Abbildung 33 dargestellt, hat sich dadurch die Genauigkeit zwar minimal verbessert, allerdings reicht diese Erhöhung nicht aus, um dies auf das BERT-Modell zu übertragen und die Vorhersagen mit geringer Genauigkeit zu verwenden.

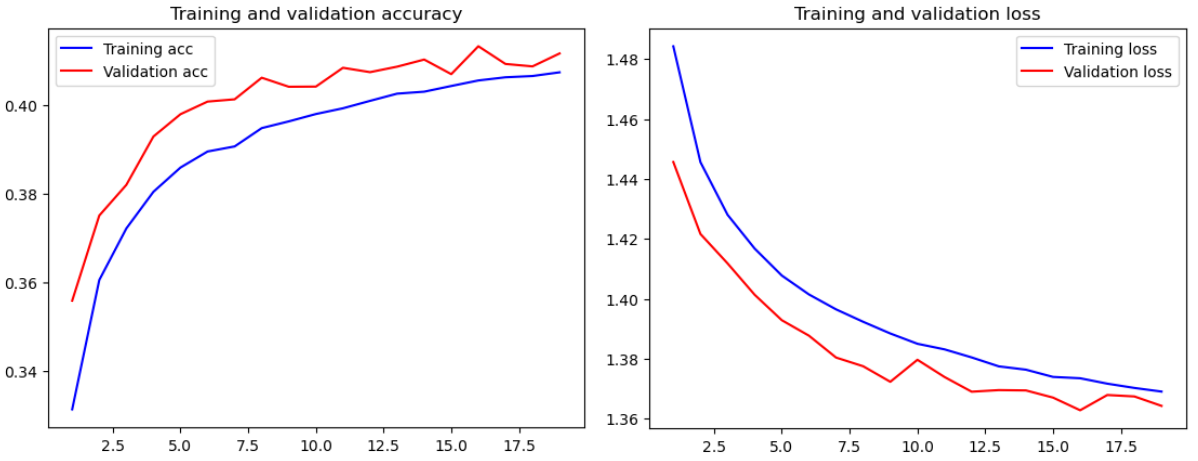

Abbildung 33: fastText, erste Iteration, Genauigkeit (links), erste Iteration, loss (rechts) [eigene Darstellung]

4.6.5 Modell pro Klasse

Statt die bestehende Architektur zu verändern, ist es möglich, die Klassifizierungsaufgabe von einem Modell auf fünf Modelle zu verteilen. Statt ein Modell die Unterschiede zwischen fünf Klassen erlernen zu lassen, wofür dessen Kapazität eventuell nicht ausreicht, kann die Klassifizierungsaufgabe durch die Verteilung auf unterschiedliche Modelle eventuell vereinfacht werden. In jedem der Modelle würde stattdessen eine binäre Klassifizierung stattfinden. Somit sagt das jeweilige Modell nur voraus, ob der Tweet der jeweiligen Partei angehört oder nicht.

Die Labels des Datensatzes werden hierfür auf die binäre Klassifikation angepasst. Die vorherzusagende Partei erhält ihren Parteinamen und die restlichen Parteien erhalten „OTHER“ als Label. Für die erste Iteration werden 33 % des Datensatzes verwendet, was durch die kürzere Trainingszeit mit fastText ermöglicht wird. Zusätzlich repräsentiert diese Menge den gesamten Datensatz somit besser.

Anhand Abbildung 34, in der die Metriken für die Grünen und anhand Abbildung 35, in der die Metriken für die FDP dargestellt sind, ist eine deutlich höhere Genauigkeit von 80-81 % aufgetreten. Bei beiden Parteien liegt die Genauigkeit und der loss in einem ähnlichen Bereich. Dies könnte auf die gemeinsame Abhängigkeit eines Großteils der Daten der „OTHER“ Klasse zurückzuführen sein, die jeweils alle anderen Parteien im Datensatz darstellen.

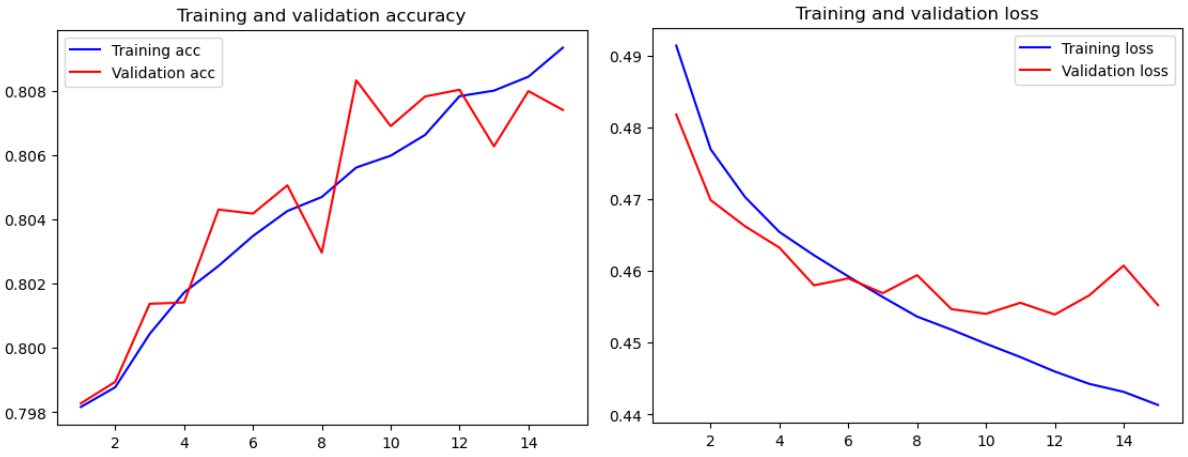

Abbildung 34: fastText, erste Iteration, Partei: Grüne, Genauigkeit (links), erste Iteration, Partei: Grüne, loss (rechts) [eigene Darstellung]

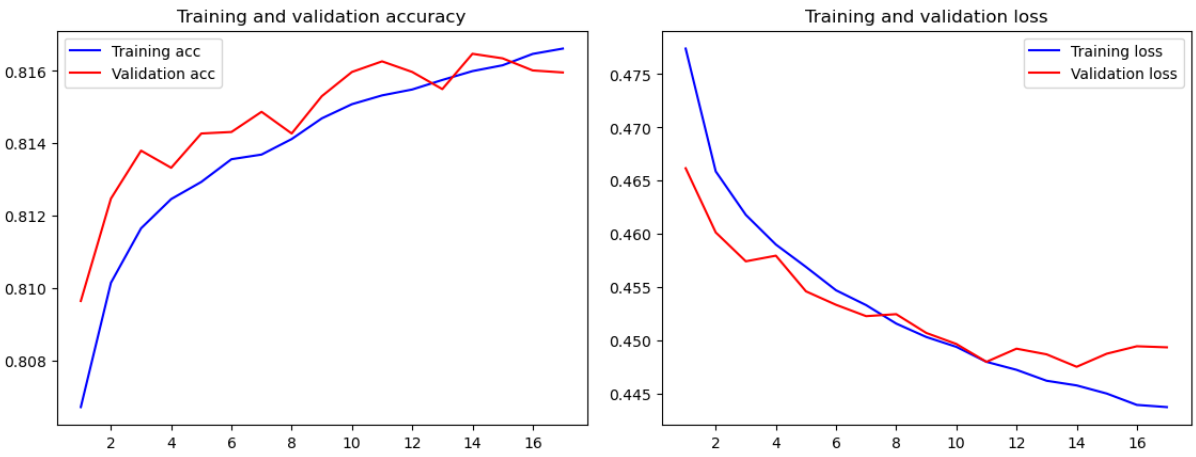

Abbildung 35: fastText, erste Iteration, Partei: FDP, Genauigkeit (links), erste Iteration, Partei: FDP, loss (rechts) [eigene Darstellung]

Wie am Anfang von Kapitel 4.6 müssen dementsprechend die Genauigkeiten in den Klassen selbst untersucht werden, da die hohe Genauigkeit von der dominierenden „OTHER“ Klasse verursacht werden könnte, ohne dass bei Vorhersagen der eigentlichen Partei hohe Genauigkeiten erzielt werden. Anhand der Abbildung 36 ist die confusion-Matrix dargestellt, die die Vorhersagen des Modells für das Testset aufweist. Wie erwartet wird die eigentliche Partei kaum korrekt vorhergesagt, sondern hauptsächlich verwechselt.

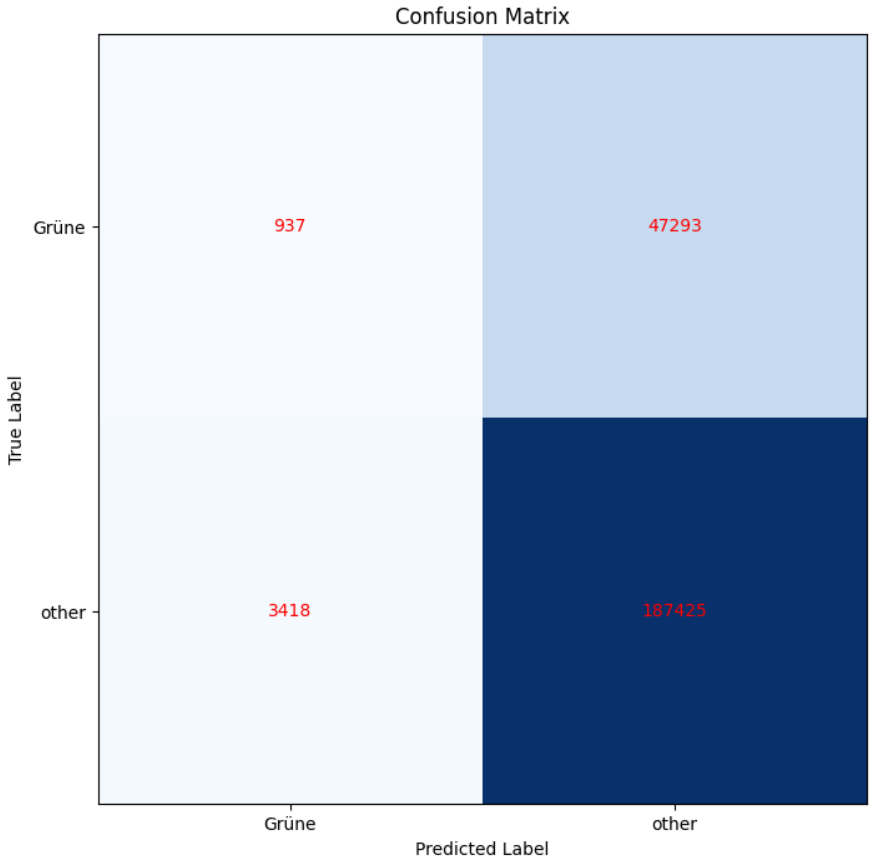

Abbildung 36: Confusion-Matrix des Testsets des Modells der Grünen [eigene Darstellung]

Eventuell sorgt der unausgeglichene Datensatz für die Tendenz, weswegen der Datensatz, wie am Anfang von Kapitel 4.6, durch undersampling angeglichen wird. Die Genauigkeit dieses Trainingsdurchlaufs ist in Abbildung 37 dargestellt. Sie ist stark gesunken und die Leistung beim Validierungsdatensatz liegt deutlich unter der vom Trainingsdatensatz.

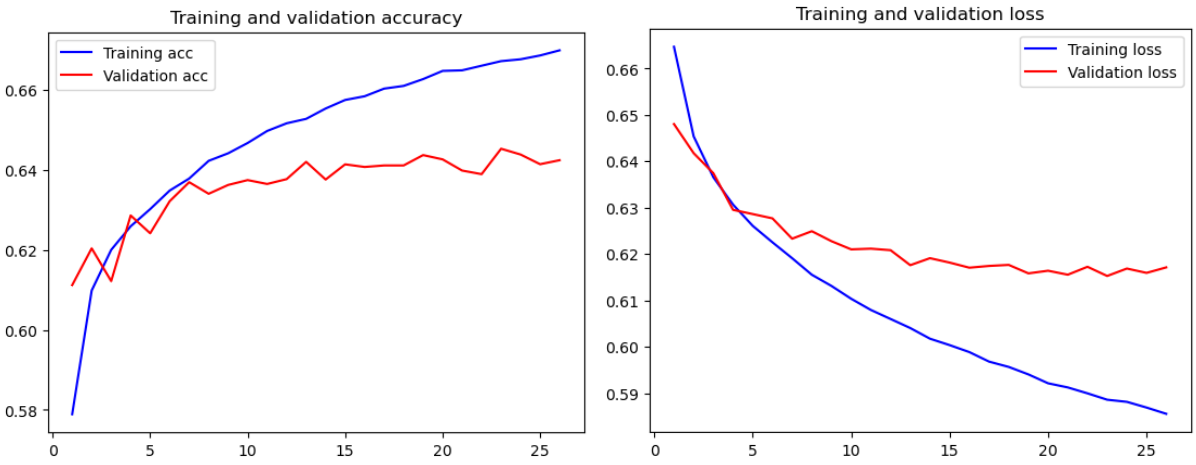

Abbildung 37: fastText, erste Iteration, Genauigkeit (links), erste Iteration, loss (rechts), Partei: Grüne, mit undersampled Datensatz [eigene Darstellung]

Nach Angleichen des Datensatzes hat sich die Tendenz zur „OTHER“ Klasse nicht verringert (siehe Abbildung 38). Die Leistung bei der eigentlichen Partei hat sich ebenfalls kaum zur vorherigen Matrix verändert.

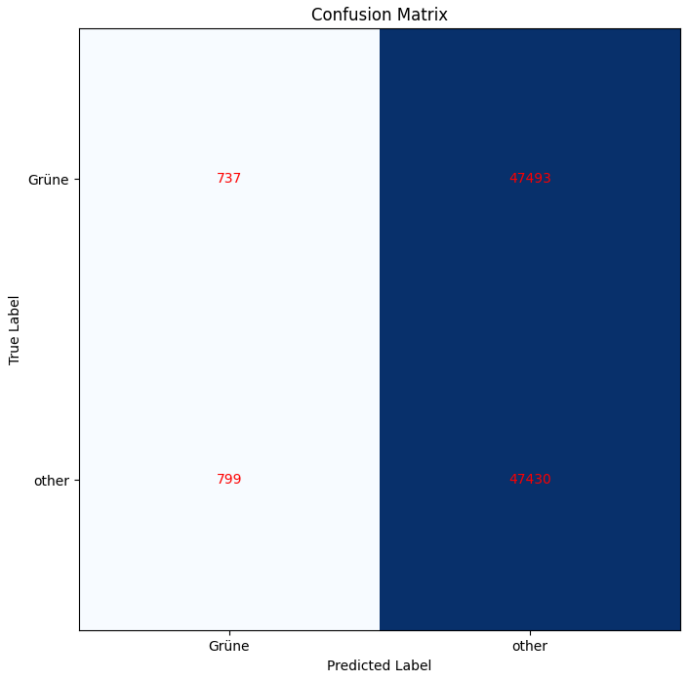

Abbildung 38: Confusion-Matrix des Testsets des Modells der Grünen, mit undersampled Datensatz [eigene Darstellung]

Somit hat auch das Aufteilen des Modells keine Verbesserung erzielt. Dementsprechend werden das Anpassen und Verändern des Modells hiermit eingestellt, da bereits die meisten Hyperparameter verändert und unterschiedliche Architekturen ohne maßgebliche Verbesserungen verwendet worden sind.

4.7 Erkenntnisse

4.7.1 Zusammenfassung der Ergebnisse

Der erste Trainingsdurchlauf hat gezeigt, dass das Modell nur eine geringe Genauigkeit von 41 % erzielt. Um den Ursprung dieses niedrigen Werts zu untersuchen, wurden iterativ Hyperparameter des initialen Modells angepasst, um die jeweiligen Auswirkungen der Parameter auf die Genauigkeit zu analysieren und eine ideale Kombination dieser zu ermitteln. Anschließend wurde die Vorverarbeitung der Daten durch das Entfernen kürzerer Tweets verändert, wodurch ebenfalls nur wenige Prozente gewonnen wurden. Deshalb wurde die Modellarchitektur angepasst, indem die Anzahl an Schichten verändert und das Modell aufgeteilt wurde, wodurch ebenfalls keine höhere Genauigkeit erzielt wurde. Das Modell mit der höchsten Genauigkeit erzielt einen Wert von 46 %.

4.7.2 Interpretation der Ergebnisse

Da das Anpassen einer Vielzahl Hyperparameter und die Veränderung der Architektur nur unwesentliche Verbesserungen erzielt haben, ist die logische Schlussfolgerung, dass die Ursache dafür der Datensatz oder die Art des NN ist, da diese im Verlauf der iterativen Anpassungen nur wenig oder nicht angepasst worden sind. Da ein CNN nach intensiver Recherche für den Theorieteil allerdings als geeignet für diese Problemstellung im Bereich text classification klassifiziert wird, ist es unwahrscheinlich, dass dies der Grund für die niedrige Genauigkeit ist. Dementsprechend ist es naheliegend, dass die Ursprungsdaten nicht für die Lösung der Problemstellung geeignet sind oder die an ihnen vorgenommene Datenvorverarbeitung verbessert werden kann.

4.7.3 Empfehlungen für weiterführende Forschung

Da die Ursache für die geringe Genauigkeit des Modells nicht eindeutig bestimmt, sondern nur eingegrenzt werden kann, könnte bei einer Weiterführung dieser Forschung an der bereits definierten Eingrenzung angesetzt werden. Da nicht ausgeschlossen werden kann, dass es eine andere Art eines NNs gibt, die eine höhere Genauigkeit als ein CNN erzielen kann, könnte erneut untersucht werden, ob ein CNN das beste NN für diesen Anwendungsfall ist. Hierbei könnten insbesondere die in Kapitel 3.6.1 erwähnten Spezialformen von CNNs verwendet oder nach weiteren Architekturen gesucht werden, die nicht in dieser Arbeit erwähnt worden sind. Des Weiteren kann der Ansatz der Datenvorverarbeitung verfolgt und eine detailliertere Analyse der benötigten Vorverarbeitungsschritte vorgenommen werden. Hierbei hat das Anpassen des Entfernens kurzer Tweets bereits Genauigkeit dazugewonnen, weshalb insbesondere an dieser Stelle weiter geforscht werden könnte. Zudem ist aus Zeitgründen undersampling verwendet worden, weshalb bei zukünftiger Forschung auch die Verwendung von oversampling in Betracht gezogen werden könnte.

5 Fazit

Um das Ziel dieser Arbeit zu erreichen, nämlich die Entwicklung eines Algorithmus zur Klassifizierung von Twitter Beiträgen deutscher PolitikerInnen, wurden im zweiten Kapitel Begrifflichkeiten, Themenbereiche und Methoden, die in Zusammenhang mit KI, ML und neuronalen Netzwerken stehen, in Bezug zur Problemstellung dargestellt und erläutert. Im darauffolgenden dritten Kapitel wurden die Methoden für die jeweiligen KDD-Schritte ausgewählt, die für die Implementierung des NNs verwendet wurden. Diese umfassen für die Datensammlung die Verwendung der Bibliothek snscrape, um die Twitter API anzufragen und Tweets der ausgewählten Parteien auszulesen. Von diesen Tweets wurden anschließend mittels Datenselektion fremdsprachige und kurze Tweets mit einer Wortanzahl kleiner fünf entfernt. Durch die Datenvorbearbeitung wurden Störfaktoren wie Sonderzeichen entfernt, der unausgeglichene Datensatz mittels undersampling angepasst und die Daten in Trainings-, Test- und Validierungsdatensätze aufgeteilt. Die Datentransformation wurde mit dem im Konzept festgelegten dbmdz-BERT-Modell durchgeführt, um anschließend mit den daraus entstandenen embeddings das NN zu trainieren. Die Architektur des Modells beläuft sich auf eine Kombination des BERT-Modells und einer angepassten Variante des KimCNN [Ki 2014].

Da das entwickelte Modell eine geringe Genauigkeit von 41 % in der ersten Trainingsiteration aufgewiesen hat, wurden Hyperparameter angepasst, wie die Filtergrößen der CLs oder die Neuronen der dense-layers, wodurch nur geringe Genauigkeitsverbesserungen erzielt wurden. Da aus diesen Anpassungen keine maßgeblichen Verbesserungen resultierten, wurde die Architektur des NN angepasst, indem die Anzahl an CLs, dense- und dropout-layers verändert wurde. Zusätzlich wurde der Algorithmus für das Erstellen der embeddings auf fastText umgestellt, um zu analysieren, ob das dynamische BERT embedding zu viel Varianz in das NN speist. Auch durch diese Anpassungen hat sich die Genauigkeit nicht verbessert. Eine Anpassung der Datenselektion auf minimal zehn Wörter pro Tweet hat die Genauigkeit allerdings leicht erhöht. Des Weiteren wurde das Modell den fünf Klassen entsprechend aufgeteilt, mit der Hoffnung, dass die jeweiligen Modelle eine bessere Leistung in einer binären Klassifikationsaufgabe erreichen. Dies hat ebenfalls keine Verbesserung bewirkt und auch nach Angleichen des Datensatzes tendierte das Modell zu einer Klasse. Im Endeffekt hat das Modell mit der höchsten Genauigkeit 46 % der Tweets korrekt vorhergesagt, was für eine zuverlässige Vorhersage der Resonanzen eines Tweets in den unterschiedlichen Parteien nicht ausreicht.

Anhand des entwickelten Modells ist aufgezeigt worden, dass die Problemstellung mit einem Modell dieser Form und den vorhandenen Daten nicht zu lösen ist. Die Gründe, die dafürsprechen, sind die geringe Genauigkeit, die geringe Verbesserung der Leistung des Modells nach dem Ausbalancieren des unausgeglichenen Datensatzes, das Anpassen einer Vielzahl Hyperparameter sowie das Verändern der Modellarchitektur, um dem Modell mehr Möglichkeiten zum Erlernen von Mustern der Daten zu bieten. Aus diesen Ergebnissen wird geschlussfolgert, dass die unzureichende Genauigkeit nicht auf die veränderten Parameter zurückzuführen ist. Stattdessen können allerdings die nicht oder nur wenig während

Kapitel 4.6 angepassten Parameter, wie die Art des neuronalen Netzwerks und die Ursprungsdaten, Gründe für die geringe Genauigkeit sein. Da ein CNN im Theorieteil allerding als geeignet für die Lösung der Problemstellung klassifiziert wurde, wurde es beibehalten. Dadurch kann aber nicht ausgeschlossen werden, dass es eine Art neuronales Netzwerk gibt, welches eine höhere Genauigkeit erzielen kann. Dies müsste in einer Fortführung der Forschung analysiert und überprüft werden. Da keine Anpassung am Modell zu einer ausreichenden Genauigkeit geführt hat, kann der Grund für die geringe Genauigkeit allerdings auch bei den Ursprungsdaten liegen. Da dies im Verlauf der Arbeit nicht bestätigt werden konnte, müsste es in zukünftiger Forschung verfolgt werden. Bei einer Weiterführung der Arbeit könnte deshalb der Ansatz des Bearbeitens der Trainingsdaten weiter untersucht werden. Hierbei sind insbesondere das Entfernen kurzer Tweets und die verwendete Methode zum Ausgleichen des Datensatzes interessant, da das Entfernen zu einer Steigerung der Genauigkeit geführt hat und der Datensatz lediglich aus Zeitgründen mit undersampling angepasst wurde. Zwar lagen nach dem Anwenden von undersampling stets ausreichende Datenmengen vor, womit eine Klassifizierung in der Regel möglich ist, allerdings kann die Verwendung von oversampling bzw. SMOTE die Genauigkeit des Modells steigern, da keine Daten verloren gehen.

Literaturverzeichnis

[AIML 2017] *Word2Vec word embedding tutorial in Python and TensorFlow* (2017, 22. Juli). Adventures in Machine Learning. Abgerufen am 14. Juni 2023, von https://adventuresinmachinelearning.com/word2vec-tutorial-tensorflow/

[AWS 2023b] *Get Started with Amazon SageMaker* (2023). Amazon Web Services, Inc. Abgerufen am 10. Juli 2023, von https://docs.aws.amazon.com/sagemaker/latest/dg/gs.html

[AWS 2023a] *Machine Learning – Amazon Web Services*. (o. D.). Amazon Web Services, Inc. Abgerufen am 15. Juni 2023, von https://aws.amazon.com/de/sagemaker/

[AML 2023] *ML as a Service | Microsoft Azure*. (o. D.). Azure Machine Learning. Abgerufen am 11. August 2023, von https://azure.microsoft.com/en-us/products/machine-learning

[AßCoHe 2021] Aßenmacher, M., Corvonato, A. & Heumann, C. (2021, 24. Februar). *Re-Evaluating GermEval17 Using German Pre-Trained Language Models*. arXiv.org. Abgerufen am 22. August 2023, von https://arxiv.org/abs/2102.12330

[BoGrJoMi 2017] Bojanowski, P., Grave, E., Joulin, A. & Mikolov, A. (2016, 15. Juli). *Enriching Word Vectors with Subword Information*. arXiv.org. Abgerufen am 15. Juni 2023, von https://arxiv.org/abs/1607.04606v2

[BoKiYo 2021] Boutet, A., Kim, H. & Yoneki, E. (2021, 3. August). *What's in your tweets? I know who you supported in the UK 2010 general election*. Proceedings of the International AAAI Conference on Web and Social Media. Abgerufen am 13. Juli 2023, von https://doi.org/10.1609/icwsm.v6i1.14283

[Br 2017b] Brownlee, J. (2019, 7. August). *What Are Word Embeddings for Text?* machinelearningmastery. Abgerufen am 15. Juni 2023, von https://machinelearningmastery.com/what-are-word-embeddings/

[Br 2017a] Brownlee, J. (2017, 9. Oktober). *A Gentle Introduction to the Bag-of-Words Model*. Machine Learning Mastery. Abgerufen am 13. Juni 2023, von https://machinelearningmastery.com/gentle-introduction-bag-words-model/

[Br 2020d] Brownlee, J. (2020, 17. April). *How Do Convolutional Layers Work in Deep Learning Neural Networks?* MachineLearningMastery.com. Abgerufen am 8. Juli 2023, von https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/

[Br 2020g] Brownlee, J. (2020, 15. August). *8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset*. MachineLearningMastery.com. Abgerufen am 16. Juli 2023, von https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

[Br 2020a] Brownlee, J. (2020, 19. August). *Difference Between Algorithm and Model in Machine Learning.* machinelearningmastery. Abgerufen am 12. Juni 2023, von https://machinelearningmastery.com/difference-between-algorithm-and-model-in-machine-learning/

[Br 2020b] Brownlee, J. (2020, 20. August). *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. MachineLearningMastery.com. Abgerufen am 7. Juli 2023, von https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

[Br 2020c] Brownlee, J. (2020, 26. August). *Train-Test Split for Evaluating Machine Learning Algorithms*. MachineLearningMastery.com. Abgerufen am 4. Juli 2023, von https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/

[Br 2020f] Brownlee, J. (2020, 28. August). *How to Control the Stability of Training Neural Networks With the Batch Size*. MachineLearningMastery.com. Abgerufen am 15. Juli 2023, von https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/

[Br 2020e] Brownlee, J. (2020, September 12). *Understand the Impact of Learning Rate on Neural Network Performance*. MachineLearningMastery.com. Abgerufen am 9. Juli 2023, von https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

[Br 2021a] Brownlee, J. (2021, 5. Januar). *Random oversampling and undersampling for imbalanced classification*. MachineLearningMastery.com. Abgerufen am 11. August 2023, von https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/

[Br 2021b] Brownlee, J. (2021, 13. Januar). *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. MachineLearningMastery.com. Abgerufen am 9. Juli 2023, von https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[Br 2022] Brownlee, J. (2022, 12. August). *When to Use MLP, CNN, and RNN Neural Networks.* machinelearningmastery. Abgerufen am 22. Juni 2023, von https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/

[BPB 2023] Bundeszentrale für politische Bildung (2023). *Wahl-O-Mat*. bpb.de. Abgerufen am 12. Juni 2023, von https://www.bpb.de/themen/wahl-o-mat/

[Bu 2022] Burchfiel, A. (2022, 16. Mai). *What is NLP (Natural Language Processing) Tokenization?* tokenex. Abgerufen am 12. Juni 2023, von https://www.tokenex.com/blog/ab-what-is-nlp-natural-language-processing-tokenization/

[ChMöPiSo 2019] Chan, B., Möller, T., Piertsch, M. & Soni, T. (2019, 14. Juni). *German BERT | State of the Art Language Model for German NLP*. deepset GmbH. Abgerufen am 6. Juli 2023, von https://www.deepset.ai/german-bert

[ChBoHaKe 2002] Chawla, N. V., Bowyer, K. W., Hall, L. J. & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, *16*, 321–357. Abgerufen am 11. August 2023, von https://doi.org/10.1613/jair.953

[Ch 2020] Cheng, H. J.-H. (2020). *Empirical Study on the Effect of Zero-Padding in Text Classification with CNN*. Abgerufen am 8. Juli 2023, von https://escholarship.org/uc/item/7bc9c7jp

[Co 2022] Coursera (2022, 14. September). *3 Types of Machine Learning You Should Know*. Coursera. Abgerufen am 17. Juni 2023, von https://www.coursera.org/articles/types-of-machine-learning

[DaYaYaCaLeSa 2019] Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V. & Salakhutdinov, R. (2019, 9. Januar). *Transformer-XL: Attentive Language models beyond a Fixed-Length context*. arXiv.org. Abgerufen am 17. August 2023, von https://arxiv.org/abs/1901.02860

[DataRobot 2022] *What are Feature Variables in Machine Learning - DataRobot AI Cloud Wiki*. (2022, 20. Juli). DataRobot AI Cloud. Abgerufen am 12. Juni 2023, von https://www.datarobot.com/wiki/feature/

[DaFaAuGr 2016] Dauphin, Y. N., Fan, A., Auli, M. & Grangier, D. (2016, 23. Dezember). *Language Modeling with Gated Convolutional Networks*. arXiv.org. Abgerufen am 7. Juli 2023, von https://arxiv.org/abs/1612.08083

[Dc 2021] *What is noise in machine learning | DeepChecks*. (2021, 5. August). Deepchecks. Abgerufen am 8. Juli 2023, von https://deepchecks.com/glossary/noise-in-machine-learning/

[De o.D.] Devlin, J. (o. D.). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* [Vorlesungsfolien; Google]. Abgerufen am 3. Juli 2023, von https://nlp.stanford.edu/seminar/details/jdevlin.pdf

[DeChLeTo 2018] Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2018, 11. Oktober). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv.org. Abgerufen am 11. August 2023, von https://arxiv.org/abs/1810.04805

[Dl 2022] De Luca, G. (2022, 15. November). *SVM Vs Neural Network*. Baeldung. Abgerufen am 22. Juni 2022, von https://www.baeldung.com/cs/svm-vs-neural-network

[DhGaWaSo 2022] Dharma, E., Gaol, F., Warnars, H. & Soewito, B. (2022, 31. Januar). *THE ACCURACY COMPARISON AMONG WORD2VEC, GLOVE, AND FASTTEXT TOWARDS CONVOLUTION NEURAL*

NETWORK (CNN) TEXT CLASSIFICATION. Abgerufen am 2. August 2023, von http://www.jatit.org/volumes/Vol100No2/5Vol100No2.pdf

[Dr 2019] Draelos, R. (2019, 15. September). *Best Use of Train/Val/Test Splits, with Tips for Medical Data*. Glass Box. Abgerufen am 2. Juli 2023, von https://glassboxmedicine.com/2019/09/15/best-use-of-train-val-test-splits-with-tips-for-medical-data/

[Fa 1997] Fayyad, U. M. (1997). Knowledge Discovery in Databases: An Overview. In *Springer eBooks* (S. 1–16). Abgerufen am 1. August 2023, von https://doi.org/10.1007/3540635149_30

[Ga 2021] Ganesan, K. (2021, 24. September). *FastText vs. Word2vec: A Quick Comparison*. Kavita Ganesan. Abgerufen am 15. Juni 2023, von https://kavita-ganesan.com/fasttext-vs-word2vec/

[GC 2023] *Produkte für KI und maschinelles Lernen | Google Cloud*. (o. D.). Google Cloud. Abgerufen am 11. August 2023, von https://cloud.google.com/products

[Gr 2021] Gross, D. (2021, 3. Juli). *Tensoren*. Theoretische Physik. Abgerufen am 14. Juli 2023, von https://www.thp.uni-koeln.de/gross/tp-blog/posts/tensoren/

[Gu 2021] Gupta, S. (2021, 13. Dezember). *Understanding Slope Calculation For Backpropagation, The Simple Way*. Medium. Abgerufen am 24. Juni 2023, von https://medium.com/analytics-vidhya/understanding-slope-calculation-for-backpropagation-the-simple-way-d9d99793ac9f

[HaAlRoBi 2018] Hamoud, A. A., Alwehaibi, A., Roy, K. & Bikdash, M. (2018, 1. Januar). *Classifying political tweets using naïve bayes and support vector machines*. Lecture Notes in Computer Science. Abgerufen am 13. Juli 2023, von https://doi.org/10.1007/978-3-319-92058-0_7

[Hi 2022] Hill, T. (2022, 5. Februar). *Part 2: Gradient descent and backpropagation - Towards Data Science*. Medium. Abgerufen am 24. Juni 2023, von https://towardsdatascience.com/part-2-gradient-descent-and-backpropagation-bf90932c066a

[Hu o.D. a] *Pretrained models*. (o. D.). Hugging Face. Abgerufen am 30. Juni 2023, von https://huggingface.co/transformers/v4.11.3/pretrained_models.html

[Hu o.D. b] *dbmdz/bert-base-german-uncased* (o. D.). Hugging Face. Abgerufen am 1. Juli 2023, von https://huggingface.co/dbmdz/bert-base-german-uncased

[IBM 2021] *IBM Documentation*. (2021, 8. März). IBM. Abgerufen am 13. Juni 2023, von https://www.ibm.com/docs/en/essl/6.1?topic=vectors-sparse-vector

[JoZh 2017] Johnson, R. & Zhang, T. (2017). *Deep Pyramid Convolutional Neural Networks for Text Categorization*. Abgerufen am 8. Juli 2023, von https://doi.org/10.18653/v1/p17-1052

[JAA 2023] JustAnotherArchivist. (2023, 22. Juni). *GitHub - JustAnotherArchivist/snscrape: A social networking service scraper in Python*. GitHub. Abgerufen am 29. Juni 2023, von https://github.com/JustAnotherArchivist/snscrape

[Ki 2014] Kim, Y. (2014, 25. August). *Convolutional Neural Networks for Sentence Classification*. arXiv.org. Abgerufen am 2. August 2023, von https://arxiv.org/abs/1408.5882

[KiBa 2015] Kingma, D. P. & Ba, J. (2015, 1. Januar). *Adam: A Method for Stochastic Optimization*. arXiv (Cornell University); Cornell University. Abgerufen am 9. Juli 2023, von https://doi.org/10.48550/arxiv.1412.6980

[La 2022] Lahera, G. (2022, 30. März). *Unbalanced Datasets & What To Do About Them - Strands Tech Corner - Medium*. Medium. Abgerufen am 16. Juli 2023, von https://medium.com/strands-tech-corner/unbalanced-datasets-what-to-do-144e0552d9cd

[La 2021] *Langdetect*. (2021, 7. Mai). PyPI. Abgerufen am 29. Juni 2023, von https://pypi.org/project/langdetect/

[Lc 1998] LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. Abgerufen am 25. Juni 2023, von https://doi.org/10.1109/5.726791

[LiOtGoDu 2019] Liu, Y., Ott, M., Goyal, N. & Du, J. (2019, 26. Juli). *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. arXiv.org. Abgerufen am 17. Juni 2023, von https://arxiv.org/abs/1907.11692v1

[Me 2021] Mei, T. (2021, 15. Dezember). *From static embedding to contextualized embedding - Ted Mei - medium*. Medium. Abgerufen am 17. August 2023, von https://ted-mei.medium.com/from-static-embedding-to-contextualized-embedding-fe604886b2bc

[Ms 2021] M, S. (2021, 20. Juli). *Feature Extraction and Embeddings in NLP: A Beginners guide to understand Natural Language Processing*. Analytics Vidhya. Abgerufen am 12. Juni 2023, von https://www.analyticsvidhya.com/blog/2021/07/feature-extraction-and-embeddings-in-nlp-a-beginners-guide-to-understand-natural-language-processing/

[MiChCoDe 2013] Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013, 16. Januar). *Efficient Estimation of Word Representations in Vector Space*. arXiv.org. Abgerufen am 15. Juni 2023, von https://arxiv.org/abs/1301.3781

[Ml 2020] *What Is Natural Language Processing*. (2020, 26. Februar). MonkeyLearn Blog. Abgerufen am 13. Juni 2023, von https://monkeylearn.com/blog/what-is-natural-language-processing/

[Mo o.D.] *Text Classification: What it is And Why it Matters*. (o. D.). MonkeyLearn. Abgerufen am 22. Juni 2023, von https://monkeylearn.com/text-classification/

[Ol 2022] Olu-Ipinlaye, O. (2022, 30. September). *Global Pooling in Convolutional Neural Networks*. Paperspace Blog. Abgerufen am 8. Juli 2023, von https://blog.paperspace.com/global-pooling-in-convolutional-neural-networks/

[Pa 2023] *pandas - Python Data Analysis Library*. (2023). Pandas. Abgerufen am 29. Juni 2023, von https://pandas.pydata.org/

[Pa 2021] Park, J. (2021, 15. Dezember). *Word Embedding in NLP: One-Hot Encoding and Skip-Gram Neural Network*. Medium. Abgerufen am 13. Juni 2023, von https://towardsdatascience.com/word-embedding-in-nlp-one-hot-encoding-and-skip-gram-neural-network-81b424da58f2

[PeSoMa 2014] Pennington, J., Socher, R. & Manning, C. D. (2014). *GloVe: Global Vectors for Word Representation*. Stanford. Abgerufen am 15. Juni 2023, von https://nlp.stanford.edu/pubs/glove.pdf

[Ph 2021] Phi, M. (2021, 14. Dezember). *Illustrated Guide to Transformers- Step by Step Explanation*. Medium. Abgerufen am 17. Juni 2023, von https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0

[Pt 2023] *PyTorch*. (2023). Abgerufen am 17. Juni 2023, von https://pytorch.org/

[RuHiWi 1986] Rumelhart, D. E., Hinton, G. E. & Williams, R. B. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. Abgerufen am 22. Juni 2023, von https://doi.org/10.1038/323533a0

[SaWoYa 1975] Salton, G., Wong, A. & Yang, C. (1975). A vector space model for automatic indexing. *Communications of The ACM*, *18*(11), 613–620. Abgerufen am 13. Juni 2023, von https://doi.org/10.1145/361219.361220

[Sa 2016] Sammons, M. (2016). *EDISON: Feature Extraction for NLP, Simplified*, 4086. ACL Anthology. Abgerufen am 12. Juni 2023, von https://aclanthology.org/L16-1645/

[SK o.D.] *SciKit-Learn: Machine Learning in Python — SciKit-Learn 1.3.0 documentation*. (o. D.). Abgerufen am 11. August 2023, von https://scikit-learn.org/stable/

[Sh 2020] Sharma, P. (2020, 21. April). *MaxPool vs AvgPool*. OpenGenus IQ: Computing Expertise & Legacy. Abgerufen am 9. Juli 2023, von https://iq.opengenus.org/maxpool-vs-avgpool/

[SrHiKrSa 2014] Srivastava, N., Hinton, G., Krizhevsky, A. & Salakhutdinov, R. (2014, 1. Januar). *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. JMLR. Abgerufen am 8. Juli 2023, von https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

[StMaGe 2020] Stark, B., Magin, M. & Geiss, S. (2020). Meinungsbildung in und mit sozialen Medien. In *Springer Reference Sozialwissenschaften* (S. 1–19). Abgerufen am 2. August 2023, von https://doi.org/10.1007/978-3-658-03895-3_23-1

[Ta 2021b] Tam, A. (2021). *Training-validation-test split and cross-validation done right*. MachineLearningMastery.com. Abgerufen am 15. Juli 2023, von https://machinelearningmastery.com/training-validation-test-split-and-cross-validation-done-right/

[Ta 2021a] Tam, A. (2021, 22. Oktober). *A Gentle Introduction to Vector Space Models*. Machine Learning Mastery. Abgerufen am 12. Juni 2023, von https://machinelearningmastery.com/a-gentle-introduction-to-vector-space-models/

[TF o.D.] *Word embeddings*. (o. D.). TensorFlow. Abgerufen am 12. Juni 2023, von https://www.tensorflow.org/text/guide/word_embeddings

[TF 2023a] TensorFlow. (2023). *TensorFlow*. Abgerufen am 17. Juni 2023, von https://www.tensorflow.org/

[TF 2023b] *Introduction to Tensors*. (2023). TensorFlow. Abgerufen am 8. Juli 2023, von https://www.tensorflow.org/guide/tensor

[ToStoKa 2020] Toshevska, M., Stojanovska, F. & Kalajdjieski, J. (2020). Comparative Analysis of Word Embeddings for Capturing Word Similarities, 7. *ResearchGate*. Abgerufen am 6. Juli 2023, von https://www.researchgate.net/publication/341284216_Comparative_Analysis_of_Word_Embeddings_for_Capturing_Word_Similarities

[TUHH o.D.] *Simple tweet preprocessing — Data Quality Explored*. (o. D.). TUHH. Abgerufen am 1. August 2023, von https://www3.tuhh.de/sts/hoou/data-quality-explored/3-2-simple-transf.html

[Tp 2023] *Tweepy*. (2023). Tweepy. Abgerufen am 29. Juni 2023, von https://www.tweepy.org/

[Va 2018] Valadkhani, M. (2018, 9. März). *Knowledge Discovery in Data (KDD) Process*. Abgerufen am 18. Juni 2023, von https://www.linkedin.com/pulse/knowledge-discovery-data-kdd-process-mohammad-valadkhani

[VaShPaUsJoGoKaiPo 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. (2017, 12. Juni). *Attention is all you need*. arXiv.org. Abgerufen am 11. August 2023, von https://arxiv.org/abs/1706.03762

[Vo 2023] Voita, L. (2023, 24. Februar). *Convolutional Models for Text*. Abgerufen am 25. Juni 2023, von https://lena-voita.github.io/nlp_course/models/convolutional.html

[Wa 2020] Wang, J., Turko, R., Shaikh, O., Park, H., Das, N., Hohman, F., Kahng, M. & Chau, P. (2020). *CNN Explainer*. Abgerufen am 25. Juni 2023, von https://poloclub.github.io/cnn-explainer/

[WaLiCaChWa 2019] Wang, R., Li, Z., Cao, J., Chen, T. & Wang, L. (2019, 1. Juli). *Convolutional Recurrent Neural Networks for Text Classification*. IEEE Conference Publication | IEEE Xplore. Abgerufen am 8. Juli 2023, von https://ieeexplore.ieee.org/document/8852406

[WeZhLuWa 2016] Wen, Y., Zhang, W., Luo, R. & Wang, J. (2016, 22. Juni). *Learning text representation using recurrent convolutional neural network with highway layers*. arXiv.org. Abgerufen am 7. Juli 2023, von https://arxiv.org/abs/1606.06905

[YaDaYaCaSaLe 2019] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. & Le, Q. V. (2019, 19. Juni). *XLNET: Generalized Autoregressive Pretraining for Language Understanding*. arXiv.org. Abgerufen am 17. August 2023, von https://arxiv.org/abs/1906.08237

[YiKaYuSc 2017] Yin, W., Kann, K., Yu, M. & Schütze, H. (2017, 7. Februar). *Comparative Study of CNN and RNN for Natural Language Processing*. arXiv.org. Abgerufen am 7. Juli 2023, von https://arxiv.org/abs/1702.01923

[YiScXiZh 2015] Yin, W., Schütze, H., Xiang, B. & Zhou, B. (2015, 16. Dezember). *ABCNN: Attention-Based Convolutional Neural Network for Modeling Sentence Pairs*. arXiv.org. Abgerufen am 7. Juli 2023, von https://arxiv.org/abs/1512.05193

[ZaPaKa 2021] Zakeri-Nasrabadi, M., Parsa, S. & Kalaee, A. (2021, März). *Format-aware learn&fuzz: deep test data generation for efficient fuzzing*. ResearchGate. Abgerufen am 24. Juni 2023, von https://www.researchgate.net/figure/Computational-graph-of-a-recurrent-neural-network-with-one-hidden-layer-29_fig4_342156022

Anhang

A) Beispieltweets vor und nach Bearbeitung

Tabelle 1: Beispieltweets vor Anwendung von preprocessing [eigene Darstellung]

Index	Time	User	Label	Tweet
0	2023-01-21 13:37:27+00:00	Christian Lindner	fdp	Lieber @henninghoene, herzlichen Glückwunsch zur Wahl als neuer Landesvorsitzender der @fdp_nrw! Die freiheitsliebende Mitte in Nordrhein-Westfalen hat eine Stimme. CL
1	2023-01-20 17:57:14+00:00	Christian Lindner	fdp	Im #Bundestag wurde heute unsere #Fachkräftestrategie vorgestellt. Neben zeitgemäßer Ausbildung und gezielter Weiterbildung steht vor allem die qualifizierte Einwanderung in unseren Arbeitsmarkt im Mittelpunkt. Wir können alle fleißigen Hände und jeden klugen Kopf gebrauchen. CL
2	2023-01-21 12:31:11+00:00	Friedrich Merz	cdu	Lieber @SebLechner, ich gratuliere Dir sehr herzlich zu Deiner Wahl zum Landesvorsitzenden der #CDU Niedersachsen. Viel Erfolg und auf gute Zusammenarbeit! (FM)
3	2023-01-19 10:42:00+00:00	Friedrich Merz	cdu	„Als politischer Pate von #JamshidSharmahd fordere ich von der iranischen Regierung einen fairen Prozess. Den Botschafter habe ich erneut aufgefordert, die Öffentlichkeit über dessen Schicksal zu informieren. Ich setze mich weiter für #Sharmahd und seine Freilassung ein.“ (tm)

Tabelle 2: Beispieltweets nach Anwendung von preprocessing [eigene Darstellung]

Index	Time	User	Label	Tweet
0	2023-01-21 13:37:27+00:00	Christian Lindner	fdp	[CLS] lieber @user herzlichen glückwunsch zur wahl als neuer landesvorsitzender der @user die freiheitsliebende mitte in nordrhein westfalen hat eine stimme [SEP] cl url [SEP]
1	2023-01-20 17:57:14+00:00	Christian Lindner	fdp	[CLS] im bundestag wurde heute unsere fachkräftestrategie vorgestellt [SEP] neben zeitgemäßer ausbildung und gezielter weiterbildung steht vor allem die qualifizierte einwanderung in unseren arbeitsmarkt im mittelpunkt [SEP] wir können alle fleißigen hände und jeden klugen kopf gebrauchen [SEP] cl [SEP]

2	2023-01-21 12:31:11+00:00	Friedrich Merz	cdu	[CLS] lieber @user ich gratuliere dir sehr herzlich zu deiner wahl zum landesvorsitzenden der #hash niedersachsen [SEP] viel erfolg und auf gute zusammenarbeit fm [SEP]
3	2023-01-19 10:42:00+00:00	Friedrich Merz	cdu	[CLS] als politischer pate von jamshidsharmahd fordere ich von der iranischen regierung einen fairen prozess [SEP] den botschafter habe ich erneut aufgefordert die öffentlichkeit über dessen schicksal zu informieren [SEP] ich setze mich weiter für sharmahd und seine freilassung ein [SEP] tm url [SEP]

B) Unterschiedliche Codeschnipsel

Codeblock 1: Aufteilen des Datensatzes in Trainings-, Validierungs- und Testset [eigene Darstellung]

```
train_df, test_df = train_test_split(df_bert, test_size=0.1, random_state=42,
stratify=df_bert['label'].values)
train_df, val_df = train_test_split(train_df, test_size=0.1, random_state=42,
stratify=train_df['label'].values)
```

Codeblock 2: Labels in numerischer Form [eigene Darstellung]

```
label_map = {label: i for i, label in enumerate(train_df['label'].unique())}
train_df['label'] = train_df['label'].map(label_map)
test_df['label'] = test_df['label'].map(label_map)
val_df['label'] = val_df['label'].map(label_map)
```

Codeblock 3: Implementation des BERT-Models [eigene Darstellung]

```
inputs = tf.keras.Input(shape=(max_length,), dtype=tf.int32)
input_masks = tf.keras.Input(shape=(max_length,), dtype=tf.int32)

bert_model = TFAutoModel.from_pretrained('dbmdz/bert-base-german-uncased')
bert_model.trainable = False
bert_output = bert_model(inputs, attention_mask=input_masks,)[0]
```

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit mit dem Titel:

_____

selbständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

Datum	Unterschrift