HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Herr
Tim Wetterau

Vergleichende Studie zu Feature-Selektionsalgorithmen zur Klassifikation von gerichteter offensiver Sprache

Mittweida, 2022

Fakultät CB

BACHELORARBEIT

Vergleichende Studie zu Feature-Selektionsalgorithmen zur Klassifikation von gerichteter offensiver Sprache

Autor:
Herr

Tim Wetterau

Studiengang:
Allgemeine und Digitale Forensik

Seminargruppe:
FO19w2-B

Erstprüfer:
Prof. Dr. rer. nat. Dirk Labudde

Zweitprüfer:
Dr. rer. nat. Michael Spranger

Einreichung:
Mittweida, 30.09.2022

Verteidigung/Bewertung:
Mittweida, 2022

Faculty CB

BACHELOR THESIS

Comparative Study on Feature Selection Approaches for Classification of Targeted Offensive Language

author:
Mr.

Tim Wetterau

course of studies:
General and Digital Forensics

seminar group:
FO19w2-B

first examiner:
Prof. Dr. rer. nat. Dirk Labudde

second examiner:
Dr. rer. nat. Michael Spranger

submission:
Mittweida, 30.09.2022

defence/ evaluation:
Mittweida, 2022

Bibliografische Beschreibung:

Wetterau, Tim:
Vergleichende Studie zu Feature-Selektionsalgorithmen zur Klassifikation von gerichteter offensiver Sprache. 2022. 6, 53, 7 S.
Mittweida, Hochschule Mittweida, Fakultät CB, Bachelorarbeit, 2022

Referat:

Offensive Sprache im Internet ist ein stark diskutiertes Problem in sozialen Medien. Angriffe richten sich oftmals gegen Einzelpersonen, können aber auch auf Gruppen und andere Strukturen abzielen. Die Erkennung angreifender Inhalte funktioniert in vielen Ansätzen bereits sehr gut. Die Erkennung der Ziele hingegen ist bisher nur wenig erforscht. Die vorliegende Arbeit befasst sich mit der Aufarbeitung des aktuellen Forschungsstandes zu offensiver gerichteter Sprache, den Grundlagen derer Erkennung und dem Vergleich verschiedener Ansätze. Die Auswirkungen von Vorverarbeitung und Parametrisierung der Modelle werden analytisch diskutiert.

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ACO	Ant Colony Optimization
ACS	Ant Colony System
AI	Artificial Intelligence
AS	Ant System
AUC	Area under the Curve
BERT	Bidirectional Encodings Representations from Transformers
BiLSTM	Bidirectional Long Short-Term Memory
BOW	Bag of Words
CNN	Convolutional Neural Network
ELMo	Embeddings from Language Models
GPT	Generative Pre-Trained Transformer
GRP	Group (Klasse)
HBBO	Human behavior-based Optimization
IDF	Inverse Dokumentenfrequenz
IND	Individual (Klasse)
MLM	Masked Language Modelling
NLP	Natural Language Processing
NOT	Not offensive (Klasse)
NSP	Next Sentence Prediction
OTH	Others (Klasse)
PMI	Pointwise Mutual Information
POS	Part of Speech
RBF	Radiale Basisfunktion
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristics
SVM	Support Vector Machine
TF	Termfrequenz
TIN	Targeted Insult (Klasse)
UNT	Untargeted (Klasse)
URL	Uniform Resource Locator

1 Einleitung

Das Internet ist heute Bestandteil des Alltags vieler Menschen, besonders der jungen Generationen. Dabei verlagert sich die Kommunikation untereinander weitläufig in den Bereich des World Wide Webs [1]. Plattformen wie Instagram, WhatsApp und Twitter haben einen essenziellen Bestandteil daran. Sie bieten einfache als auch anonyme Möglichkeiten zum schnellen Informationsaustausch. Dies hat unbestritten viele Vorteile, birgt aber ebenso Potenzial, um nahezu anonym böswillige Absichten zu verfolgen. Durch das einfache Erlangen von Anonymität im Internet kommt es immer häufiger zu Vorfällen in denen Nachrichten mit teils offensivem Inhalt versendet werden. Diese Inhalte richten sich gegen eine Vielzahl verschiedener Ziele. Besonders zu bestimmten Themen, Gruppen und Personen werden Posts veröffentlicht, die sich angreifend gegen diese wenden. Hierbei ist besonders wichtig das Ziel zu identifizieren, welches angegriffen wurde.

1.1 Motivation

Die Identifikation der Ziele ist wichtig, um diese besser zu schützen und solcherlei Angriffe verhindern zu können. Bei besonders schweren Angriffen im Fall von Mobbing und Hassrede sind bereits Fälle von psychischen Problemen und Depressionen bekannt [2]. Um diese Art Schäden zu verhindern, muss eine Lösung gefunden werden, welche eine effiziente Eindämmung offensiver Sprache ermöglicht.

Aufgrund des hohen Aufkommens von Nachrichten mit angreifendem Inhalt muss es Möglichkeiten für Betreiber sozialer Netzwerke und Strafverfolgungsbehörden zur Identifikation der Geschädigten geben. Ebenso soll die Community der Onlineplattformen weitgehend vor solchen Fällen geschützt werden. Durch eine manuelle Selektion der einzelnen Nachrichten ist diese Aufgabe nicht zu stemmen. Deshalb ist eine intelligente Lösung notwendig, um die Detektion gerichteter offensiver Sprache flächenabdeckend umzusetzen. Dies kann durch verschiedene Methoden und Algorithmen, wie maschinelles Lernen oder Deep Learning realisiert werden.

1.2 Ziele

Diese Arbeit soll einen Vergleich von ausgewählten Methoden zur Klassifikation von gerichteter offensiver Sprache zeigen. Dabei werden von klassischen Methoden, wie Bag-of-Words und TF-IDF Vektorisierung über Schwarmintelligenzalgorithmen bis hin zu neuronalen Ansätzen Möglichkeiten zum Problem der gerichteten offensiven Sprache dargestellt. Drei Ansätze werden in Experimenten implementiert und deren Performance verglichen. Ziel ist es ein Modell mit dessen Parametern zu definieren, welches auf dem Task der gerichteten offensiven Sprache am besten performt.

Es wird eine ausführliche Literaturdiskussion über bereits bekannte Ansätze in Kapitel 2 durchgeführt. Dazu wird der aktuelle Stand der Analyse von gerichteter offensiver Sprache erörtert und Vorgänger des Aufgabenbereiches erwähnt. In Kapitel 3 sollen die bereits auf-gezeigten Ansätze näher beleuchtet und deren Charakteristiken beschrieben werden. Besonders wird auf die in den Experimenten verwendeten Methoden eingegangen. In Kapitel 4 folgt die Beschreibung des Aufbaus der Experimente und der vorliegenden Daten. Dabei wird auf die Parameter der einzelnen Modelle eingegangen sowie deren Optimierung. Kapitel 5 präsentiert die Ergebnisse der Modelle und diskutiert diese eingehend. In Kapitel 6 werden die Erkenntnisse der Arbeit zusammengefasst.

2 Literaturdiskussion

Die Detektion offensiver Sprache beschäftigt Forscher bereits seit vielen Jahren. Dabei wird das Thema aus linguistischer Sicht betrachtet und Möglichkeiten zur computergestützten Erkennung entwickelt. Offensive Sprache umfasst viele Themengebiete, wie die Erkennung von Aggression, Cybermobbing als auch Hassrede. Die folgenden zwei Unterabschnitte befassen sich mit verwandter Literatur und bereits getätigten Forschungen. Weiterhin soll der Begriff der offensiven Sprache definiert und die Struktur des OLID-Tasks erläutert werden.

2.1 Verwandte Arbeiten

Um bereits getätigte Forschungen umfassend zu analysieren, muss neben aktuellen Arbeiten auch ein Blick auf die ersten Forschungen auf diesem Gebiet geworfen werden. Das Forschungsgebiet er offensiven Sprache kristallisierte sich aus dem Gebiet der Hassrede heraus.

Anfänglich bearbeiteten Razavi et al. [3] das Thema offensive Sprache im Internet. Dabei stellten sie fest, dass sich Kommunikation stetig in Bereiche des Internets verlagert. Sie zeigen auf, dass zum damaligen Zeitpunkt die Erkennung dieser Sprache für Computerprogramme eine schwierige Aufgabe war. Anhand statistischer Modelle und regelbasierten Mustern experimentierten sie mit einer 3-Level-Klassifikation. Die Beispieldaten wurden in zwei Label unterschieden: *Flame* und *Okay.* Die Definition von Flame umfasst viele Formen von missbräuchlich verwendeter Sprache.

2012 richten Warner und Hirschberg [4] den Fokus ihrer Forschung nach der Erkennung von Hassrede in Bezug auf Rassismus. Sie stellen fest, dass Hassrede besonders zwischen Gruppen stattfindet. Interessant ist, dass die offensiv erscheinende Sprache innerhalb Gruppen gleicher Merkmale nicht unbedingt als solche gilt. Beispielsweise das Wort „*Nigga*“ als Abwandlung des N-Wortes indiziert innerhalb Gruppen von afroamerikanischen Individuen Solidarität und hat keinen angreifenden Inhalt. Somit sollte der Kontext der Sprache in die Klassifikation mit einbezogen werden. In der Arbeit wurden ebenso ein Korpus und ein Klassifikationsmodell für antisemitische Sprache erarbeitet.

Wang et al. [5] versuchten das Fluchen (engl. „cursing“) vom realen Leben ins Internet zu projizieren. Fluchen meint in deren Sinne allgemeine und ungerichtete anstößige Sprache. Hierbei stellten sie fest, dass Fluchen im realen Leben immer häufiger vorkommt und besonders junge Menschen dazu tendieren. Des Weiteren fluchen Männer öfters als Frauen und verwenden ebenfalls einen anderen Wortschatz. Im Internet lässt sich dieser Trend ebenfalls verfolgen, wobei es entscheidende Barrieren zu deren Erkennung gibt. Die veränderten Schreibweisen sind eine der größten davon, da hier alle Formen bekannt sein müssten, um sie mit lexikalischen Ansätzen erkennen zu können. Dieser Versuch gelingt den Autoren mit einem F1-Maß von 0.83.

Weitere interessante Publikationen, die sich mit angreifenden Nachrichten im Internet beschäftigen stammen aus 2015 und 2016. Van Hee et al. [6] definierten einen Leitfaden zur Annotation von Datensätzen für Cybermobbing. Hier unterscheiden die Autoren nur zwischen dem Mobbing aus rassistischen und sexistischen Motiven. Nobata et al. [7] verfolgten einen ähnlichen Ansatz, wie [6], teilten nur die Klassen anders ein. Sie unterscheiden Hassrede von verächtlicher Sprache und allgemein missbräuchlich verwendeter Sprache. Verächtliche Sprache wird als angreifend gegen Individuen und Gruppen bezeichnet, welche keine Hassrede darstellen. Waseem und Hovy [8] stellten die Begriffe Hassrede und offensive Sprache gleich. Sie definierten erstmals Kriterien die Hassrede im Internet erfüllen muss. Ziel dessen war eine Vereinfachung der Detektion von dieser Sprachform im Internet. Die Autoren erstellten einen Datensatz von ca. 16.000 Tweets zur annotierten sie ähnlich wie [6] als rassistisch, sexistisch oder kein Angriff. Sie trainierten ein lineare Regressionsmodell und erreichten damit einen F1-Score von 0.73.

Davidson et al. [9] beschäftigten sich 2017 eingehender mit der Trennung von Hassrede und offensiver Sprache. Sie weisen darauf hin, dass in vorangegangenen Forschungen der Anspruch an die Trennung der Gebiete nicht hoch genug gewesen sei und Hassrede nur einen Teil der offensiven Sprache ist. Die Autoren definieren Hassrede als:

> *„Sprache, die verwendet wird, um Hass gegen eine bestimmte Gruppe auszudrücken oder die abwertend, erniedrigend oder beleidigend sein soll, um Mitglieder einer Gruppe zu verletzen.“* [9, S. 1]

Hierbei lässt sich klar erkennen, dass der Begriff offensive Sprache nicht auf diese Definition passt. Auch Cybermobbing und Aggressionen sind dadurch nicht zu identifizieren. Offensive Sprache stellt somit angreifende Sprache dar, die keine Hassrede entsprechend der Definition beinhaltet. Mit dieser Publikation veröffentlichen die Autoren ebenfalls einen neuen Datensatz zur Detektion von Hassrede[1]. Dieser besteht aus 25.000 Tweets und unterteilt sich in die Klassen Hassrede, offensive Sprache und nichts der beiden. Exemplarisch werden in der Arbeit verschiedene maschinelle Lernalgorithmen herangezogen, um eine Klassifikation vorzunehmen. Als Eingabefeature wurden gestemmte Uni-, Bi- und Trigramme mit deren TF-IDF gewichtet verwendet. Die beste Performance liefert dabei eine logistische Regression mit L1 Regularisierung sowie ein SVM mit einem F1-Score von 0.90.

Eine weitere Publikation 2017 von Waseem et al. [10] vertieft das Problem der offensiven Sprache weiter. Deren Ziel war es verschiedene Typologien für verbale Angriffe herauszuarbeiten und entsprechende Feature für maschinelles Lernen zu empfehlen. Erstmals wird unterschieden, ob einzelne Individuen, Entitäten oder generell „Andere“ angegriffen werden. Unter dem Sammelbegriff „Andere“ fallen beispielweise Menschen mit bestimmter Ethnie oder Religion. Nach der Aufteilung durch Waseem et al. Lassen sich vier Typologien unterschieden. Diese teilen sich in explizite und implizite Angriffe auf, die jeweils gerichtet oder ungerichtet sein können. Besonders gerichtete Angriffe lassen sich als Cy-

[1] Datensatz nach [9] verfügbar unter: https://github.com/t-davidson/hate-speech-and-offensive-language

bermobbing bzw. Angriffe gegen Gruppen herausstellen. Um deren Klassifikation und Detektion zu vereinfachen, schlagen die Autoren eine Sammlung von Features nach den Typologien vor. Zur Identifikation von Cybermobbing schlagen sie Benutzererwähnungen, benannte Entitäten und deren Koreferenzresolution vor. Wohingegen lexikalische Ansätze bei Angriffen gegen Gruppen besonders gute Klassifikationen ermöglichen. Waseem et al. verweisen für weitere Arbeiten darauf, dass die Gebiete der offensiven Sprache nach den angegriffenen Zielen getrennt werden sollen.

2019 veröffentlichen Zampieri et al. [11] ein Paper, welches besonders einschlägig für das Forschungsgebiet ist. Dieses baut auf der Empfehlung von [10] auf und trennt verschiedene Formen angreifender Sprache nach deren Ziel. Somit sind Art und Ziel erstmals gemeinsam im Fokus zur genaueren Analyse von gerichteter offensiver Sprache. Um den Anforderungen gerecht zu werden und eine bessere Identifikation der Ziele umzusetzen, wurde ein neues Annotationsschema eingeführt. Mit dem neuen Datensatz OLID (Offensive Language Identification Dataset) wurden etwa 14.000 Tweets nach dem neuen 3-Stufen-Schema annotiert[2]. Dieses Schema umfasst die folgenden Stufen:

A: Detektion von offensiver Sprache
B: Kategorisierung der offensiven Sprache
C: Identifikation der Ziele offensiver Sprache

Der Aufbau des Ansatzes nach Zampieri et al. [11] wird im nächsten Unterkapitel genauer aufgeschlüsselt und detailliert dargelegt. Auf dem erstellten Datensatz wurden mehrere maschinelle Lernalgorithmen trainiert und evaluiert. Dazu wurden ein SVM, BiLSTM und CNN herangezogen. Das CNN übertraf die anderen beiden Modelle und erreichte in allen drei Stufen den Bestwert, welche in Tabelle 1 dargestellt sind. Die Eingabewerte für die SVM waren einfache Unigramme ohne Wichtung mit TF-IDF. Das BiLSTM und CNN hatten jeweils zwei Inputkanäle. Diese setzen sich aus FastText Einbettungen und updatebaren Einbettungen, welche während des Trainings angepasst werden, zusammen.

Tabelle 1: F1-Maße je Subtask von Zampieri et al. [11]

Subtask	**A**	**B**	**C**
F1-Maß	0.80	0.69	0.47

Noch im gleichen Jahr wurde ebenfalls von Zampieri et al. [12] genau dieses Klassifikationsproblem als Shared Task von SemEval ausgeschrieben. Daran nahmen 115 Teams teil, welche eine breite Auswahl an Klassifikationsmodellen auf dem Task testeten. Besonders hervorstechend zeigte sich BERT. Ein Deep Learning Modell, welches auf einer Transformerstruktur basiert und intensiv vortrainiert wird. Genauer wird dieses Modell in Punkt 3.3.5 erläutert. BERT-Modelle sind besonders in den ersten 10 Plätzen der jeweili-

[2] Datensatz nach [ZMN+19a] verfügbar unter: https://scholar.harvard.edu/malmasi/olid

gen Subtasks stark vertreten. Allgemein konnten die Ergebnisse aller Subtasks verbessert werden im Vergleich zu [11]. Eine Übersicht zu den Ergebnissen der einzelnen Subtasks liefert Tabelle 2.

Tabelle 2: F1-Maße je Subtask der besten Teams des SemEval-2019 Task 6 [12]

Subtask	F1-Maß	Modell	Team
A	0.83	BERT	NULI
B	0.76	Regelbasiertes Modell	jhan014
C	0.66	BERT	vradivchev_anikolov

NULI [13] experimentierten mit mehreren Modellen, wobei sich BERT in allen drei Subtasks als bestes herausstellt. Sie segmentierten Hashtags, ersetzten Emojis und verarbeiteten „URL“ und „@USER“ Token. jhan014 [14] verwendeten ein selbstdesigntes Modell zur Klassifikation. Dieses besteht aus einer Einbettungsschicht, sowie vier GRU- und fünf Denseschichten. Dabei basieren die Regeln auf Lexika von angreifenden Wörtern, sowie weiteren Inhalten aus einem Tweet, wie Hashtags und Usermentions. vradivchev_anikolov [15] nutzen mehrere verschiedene Modelle, aus denen sie die besten auswählten. Aus den ausgewählten Modellen assemblierten sie die Vorhersagen. Außerdem entfernten sie Punktation außer ‚#‘ und ‚@‘, segmentierten Hashtags und entfernten Stoppwörter. In Task B und C wurden Personalpronomen behalten, da sie für die Richtung eines Angriffes von Relevanz sin können.

2019 als auch in den darauffolgenden Jahren wurde der Task nach Zampieri et al. [11] für weitere Sprachen adaptiert. Darunter gibt es Publikationen für Arabisch [16], Griechisch [17] und Dänisch [18]. Diese folgen alles dem gleichen Annotationsschema und erstellen mit diesem einen neuen Datensatz mit Tweets der jeweiligen Sprache.

Eine wegweisende Publikation in Form einer kritischen Analyse der bisher betrachteten Ansätze im Bereich angreifender Sprache veröffentlichten Nakov et al. [19]. Sie stellten fest, dass bisherige Arbeiten oft auf eine Online-Plattform konzentriert sind. Viele Forschungen beziehen sich auf Twitter. Dabei ist zu beachten, dass jede Online-Plattform offensive Sprache bzw. auch Hassrede anders definiert. Dazu verglichen sie die Policen von Plattformen, wie Facebook und Twitter, sowie die verschiedenen Typen dieser Plattformen, wie soziale Medien, Video oder Gaming-Community. Weiterhin heben sie Probleme hervor, wie die Entwicklung neuer Ausdrücke, Kontextverständnis und die Spezifität eines Datensatzes zu einem bestimmten Thema. Sie empfehlen das maschinelle Lernen von angreifender Sprache als repetitiven Prozess zu designen und die Modelle an bestimmte Plattformen, je nach deren Anforderungen, angepasst zu entwickeln.

Eine letzte, weiterführende Arbeit stammt von Rosenthal et al. [20]. Sie entwickelten einen erweiterten Datensatz nach dem Schema von Zampieri et al. [11]. Dabei wurde der Datensatz OLID als Trainingsdatensatz verwendet, um neue Tweets halbüberwacht zu labeln. Der neue SOLID-Datensatz (Semi-supervised Offensive Language Identification Dataset) umfasst nun ca. 9 Mio. Tweets. Zur Klassifikation wurden verschiedene Modelle wie PMI, FastText, LSTM und BERT in einer demokratischen Gemeinschaftsentscheidung

verwendet. Ebenso enthält der Datensatz der eine Einteilung in einfach und schwierig zu klassifizierende Tweets. Diese Zusatzinformation bekommt ein Tweet je nach der Wahrscheinlichkeit für eine Klasse c des jeweiligen Tasks. Anhand dieses Datensatzes ist es fortführend möglich neue Tweets dem OLID-Datensatz hinzufügen. Somit können Techniken, wie Klassenwichtung und Sampling umgangen werden.

2.2 Eingrenzung gerichtete offensive Sprache

Im folgenden Abschnitt wird der Task nach Zampieri et al. [11] detailliert dargestellt, da sich die vorliegende Arbeit an deren Ansatz orientiert. Dabei soll besonders auf das verwendete Annotationsschema und den Aufbau des Datensatzes eingegangen werden. Weiterhin erfolgt eine Definition für den Begriff der gerichteten offensiven Sprache.

Zampieri et al. orientierten sich bei der Entwicklung der Struktur an der Empfehlung von [10]. Somit gingen sie erstmals gleichzeitig auf die Analyse der Art und des Ziels eines Angriffes in Textnachrichten zusammen ein. Zur Umsetzung des Ziels, teilt sich das Problem, wie oben bereits dargelegt, in 3 Stufen. Dabei handelt es sich bei Task A und B um binäre Entscheidungen. Task C hingegen unterscheidet zwischen 3 Labels. Abbildung 1 stellt die Struktur des Tasks nach Zampieri et al. [11] schematisch dar.

Task A dient der Trennung von angreifender Sprache und nicht angreifender Sprache. Folgende Label werden durch die Autoren definiert:

- ➔ **OFF (offensiv):** jegliche Art von missbräuchlich verwendeter Sprache, sowie direkte und indirekte Angriffe.

- ➔ **NOT (nicht offensiv):** Posts, die keinen offensiven Inhalt haben

Task B grenzt ab, ob ein Angriff (OFF) gerichtet ist oder nicht. Die zugehörigen Label sind definiert als:

- ➔ **TIN (gerichteter Angriff):** Nachrichten, die angreifend gegenüber Personen, Gruppen oder anderen sind.
- ➔ **UNT (ungerichteter Angriff):** Nachrichten, die generelle offensive Ausdrücke beinhaltet. Dazu zählen Kraftausdrücke und Fluchen.

Der Mehrklassentask C unterscheidet drei Klassen, um das Ziel eines gerichteten Angriffes (TIN) zu identifizieren.

- ➔ **IND (Individuum)** Nachrichten, die sich gegen ein spezifisches Individuum richten. Dies können bekannte oder benannten Personen, sowie andere Chatteilnehmer sein.

- ➔ **GRP (Gruppen)** Nachrichten, die sich gegen Gruppen richten, welche gemeinsame Merkmale aufweisen. Dazu zählen unter anderen die sexuelle Orientierung, der religiöse Glauben und politische Einstellungen.

➔ **OTH (Andere)** Nachrichten, die keiner der beiden vorherigen Klassen zuzuordnen sind. Möglicherweise sind dadurch Situationen, Events oder Organisatorische Strukturen gemeint.

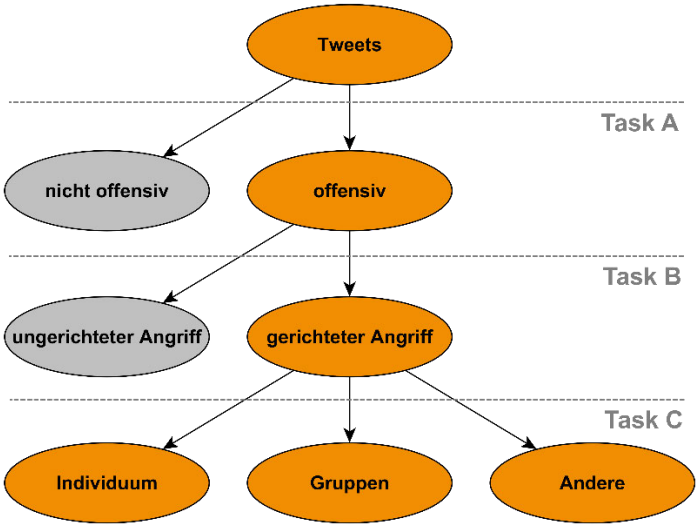

Abbildung 1: Struktur des Task der gerichteten offensiven Sprache. Schema nach Zampieri et al. als dreistufige Entscheidung. [11]

Um das neue Annotationsschema erfolgreich anwenden zu können, erstellten die Autoren den neuen Datensatz OLID mit insgesamte 14.100 englischen Tweets. Dieser wurde durch mehrere, englische Schlüsselwörter zum Zeitpunkt des Verfassens aktueller Themen erstellt. Zu dieser Sammlung gehören „MAGA“ („make America great again“), „BreitBartNews“, „medical marijuana“ und „gun control“. Diese Wörter konzentrieren sich besonders auf Tweets mit politischen Hintergründen.

Die entsprechenden Tweets wurden durch die Twitter API akquiriert und durch zwei Annotatoren gelabelt. Ca. 60% der Labels stimmten überein. Die restlichen wurden durch eine dritte Annotation mehrheitsentscheidend einem Label zugewiesen. Die Annotationsstruktur gibt vor, dass nur die Tweets eines Tasks gelabelt werden können, wenn sie im vorherigen Task als positive Klassen gekennzeichnet wurden (OFF und TIN). Nicht offensive Tweets können nicht gerichtet sein und ungerichtete Angriffe richten sich weder gegen Individuen, Gruppen oder Andere. Nachrichten der Klassen NOT und UNT werden keiner weiteren Klassifikation im folgenden Subtask unterzogen. Mit diesen Bedingungen ergeben sich gelabelte Tweets, wie in Tabelle 3 zu sehen ist.

Wie bereits erwähnt, kann durch die Identifikation des Ziels auf die Art des Angriffes geschlossen werden. Jedes der einzelnen Endblätter im Annotationsschema stellt eine spezifische Angriffsform dar, mit der Ausnahme des Labels NOT im Substask A. Bei ungerichteten Angriffen handelt es sich lediglich um eine Art der offensiven Sprache, die allgemein missbräuchlich verwendete Sprache enthält. Dazu zählen Schimpfen und Fluchen. Damit beschäftigten sich [5] bereits und bilden ein erstes Gebiet der offensiven Sprache. Nachrichten, die sich gegen ein Individuum richten, könnten als eine Art Cyber-

Tabelle 3: Beispiel Tweets aus OLID mit deren Labels je Subtask

Tweet	A	B	C
Always smack URL	NOT	---	---
.....bitch what URL	OFF	UNT	---
@USER ...with his stubby little arm.	OFF	TIN	IND
@USER #metoo are all racist!	OFF	TIN	GRP
@USER @USER @USER We call that game #Bullshit.	OFF	TIN	OTH

mobbing bezeichnet werden. Eindeutig ist das nicht in allen Tweets festzustellen, da Mobbing nach deren Definition einen repetitiven Prozess darstellt [6]. Da die Nachrichten im OLID-Datensatz ohne weitere Zusammenhänge ausgenommen werden, kann nicht immer von Cybermobbing ausgegangen werden. Tweets, die Gruppen betreffen, werden als Hassrede bezeichnet, wie sie [9] bereits definierten. Hier liegen Merkmale vor, die gruppenspezifisch sind und Ziel der Nachrichten sind. Die übrigen Tweets, die sich weder gegen Individuen noch Gruppen richten, beinhalten ebenfalls gerichtete Angriffe gegen andere Entitäten, wie Organisationen oder Events.

3 Feature-Selektionsalgorithmen

Im vorherigen Kapitel wurde gezeigt, dass die Detektion offensiver Sprache ein viel beforschtes Gebiet im Bereich des maschinellen Lernens ist. Ersichtlich wird, dass zur Erfüllung dieser Aufgabe viele Möglichkeiten bestehen, um Texteingaben zu verarbeiten. Die Verarbeitung natürlicher Sprache (engl. Natural Language Processing - NLP) bedient sich einem großen Repertoire von Methoden zur Umwandlung von Text in maschinenverständliche Form. [21] stellten bereits fest, dass Nachrichten auf sozialen Medien sehr unregelmäßig und grammatikalisch schwierig zu verstehen sind. Dazu zählen auch alternative Formen von Wörtern und Schreibfehler, wodurch lexikalische Ansätze teils schlecht performen. Es besteht die Notwendigkeit Modelle zu entwerfen, die in der Lage sind diese Unregelmäßigkeiten zu erkennen und möglichst robuste Feature zu selektieren.

Um Computern textuelle Zusammenhänge lernen zu können, muss eine Vorverarbeitung von Eingabetexten zu maschinenverständlichen Formaten erfolgen. Für diesen Vorgang wurden über die vergangenen Jahre hinweg eine Reihe an Methoden entwickelt, die es ermöglichen robuste Feature aus Eingabetexten zu extrahieren, um mit ihnen maschinelle Lernmodelle zu trainieren und für Vorhersagen zu nutzen.

In den folgenden Unterkapiteln sollen Ansätzen dargelegt und erläutert werden, die Feature aus Texteingaben extrahieren können und so kategorische Daten in numerische umwandeln. Es werden zuerst klassische Methoden betrachtet, die auch heute noch teils gute Ergebnisse liefern. Als weiteren Vertreter werden Schwarmintelligenzalgorithmen beschrieben. Diese können anhand von mehreren Sammlungen, das beste Featureset iterativ identifizieren und zur Klassifikation nutzen. Als dritte Kategorie werden neuronale Ansätze betrachtet. Sie prägen aktuelle State-of-the-Art Modelle und führen zu stetigen Verbesserungen im Bereich der Erkennung von offensiver Sprache.

3.1 Klassische Methoden

Klassische Methoden basieren auf Konzepten, die weit in das 20. Jahrhundert hineinreichen. Bereits 1954 erwähnte Harris [22] das Konzept des Bag-of-Words (BOW). Dieses wurde früh für Probleme des Text Retrievals adaptiert und bildet die Basis weiterer Ansätze zur Featureextraktion. Weiterhin wurde das BOW-Konzept für Bereiche wie Bildklassifizierung übernommen [23]. Aus diesem Ansatz entstanden weitere Entwicklungen, die mit verschiedenen Frequenzen der enthaltenen Wörter bessere Darstellungen bieten. Diese Repräsentationen können in einem Vektorraummodell dargestellt und verglichen werden.

Um die Konzepte genauer zu erläutern, ist eine Reihe von formalen Definitionen notwendig. Diese gelten für die in den folgenden Unterkapiteln beschriebenen Methoden.

Tabelle 4: Definitionen zur Erläuterung folgender Konzepte

Begriff	Formale Definition
Wörter	$w_i \mid i \in \mathbb{N}$
Vokabular	$V = \{w_1, w_2, \dots, w_N\}$
Dokument	$d_j = d_{i1}, d_{i2}, \dots, d_{im} \mid d_{ij} \in V$
Anfrage (Query)	$q = q_1, q_2, \dots, q_m \mid q_i \in V$
Sammlung (Collection)	$C = \{d_1, d_2, \dots, d_M\}$

3.1.1 Bag-of-Words

Das Konzept des Bag-of-Words (dt.: Tüte von Wörtern) ist eines der ältesten im Bereich des maschinellen Lernens und stammt ursprünglich aus der Linguistik [22]. Ein Problem, was Harris schon in seiner Publikation erwähnte, ist dass der Zusammenhang der Terme in dieser Repräsentation verloren geht.

Dokumente d_j bzw. Queries q stellen ein Textobjekt dar, welches aus einer geordneten oder ungeordneten Reihe an Wörtern w_i besteht. BOW zielt darauf ab, alle einzigartigen Wörter dieser Textobjekte darzustellen. Dabei sind BOW-Repräsentationen eine Menge von Wörtern, welche in den jeweiligen Textobjekten vorkommen. Die Textobjekte werden in Binärvektoren überführt, wobei nur die Elemente aktiviert sind, deren Wörter auch im Text vorkommen. Eine Variante der Implementation ist das Nutzen der Binärvektoren als Dokumentvektoren. Damit sind alle Wörter unabhängig von deren Vorkommen gleich gewichtig. Eine andere Art ist das Hinzuziehen der Termfrequenz der einzelnen Wörter. Durch das Zählen des jeweiligen Wortes w_i, werden die Wörter wichtiger, je häufiger sie vorkommen. Tabelle 5 stellt das BOW-Konzept dar an einem einfachen Beispiel dar.

Tabelle 5: Beispiel für Bag-of-Words Konzept.

Text 1:	Frank informiert sich über neue Bioprodukte.
Text 2:	Neue Bioprodukte: Das Gesundheitsamt informiert.

Vokabular:		frank	nformiert	sich	über	neue	bioprodukte	das	gesundheitsamt	
BOW 1 =	(	1	1	1	1	1	1	0	0	)
BOW 2 =	(	0	1	0	0	1	1	1	1	)

3.1.2 One Hot Encoding

Eine weitere einfache Möglichkeit zur Featureextraktion aus kategorischen Variablen ist das One Hot Encoding. Dieses ähnelt dem Prinzip des BOW, wobei die Wörter aber als einzelne Binärvektoren dargestellt werden. Es liegt stets ein indexiertes Wörterbuch zugrunde, welches als Vokabular V angenommen werden kann. Die Binärvektoren bekommen an der Stelle i eine 1 zugewiesen, welche dem Index des Wortes im Wörterbuch entspricht. Somit besitzt jeder Vektor exakt eine Stelle, in der eine 1 vorhanden ist [24]. Der Rest besteht aus Nullwerten. Aufgrund vieler Nullwerte eignet sich diese Methode nicht zum Kodieren von Feature aus großen Datenmengen. Auch hierzu folgt ein kurzes Beispiel.

Tabelle 6: Beispiel für One Hot Encoding Konzept

Text:	Frank informiert sich über neue Bioprodukte.					
	Frank	nformiert	sich	über	neue	bioprodukte
Vokabular						
bioprodukte	0	0	0	0	0	**1**
frank	**1**	0	0	0	0	0
informiert	0	**1**	0	0	0	0
neue	0	0	0	0	**1**	0
sich	0	0	**1**	0	0	0
über	0	0	0	**1**	0	0

3.1.3 TFIDF

TFIDF (Term Frequenz / Inverse Dokumentenfrequenz) ist eine Erweiterung der einfachen BOW-Repräsentation. BOW hat einen entscheidenden Nachteil in Bezug auf den Einfluss von Wörtern auf das Ranking des Dokuments. Alle Wörter erhalten die gleiche Gewichtung. Es ist wahrscheinlich, dass mehrere Dokumente das gleiche Ranking im Text Retrieval erhalten. Um diesem Problem entgegenzuwirken, besteht das TFIFDF-Konzept aus zwei Bestandteilen. Es werden die Termfrequenz TF und die inverse Dokumentenfrequenz eines Wortes w_i betrachtet. Dieser Ansatz ermöglicht eine feingranulare Darstellung von Dokumentenvektoren [25]. Der Methode liegt ein wohldefiniertes Vokabular V zugrunde. Jedes Wort $w_i$besitzt einen eindeutigen Index in V [26]. Somit kann einem Wort w_i eindeutig das Gewicht im Dokumentenvektor zugeordnet werden.

Die Termfrequenz beschreibt die Häufigkeit eines Wortes w_i in einem Dokument d. Formal ist diese definiert als die Frequenz f von w in d, wie in Formel 1 dargestellt.

$$TF(w,d) = f_{w,d} \tag{1}$$

Mit Verwendung der Termfrequenz ist bereits eine Gewichtung der einzelnen Wörter möglich. Dadurch werden alle w_i nur durch deren Häufigkeit gewichtet. Tritt ein Wort häufig in einem Dokument auf, wird dessen Wichtigkeit immer höher. So werden Stoppwörter, wie „auch“ oder „ist“ besonders hoch gewichtet. Stoppwörter sind Terme, die im Sprachgebrauch sehr oft vorkommen und themenunspezifisch sind. Durch deren häufiges Vorkommen, werden sie durch die Termfrequenz sehr hoch gewichtet, obwohl sie für ein Thema oder eine Klasse eines Dokumentes keine Aussagekraft haben. [27]

Um diesem Phänomen entgegenzuwirken, wird zusätzlich die inverse Dokumentenfrequenz berechnet. Diese richtet sich nach der Anzahl der Dokumente d in denen das Wort w_i in der Sammlung C vorkommt. Durch einen Logarithmus wird eine Normalisierung vorgenommen. Formal wird die inverse Dokumentenfrequenz nach Formel 2 definiert.

$$IDF(w) = log\left(\frac{|C| + 1}{k}\right) \tag{2}$$

$|C|$ ist die Anzahl aller Dokumente in der Sammlung C. k stellt die Anzahl der Dokumente dar, in denen das Wort w vorkommt [28]. Die beiden Teile ergeben zusammen die Formel 3 und somit das Produkt beider Frequenzen.

$$TFIDF(w) = f_{w,d} \cdot log\left(\frac{|C| + 1}{k}\right) \tag{3}$$

Es lässt sich beobachten, dass je häufiger ein Wort in einem Dokument und weniger in der gesamten Sammlung vorkommt, desto höher dessen TFIDF-Gewicht ist. Somit wird4 häufig auftretenden Wörtern (Stoppwörter) ein geringes Gewicht zugewiesen. Terme, die besonders selten und aussagekräftig sind, bekommen dagegen ein deutlich höheres Gewicht. Aufgrund dieser Eigenschaften ist es möglich themenspezifische Begriffe bedeutender für die Klassifikation zu machen und bessere Ergebnisse zu erzielen.

3.1.4 Vektorraummodell

Die bisher beschriebenen Methoden zeigen, wie kategorische Daten in numerische überführt werden können. Das gilt für die Dokumente d_j einer Sammlung, sowie für die gestellten Queries bzw. die ungesehenen Dokumente. Basis des Modells ist die Instanziierung der Dimensionen, welche durch die Wörter w_i des Vokabulars V beschrieben werden. Die Anzahl der Dimensionen des Vektorraummodells entspricht der Länge von V. Durch eine TFIDF-Gewichtung der Dokumenten- und Query-Vektoren können diese im Vektorraum platziert werden.

Ziel ist es unabhängig vom Einsatzgebiet, dass zu einer Query q ein möglichst ähnliches d in C gefunden werden soll. Abbildung 2 veranschaulicht ein dreidimensionales Vektorraummodell. Beispielhaft wird hier $\vec{q}$ als Query-Vektor angenommen. Bestimmt wird, welcher der Vektoren $\overrightarrow{d_1}$ oder $\overrightarrow{d_2}$ diesem am ähnlichsten ist. Es muss eine Möglichkeit geschaffen werden, bestehende Dokumente mit neuen zu vergleichen und deren Ähnlichkeit

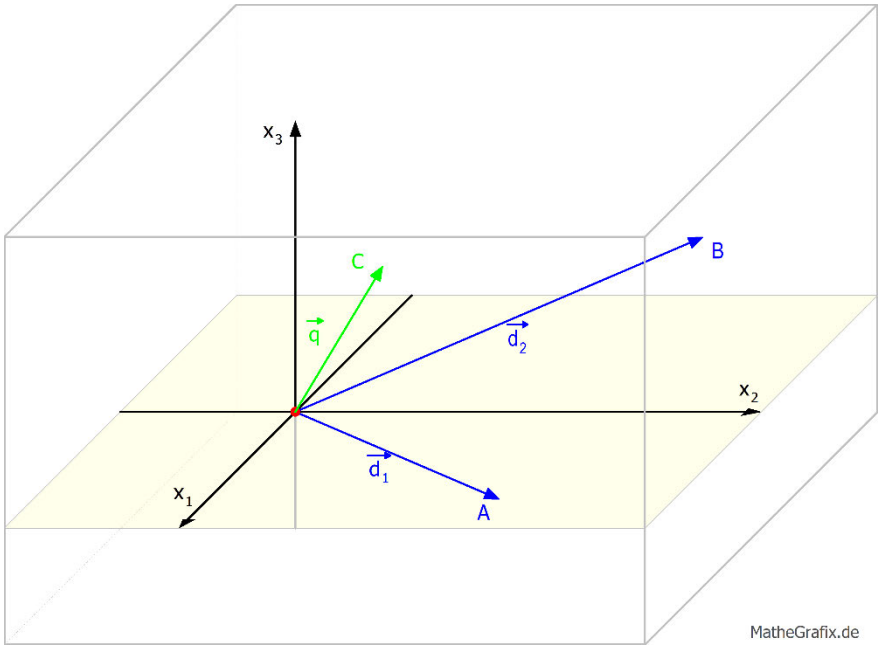

Abbildung 2: Dreidimensionales Vektorraummodell

zueinander zu bestimmen. Dafür wird ein Ähnlichkeitsmaß $sim(q, d)$ definiert. Die Eingabewerte

$$q = (x_1, \dots, x_N) \tag{4}$$

$$d = (y_1, \dots, y_N) \tag{5}$$

werden als gewichtete Dokumenten- und Query-Vektoren betrachtet [29]. x_i und y_i beschreiben die Feature (Gewichte) der Wörter des jeweiligen Textobjektes. Als einfachstes Ähnlichkeitsmaß ist das Skalarprodukt heranzuziehen. Die Produkte der Feature werden über den Vektor aufsummiert mit der Formel:

$$sim(q, d) = q \cdot d = \sum_{i=1}^{N} x_i \cdot y_i \tag{6}$$

beschrieben. Eines der am häufigsten verwendeten Maße ist die Kosinus-Ähnlichkeit. Diese basiert auf dem Winkel zwischen zwei Vektoren in einem n-dimensionalen Raum. Auch hier wird das Skalarprodukt verwendet mit der Normalisierung durch das Produkt der Normen der Vektoren. Gleichung 7 zeigt die Formel zur Kosinus-Ähnlichkeit. [29]

$$Cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\|\|\vec{b}\|} = \frac{\sum_1^n a_i \cdot b_i}{\sqrt{\sum_1^n a_i^2} \cdot \sqrt{\sum_1^n b_i^2}} \tag{7}$$

Mit den in diesem Kapitel aufgeführten Methoden können Queries den bestehenden Dokumenten zugeordnet werden. Um festzustellen, welchem Dokument sie am ähnlichsten sind ist es nötig die Dimensionen zu instanziieren, Gewichtung zu berechnen und eine

Ähnlichkeit zwischen den einzelnen Textobjekten zu definieren. Somit sind Textklassifikationsprobleme grundlegend zu lösen. Auch in aktuellen Arbeiten wird weiterhin mit diesen gearbeitet. Ein Beispiel dafür ist eine Arbeit von Mohammed und Omar zur Klassifikation von Fragen mit einer modifizierten Funktionsweise von TF-IDF [30]. Oftmals werden sie auch als Baseline in Verbindung mit Modellen wie SVM oder logistischer Regression verwendet.

3.2 Schwarmintelligenzalgorithmen

Ansätze, die in den bisherigen Beschreibungen dieser Arbeit und in anderen verwandten Arbeiten nicht zu finden sind, sind Schwarmintelligenzalgorithmen. Diese Algorithmen finden auf vielerlei Gebieten Anwendungen. Das US-Militär verwendet ihre Drohnenschwärme mit Techniken, die sich an natürlichem Schwarmverhalten orientieren. Bereits 1992 simulierten [31] einen Versuch, um mittels Schwarmverhalten Nanoroboter zu steuern, die im Körper Krebszellen bekämpfen sollen.

Schwarmintelligenzalgorithmen finden Anwendung bei Problemen, welche mit einer hohen Komplexität verbunden sind. Sie konzentrieren sich auf mathematische Funktionen, deren Berechnung unverhältnismäßig lang dauern würde. Durch einfaches, aber zielorientiertes Testen und Verbessern von Lösungen für diese Funktionen, finden Schwarmintelligenzalgorithmen iterativ nahezu optimale Lösungen. Marco Dorigo gilt für schwarmorientierte Algorithmen als Vorreiter. Ebenfalls 1992 verfasste er seine Dissertation über Optimierung und Lernen von naturorientierten Algorithmen. In den folgenden Jahren entwickelte er einen Ansatz namens Ant System [32]. 2006 veröffentlicht er eine Publikation über den Ant Colony Algorithmus, welcher heute noch einer der bekanntesten ist [33]. Einer der neusten Vertreter ist HBBO. Ahmadi [34] führte diesen 2017 ein und verwendete damit erstmals menschliches Verhalten als Vorlage für Optimierungsalgorithmen. Dieser Ansatz ist für die vorliegende Arbeit der wichtigste und wird in Abschnitt 3.2.2 tiefgehender betrachtet.

3.2.1 Ant Colony Optimization

Die Grundlage für den Ant Colony Optimization Algorithmus legte Marco Dorigo et al. 1996 mit seiner Publikation über das Ant System [32]. Dieses System ist auf kombinatorische Optimierungsprobleme ausgelegt und basiert auf dem Kommunikationsverhalten Ameisen. Diese verständigen sich blind über neue kurze Wege zu Nahrungsfundorten. Das bekannteste Problem, welches mit dem Algorithmus behandelt wird, ist das Traveler Salesman Problem. Aufbauend auf den bisherigen Forschungen, veröffentliche Dorigo et al. 2006 [33] eine Publikation über die Metaheuristik des Ant Colony Optimierungsalgorithmus. Er beschreibt besonders den biologischen Hintergrund, welcher im folgenden Kapitel kurz erläutert wird.

3.2.1.1 Biologischer Hintergrund

Bereits 1959 stellte der französische Entomologe Grassé [35] fest, dass Ameisen blind kommunizieren über einen Stoff, den er „*Stigmergy*“ nennt. Sie stellen eine indirekte Kommunikation dar und sind lokal gebunden. Diese Stoffe heißen heute Pheromone und sind Grundlage für die Kommunikation der Ameisen. Beim Kriechen werden sie abgelegt und durch andere Mitglieder einer Kolonie wahrgenommen. Ameisen tendieren eher dazu einen Weg zu gehen, der mit Pheromonen gekennzeichnet ist. Je mehr Pheromone auf einem Weg vorhanden sind, desto wahrscheinlicher ist die Nutzung durch eine Ameise. Während der Nutzung des Weges werden wieder Pheromone abgelegt, was zu einer immer höheren Konzentration führt. Kurze Wege führen dazu, dass Ameisen schneller am gleichen Punkt wieder sind und erneut Pheromone platzieren. Je kürzer der Weg, desto schneller werden die Stoffe inkrementiert. Somit haben kurze Wege eine höhere Pheromonkonzentration als lange und werden mehr genutzt. Diese Gegebenheiten wurden adaptiert, um für komplexe Probleme, möglichst kostengünstige Lösungen (kurze Wege) zu finden.

3.2.1.2 Technische Umsetzung

Im Ant Colony Optimization-Algorithmus bilden eine Vielzahl an Ameisen eine Lösung für Optimierungsprobleme und tauschen Informationen über deren Qualität aus. Das geschieht mittels eines Kommunikationssystems, welches an das Verhalten von echten Ameisen angelehnt ist [33]. Um den Algorithmus möglichst genau darzulegen, ist eine Reihe an Definitionen notwendig.

Die Grundlage bildet ein Modell $P = (S, \Omega, f)$. S ist der Suchraum über ein finites Set $X_i, i \in \mathbb{N}$ von Entscheidungsvariablen, wobei $X_i \in D_i = \{v_i^1, \dots, v_i^{|D_i|}\}$. Eine mögliche Lösung $s \in S$ ist ein vollständiges Set von Variablen, diese sind mit Ω vereinbar sind. Ω ist ein Set von Regeln für das Modell. f ist die Zielfunktion, welche den Suchraum auf positive reelle Zahlen abbildet: $f: S \to \mathbb{R}_0^+$. Lösungskomponenten werden als $c_{ij} \in C, i = 1, \dots, n,\ j = 1, \dots, |D_j|$ bezeichnet und stellen Zuordnungen zwischen den $X_i = v_i^j$ dar. Das Modell wird auf einen vollverknüpften Konstruktionsgraphen $G_C(V, E)$ angewandt, dabei stellen V die Knoten (Vertices) und E die Kanten (Edges) dar.

Ant Colony Optimization Algorithmus

```
Parameter setzen
Pheromone auf Pfaden initialisieren
solange Abbruchbedingung nicht erfüllt
    Konstruiere Ameisen Lösung
    Lokale Suche
    Pheromon Update
ende solange
```

Abbildung 3: Pseudocode zum Ant Colony Algorithmus nach [36]

Mit diesen Formalisierungen ist der Ant Colony Optimierungsalgorithmus durchführbar. Wesentlich besteht dieser aus 3 Schritten, wie in Abbildung 3 als Pseudocode dargestellt.

Je nach Implementierung werden die Pheromone auf den Pfaden unterschiedlich initialisiert. Möglich ist das Setzen der Pheromonstärke $\tau_{ij} = 0$ oder das setzen auf einen initialen Wert. Zweiteres ist die häufiger vorzufindende Variante. Nach der Initialisierung iteriert eine Schleife über die drei Schritte solange bis ein Abbruchkriterium erfüllt ist.

Zu Beginn erstellen die m Ameisen eine Lösung aus den Lösungskomponenten c_{ij}, welche einem Set von partiellen Lösungen s^p hinzugefügt wird. Dieses ist anfänglich leer. In jedem Schritt des Algorithmus wird diesem Set eine Lösung hinzugefügt, die mit den Definitionen in Ω konform ist. Die Auswahl der partiellen Lösung wird durch die Ameisen bedingt, die je nach Pheromonstärke τ_{ij} über den Graphen traversieren.

Optional kann nach diesem Schritt eine lokale Suche erfolgen. Diese dient dazu gefunden Lösungen möglicherweise zu verbessern. Dazu wird die partielle Lösung s^p minimal modifiziert und wiederrum mit der Zielfunktion f überprüft. Findet eine Verbesserung des Funktionswertes statt, wird die vorherige Lösung durch die neue ersetzt, sonst verworfen.

Letztliche werden die Pheromonkonzentrationen τ_{ij} auf den c_{ij} verbessert, welche Teil der partiellen Lösung s^p sind. Im ursprünglichen Ant System [32] wurde das durch eine Verdampfung der aktuellen Pheromone realisiert, welche durch Formel 8 gezeigt wird.

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{8}$$

τ_{ij} stellt die Pheromonstärke auf c_{ij} dar. ρ ist der Verdampfungsfaktor, welcher bewirkt, dass abgelegtes Pheromon nicht konstant ist, sondern über mehrere Iterationen hinweg abgebaut wird. Schließlich wird die Gesamtmenge Pheromon $\Delta\tau_{ij}^{k}$, welche durch alle Ameisen auf dem Pfad c_{ij} abgelegt wurde aufsummiert.

Ein wenig anders wird das Problem vom Ant Colony System (ACS) [36] gelöst. Hier wird die Aktualisierung online (nach jedem Konstruktionsschritt) und offline (einmal pro Iteration) vorgenommen. Formel 9 zeigt die Online Aktualisierung, wobei τ_0 der Initialwert ist.

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \rho \cdot \tau_0 \tag{9}$$

Jede Ameise wendet diese Formel auf der letzten Kante an, auf der sie gegangen ist. Die Offlineaktualisierung erfolgt mit einer ähnlichen Gleichung, wie bei AS. Das Update wird aber nur von der jeweils besten Ameise einer Iteration durchgeführt. Dadurch wird sichergestellt, dass ausschließlich möglichst optimale Lösungen auf das Gesamtergebnis einfließen. Formel 10 zeigt, dass die aktualisierte Pheromonstärke nur übernommen wird, wenn diese zur besseren Lösung beiträgt.

$$\tau_{ij} \leftarrow \begin{cases} (1-\rho) \cdot \tau_{ij} + \rho \cdot \Delta\tau_{ij}, & wenn\ (i,j)\ zur\ besten\ Lösung\ gehört \\ \tau_{ij}, & sonst \end{cases} \tag{10}$$

Mit diesem Algorithmus ist es möglich den optimalen Pfad durch einen Graphen zu finden, um die Kosten möglichst gering zu halten. Die besten Pfade werden schließlich durch die stärksten Pheromonkonzentrationen gekennzeichnet. Probleme, wie das Traveler Salesmen Problem lassen sich so iterativ lösen. Weitere denkbare Anwendungen können die Navigation in Straßennetzen [37], verbindungsorientiertes Routing [38] oder Erkennung von Kanten in Bildern sein [39]. Anwendungen im Bereich der Feature-Selektion gibt es nur wenige. Eine davon stammt von der Hochschule Mittweida in einem Ansatz zur Erkennung von Texten aus dem Internet, die sich auf Musikevents beziehen [40]. Hierbei wurde die Featureselektion mit dem ACO umgesetzt. Der Ansatz erreichte F1-Maße von ≥ 0.95 und zeigt somit eine herausragende Performanz.

3.2.2 Human-behavior based Optimization

Human behavior-based Optimization (HBBO) ist einer der neusten Ansätze im Bereich der Schwarmintelligenz. Ahmadi veröffentlicht diesen 2017 und betrachtet erstmals den Menschen als einzelnes Individuum im Fokus von Schwarmintelligenz [34]. HBBO ist im Gegensatz zu ACO nicht graphen-, sondern gruppenorientiert. Jedes Individuum besitzt Werte, welches als seine Merkmale angesehen werden können. Die Zielfunktion gilt es für die Individuen zu verbessern, indem sie ihre Werte iterativ anpassen und verbessern. Die folgenden zwei Abschnitte betrachten die biologische Grundlage, die technische Umsetzung und eine Adaption zur Selektion effektiver Feature für den Bereich der Textklassifikation.

3.2.2.1 Biologischer Hintergrund

Die Grundlage für dieses Verfahren bildet der Mensch. Menschen kommunizieren. Nach Paul Watzlawick können Menschen nicht nicht kommunizieren. Jeder Mensch versucht nach seinen besten Möglichkeiten einen Zweck zu erfüllen und Ziele zu erreichen. Um diesem Anspruch gerecht zu werden, sollte er die beste Version seiner selbst sein. Allerdings hat jeder Mensch andere Ziele, Interessengebiete und Fähigkeiten. Je nachdem, wie Menschen lernen, sind ihre Fähigkeiten besser oder schlechter als die, anderer Menschen. Tauschen sich Menschen in ihrer Kommunikation darüber aus, können sie von anderen lernen und ihre eigenen Fähigkeiten verbessern. Weiterhin lässt sich beobachten, dass manche Menschen ihre Interessengebiete im Laufe des Lebens wechseln. Die eignen sich Fähigkeiten auf anderen Gebieten an, tauschen sich aus und lernen dabei. Weiterhin gibt es Menschen, die ein Interessengebiet besonders gut beherrschen. Aufgrund deren herausragender Qualität, können alle anderen Menschen in diesem Gebiet von ihm Lernen.

Anhand dieser Beobachtungen in der realen Welt und Annahmen über das menschliche Verhalten, hat Ahmadi [34] einen mächtigen Optimierungsalgorithmus entworfen, welcher für eine Vielzahl von Problemen anwendbar ist. Im folgenden Kapitel soll dieses Verfahren näher beschrieben werden.

3.2.2.2 Technische Umsetzung

Wie bereits beschrieben besteht das Lernen des Menschen aus verschiedenen Schritten. So ist auch der Algorithmus in 5 Schritte eingeteilt, um das Leben bzw. den Fähigkeitserwerb des Menschen zu modellieren. Dabei dient Schritt 1 der Initialisierung und Schritt 5 dem Abschluss der Prozedur. Schritt 2 bis 4 werden iterativ wiederholt, bis eins der definierten Abbruchkriterien erfüllt ist.

Human behavior-based Optimization

Parameter setzen
Initialisierung
solange Abbruchbedingung nicht erfüllt
    Lernen
    Konsultation
    Feldwechsel
ende solange
Abschluss

Abbildung 4: Pseudocode zum HBBO-Algorithmus nach [34]

Die Initialisierung ist der erste Schritt, nachdem die Parameter des Algorithmus gesetzt sind. Hier werden den Individuen je nach Optimierungsproblem deren initialen Feature (Merkmale) bzw. Variablen zugewiesen. Für ein Problem mit N_{var} Variablen werden Individuen mit N_{var} Merkmale definiert. Es gilt: $Ind = [x_1, x_2, \dots, x_{Nvar}]$. Die Anzahl der Individuen N_{pop} ist frei zu wählen und sollte auf das jeweilige Problem individuell angepasst werden. Jedes Individuum gehört einem Feld F an. Die initialisierten Individuen werden gleichmäßig auf die N_{field} Felder verteilt. Zum Schluss der Initialisierung wird für jedes Individuum dessen Funktionswert der Zielfunktion f nach Formel 11 berechnet. Dabei sind x_i die Features eines Individuums. Individuen mit dem besten Funktionswert eines Feldes F werden als dessen Experte bezeichnet.

$$function\ value = f(x_1, x_2, \dots, x_{Nvar}) \tag{11}$$

Der Schritt des Lernens ist der erste im Optimierungsprozess. Dieser Schritt wird anhand eines N_{var}-dimensionalen Koordinatensystems durchgeführt. Der Experte eines Feldes ist der Ursprung dessen und die restlichen Individuen bewegen sich in einem sphärischen Raum um diesen herum. Dieser Raum um den Experten wird eingegrenzt durch eine Zufallsvariable r, für die gilt:

$$r_{min} = k_1 d < r < r_{max} = k_2 d \tag{12}$$

k ist der Gewichtsfaktor zur Steuerung des Abstandes zum Experten und d stellt die aktuelle euklidische Distanz des Individuums zu diesem dar. Es werden außerdem $N_{var} - 1$ Winkel θ_i zufällig gewählt in deren Richtung das Individuum bewegt wird. Dabei gilt, dass θ_{N-1} zwischen 0 und 2π liegt und alle anderen Winkel zwischen 0 und π. Anschließend wird das Individuum mit dem gewählten r und den Winkeln θ_i um den Experten herumbe-

wegt. Ziel dieses Schrittes ist es, jedes Individuum näher an den Experten heranzubewegen. Dadurch wird bewirkt, dass es teils die Eigenschaften des Experten annimmt und so seinen eigenen Funktionswert verbessert.

Im nächsten Schritt gehen die Individuen in die Konsultation. Dieser Schritt basiert auf dem Austausch von Fähigkeiten zwischen Menschen. Jedes Individuum findet ein zufälliges anderes Individuum der gesamten Population und tauscht mit ihm N_c Variablen aus. Für das nun entstandene Feature der Individuen wird der Funktionswert berechnet. Wird dieser durch den Austausch verbessert, wird das Set beibehalten, andernfalls verworfen. Die Anzahl der ausgetauschten Variablen berechnet sich aus dem Konsultationsfaktor $\sigma \in [0,1]$ anhand der Formel 13.

$$N_c = round(\sigma \cdot N_{var}) \tag{13}$$

Je höher σ, desto mehr Feature werden ausgetauscht. Das kann einerseits eher zu zufälligen besseren Werten führen. Andererseits beeinträchtigt es Instabilität der Werte einzelner Individuen, womit der Variablenaustausch an Effektivität verlieren kann.

Den letzten Schritt der Schleife stellt der Feldwechsel dar. Hier gibt es für Individuen die Möglichkeit ihr Feld F zu wechseln. Dazu werden alle Felder, wie in Formel 14 gezeigt, je nach deren Expertenqualität bezogen auf die Zielfunktion geordnet.

$$sortierte\ Felder = [field_1, field_2, \dots, field_{N_{field}}] \tag{14}$$

$field_1$ hat den schlechtesten und $field_{N_{field}}$ den besten Experten. Anschließend berechnet sich die Wahrscheinlichkeit, dass ein Individuum sein Feld wechselt anhand der Position in der Liste der sortierten Felder (O_i) und der Gesamtanzahl aller Felder. Formal wird diese Wahrscheinlichkeit nach Formel 15 berechnet.

$$P_i = \frac{O_i}{N_{field} + 1} \tag{15}$$

Zur finalen Entscheidung zum Feldwechsel eines zufällig ausgewählten Individuums jeden Feldes wird zusätzlich eine Zufallsvariable $rand$ bestimmt. Ist die berechnete Wahrscheinlichkeit $P_i \geq rand$, wird der Feldwechsel durchgeführt.

Der Abschluss wird nur dann erreicht, wenn eines der vorher definierten Abbruchkriterien erfüllt ist. Dazu können das Erreichen der maximalen Anzahl an Iterationen zählen als auch die Möglichkeit, dass sich der Funktionswert nicht mehr verbessert. Mit dem nun entstandenen möglichst optimalen Set von Variablen, kann das Optimierungsproblem gelöst werden.

HBBO lässt sich auf eine Vielzahl von Funktionen anwenden. Ahmadi zeigt dies an 14 Benchmark-Funktionen in der Publikation von 2017 [34]. HBBO erreichte neben Algorithmen wie genetischer Programmierung und Partikelschwarmoptimierung stets den Bestwert. Soto et al. [41] wendeten HBBO auf das Manufacturing Cell Design Problem an. Ihr Ansatz erreicht gute Werte, bedarf allerdings noch Verbesserungen laut der Autoren.

3.2.2.3 HBBO zur Feature-Selektion in Textklassfikationsproblemen

Wie bereits ACO ist auch HBBO für die Feature-Selektion in Textklassifikationsproblemen anwendbar. 2022 entwarf das Forscherteam FoSIL der Hochschule Mittweida eine Adaption des HBBO-Algorithmus für ein solches Problem [42]. Dabei testeten sie den Ansatz bei der Klassifikation von Fake-News, welche insbesondere im Zuge der Corona-Pandemie aufkamen. In diesem Abschnitt wird der Ansatz nach [42] als Abwandlung von HBBO dargelegt.

Für diese Aufgabe sind einige Modifikationen nötig. Der algorithmische Ablauf wird beibehalten und durch Abbildung 4 zu beschrieben. Als Individuen I werden die Dokumente des Datensatzes angenommen. Sie repräsentieren die Wörter des Dokumentes als Bitvektor, wie in Formel 16 gezeigt.

$$I = (x_1, x_2, \dots, x_N)^T \tag{16}$$

Die verwendete Zielfunktion ist das F_1-Maß, welches es zu maximieren gilt. Ein Individuum $I^{(F)}$ des Feldes F welches die geringste Fehlerfunktion aufweist wird als Experte $E^{(F)}$ des Feldes angenommen. Es gilt Formel 17.

$$I^{(F)} \rightarrow E^{(F)} = argmin\, f^{error}(F) \tag{17}$$

Wie beschrieben, werden die Individuen mittels eines Bitvektors aus den Eingabedokumenten initialisiert. Die Felder des Optimierungsproblems richten sich nach den Klassen der Textklassifizierung. Jedes Individuum bekommt das entsprechende Label des Dokumentes zugewiesen. Ein weiterer Unterschied besteht darin, dass die Individuen nicht mehr einzeln, sondern in Gruppen optimiert werden. Die Einteilung der Gruppen wird mittels einer Gleichverteilung der Individuen nach deren Feld realisiert.

Der Schritt des Lernens unterscheidet sich wesentlich von dem des ursprünglichen Verfahrens. Es wird eine Gruppe von Individuen pro Feld bestimmt, welche den höchsten Funktionswert erzeugen. Diese Gruppe heißt Expertengruppe nach Formel 18.

$$I^{(F)} \rightarrow E^{(F)} = argmax\, F_1 \tag{18}$$

Von dieser Gruppe wird eine Untermenge von Feature S_E genutzt, um die übrigen Individuen der Gruppe zu verbessern. Die Individuen nehmen diese Untermenge an Feature in deren eigene Vektoren temporär auf. Daraufhin wird der Fehlerwert der Individuen bestimmt. Verbessert sich dieser, wird das Featureset übernommen, andernfalls verworfen. Formel wird die Bedingung durch Gleichung 19 beschrieben.

$$I_i^{(F)} = \begin{cases} I_i^{(F)} \cup S_E, & if\ f^{error}\left(I_i^{(F)} \cup S_E\right) < f^{error}\left(I_i^{(F)}\right) \\ I_i^{(F)}, & sonst \end{cases} \tag{19}$$

Die Konsultation ähnelt dem Verfahren von [34]. Es werden allerdings unabhängig vom Expertenstatus Untermengen von Feature zwischen je zwei zufälligen Gruppen ausgetauscht. Es gilt eine ähnliche Formel, wie im Lernprozess:

$$I_i^{(F)} = \begin{cases} I_i^{(F)} \cup S_j, & if\ f^{error}\left(I_i^{(F)} \cup S_j\right) < f^{error}\left(I_i^{(F)}\right) \\ I_i^{(F)}, & sonst \end{cases} \quad (20)$$

Für den Feldwechsel werden alle Felder nach dem Schema von [34] in geordnete Reihenfolge gebracht, sodass das Feld mit dem schlechtesten Experten zuerst und das mit dem besten zuletzt aufgenommen wird. Die Feature bleiben beim Feldwechsel unberührt. Nur das Feld des Individuums ändert sich. Ein Individuum kann nur zu dem Feld wechseln, welchem die beste Expertengruppe angehört. Somit bestimmt die Expertengruppe mit dem geringsten Fehlerwert, das Feld des wechselnden Individuums. Bei binären Klassifikationsproblemen findet nur eine Inversion des aktuellen Feldes statt. Auch hier gilt, dass der Wechsel nur stattfindet, wenn er eine Verbesserung des Zielfunktionswertes bewirkt.

Ludwig et al. [42] testen diesen Ansatz anhand der Detektion von Fake-News. Sie unterscheiden dabei 4 Klassen: wahr, falsch, teilweise falsch und andere mit Verweis auf möglichen Inhalt von falschen Aussagen. Der Trainingskorpus umfasst 1264 Dokumente und ist damit moderat groß. Die Vorverarbeitung der Daten ähnelt der, die auch in der vorliegenden Arbeit praktiziert wird. Außerdem wird eine 5-Fold-Kreuzvalidierung angewandt. Die Mehrklassenentscheidungen wurden auf jeweils 4 Binärentscheidungen reduziert. Überraschender Weise fallen die F1-Maße von den Trainingsdaten auf die Testdaten massiv ab. Wo im Training ein F1-Maß von 0.602 erreicht wird, kann auf den Testdaten nur ein Wert von 0.251 beobachtet werden. Möglicherweise neigt das Modell zu einer schnellen Überanpassung an die Trainingsdaten über eine Dauer von 125 Iterationen pro Fold. Des Weiteren ist den Autoren nach notwendig die Dokumente einem Faktencheck zu unterziehen, da die Erkennung von Falschaussagen rein linguistisch nicht einwandfrei realisierbar sei [42].

3.3 Neuronale Ansätze

Neuronale Ansätze sind der letzte Vertreter zu Feature-Selektionsalgorithmen, der in dieser Arbeit Erwähnung finden soll. Wie auch in den klassischen Methoden geht deren Ursprung weit zurück. Schon 1958 beschrieb Frank Rosenblatt das Perzeptron als Vorreiter für künstliche neuronale Netze (kurz: neuronale Netze) [43]. Es ist ein probabilistisches Modell zur Speicherung und Verarbeitung von Informationen im Gehirn. Dieses war das Vorbild für die Entwicklung solcher Netze, da deren Struktur ebenso aus Neuronen und Synapsen (Verbindungen) besteht.

Viele Probleme können durch maschinelles Lernen mit den Algorithmen gelöst werden, die bereits gezeigt wurden. Die Verarbeitung durch maschinelles Lernen bedingt, dass eine Vorverarbeitung stattfinden muss. Sie kann bspw. durch Tokenisierung und TFIDF-Gewichtung stattfinden. Erfolg versprechen diese Methoden, wenn aussagekräftige Feature für das jeweilige Problem ausgewählt und vorverarbeitet wurden. Dieser Schritt kann bei Ansätzen basierend auf neuronalen Netzen teilweise eingespart werden. Besonders tiefe Netze (Deep Learning Networks) sind in der Lage, anhand der Erfahrung durch den Lernprozess zu entscheiden, welche Feature besonders sinnvoll für die Klassifikation sind. Eine vorteilhafte Eigenschaft für Probleme, bei denen nicht klar abzutrennen ist, welche Merkmale aussagekräftig sind und welche nicht.

Den Vorteil machen sich heute Anwendungen aus verschiedensten Bereichen zu nutze. Typische Anwendungen für neuronale Netze sind neben Bild-, Sprach- und Videoverarbeitung auch Robotik, Bioinformatik und Chemie. Außerdem erfüllen sie im Bereich des Internets ebenso viele Aufgaben, wie in Videospielen, Suchmaschinen und Onlinewerbung. Auch das Problem der gerichteten offensiven Sprache kann durch neuronale Netze gelöst werden.

Das folgende Kapitel beschreibt den grundlegenden Aufbau dieser Netze. Es wird auf deren kleinste Einheit eingegangen und der Lernprozess beschrieben. In Bezug auf die Feature-Selektion im Bereich des NLPs werden bekannte Vertreter genannt und deren Eigenschaften erläutert. Der Fokus liegt auf dem State-of-the-Art-Modell BERT, welches detailliert beschrieben wird.

3.3.1 Neuronale Netze

Neuronale Netze sind Bestandteile des maschinellen Lernens und werden als Deep Learning bezeichnet. Sie werden allgemein als überwachte Lernverfahren bezeichnet, da sie eine Zielvariable benötigen, welche sie erzeugen sollen [44]. Es gibt, wie später beschrieben, auch Ansätze, welche ohne direkte Überwachung arbeiten. Das biologische Vorbild ist das Gehirn, speziellen das des Menschen. Es beschreibt ein hochkomplexes, nichtlineares und paralleles Informationsverarbeitungssystem. Das Gehirn besteht aus mehreren Neuronen, welche über Synapsen miteinander verbunden sind und Informationen austauschen und speichern [45]. Durch die Autoren werden neuronale Netze wie folgt beschrieben:

> *„Massiv parallel arbeitender Prozessor, bestehend aus einfachen Verarbeitungseinheiten mit der natürlichen Eigenschaft Wissen zu speichern und anzuwenden.“* [45, S. 2]

Zwei Eigenschaften sind für die Nachbildung des Gehirns von Bedeutung. Zum einen ist das die Fähigkeit, Wissen aus der Umgebung des Netzwerkes in einem Lernprozess zu erwerben. Und zum anderen das Speichern des erworbenen Wissens durch Trainieren der Gewichte auf den zwischenneuronalen Verbindungen.

Neuronale Netze müssen für den jeweiligen Zweck angepasst und für bestimmte Aufgaben entworfen werden. Ziel der Netze ist die Erfüllung einer Funktion, welche folgend als Zielfunktion f^* bezeichnet wird und durch das Netz möglichst genau zu approximieren ist. Die Zielfunktion enthält eine Variable y, welche die Ausgabe darstellt. Somit ist eine Zuordnung, wie in Formel 21 zu definieren.

$$y = f^*(x, \theta) \tag{21}$$

Dabei sind x die Eingabewerte in das Netz und θ dessen Parameterset, welches gelernt wird [44]. Mit jeder Berechnung der Zielvariable y und dem daraus entstandenen Fehler werden die Parameter in θ angepasst, wodurch das Netz lernt die Zielfunktion zu approximieren.

3.3.1.1 Aufbau

Die Grundlage für den Aufbau neuronaler Netze ist eine Architektur in Schichten. Diese Schichten sind vollverknüpft mit den vorangehenden als auch nachfolgenden Schichten. Abbildung 5 zeigt eine exemplarische Struktur eines einfachen Netzes. Ein neuronales Netz bekommt N Eingabewerte x, welche durch eine indizierte Menge dargestellt werden. Einfachsten Falls werden eindimensionale Daten in das Netz gespeist, wobei die Eingabe ein Vektor nach der Formel 22 ist. Anderen Falls können mehrdimensionale Daten verarbeitete werden, welche dann in Matrizen oder Tensoren übergeben werden. Formel 23 stellt das Schema einer Eingabematrix für zweidimensionale Daten dar. Die jeweiligen x_i stellen die einzelnen Features der Eingabestruktur x dar.

$$x = (x_1, x_2, \dots, x_N)^T \tag{22}$$

$$x = \begin{pmatrix} A_{1,1} & \cdots & A_{1,M} \\ \vdots & \ddots & \vdots \\ A_{N,1} & \cdots & A_{N,M} \end{pmatrix} \tag{23}$$

Eingabewerte werden durch das Netz von der Eingabe- bis zur Ausgabeschicht durchgereicht. Dieser Schritt wird als Feed Forward bezeichnet. Die Eingabewerte werden in jedem Neuron der einzelnen Schichten verarbeitet und an die nächste Schicht weitergegeben, bis die Ausgabeschicht erreicht ist. Jede Schicht von Neuronen kann als einzelne Funktion f(x) angenommen werden. Damit lassen sich die verbundenen Schichten als verkettete Funktion nach der Formel 24 darstellen. Es wird ein dreischichtiges Netz dargestellt, wobei $f^{(1)}$die Funktion der ersten Schicht und $f^{(2)}$ und $f^{(3)}$ jeweils die der zweiten und dritten Schicht sind. Sie nutzen jeweils das Ergebnis der vorhergehenden Schicht als Eingabe. [44]

$$y = f^{(3)}\left(f^{(2)}\left(f^{(1)}(x)\right)\right) \tag{24}$$

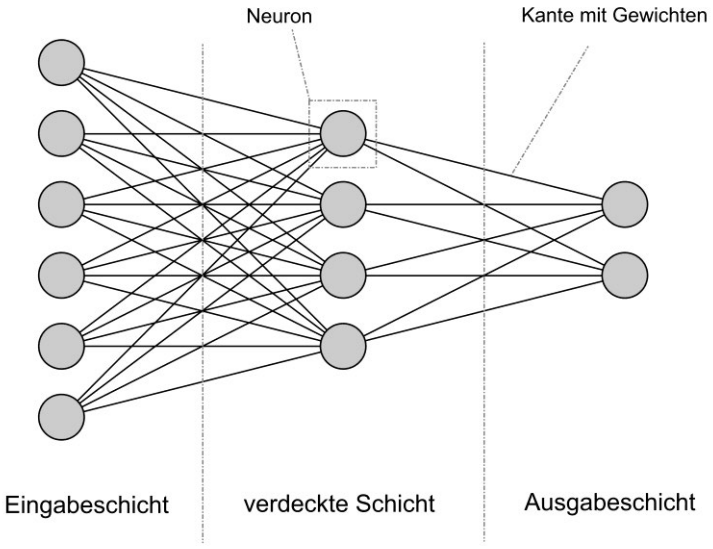

Abbildung 5: Schematischer Aufbau eines künstlichen neuronalen Netzes. Als Eingaben werden links 6 Werte eingegeben und rechts 2 Werte ausgegeben nach [24]

Der wichtigste Bestandteil eines neuronalen Netzes ist dessen kleinste Einheit, das Neuron. Es ist eine informationsverarbeitende Einheit als Fundament für neuronale Netze. Jedes Neuron bekommt Eingabesignale, verarbeitet diese und gibt sie an die nächsten Neuronen oder als Ausgabe weiter. Der Aufbau eines Neurons ist schematisch in Abbildung 6 dargestellt. Elementar besteht es aus drei Bestandteilen. Der erste ist die Sammlung von Synapsen zu anderen Neuronen. Diese Eingangssynapsen werden mit Gewichten belegt und stellen in Summe das Wissen eines Netzes dar. Ein Eingabesignal x_j über die Verbindung j an einem Neuron k wird mit dem synaptischen Gewicht w_{kj} gewichtet. Der zweite Bestandteil ist die Funktion der Neuronen. Im einfachsten Fall ist dies eine Summenfunktion über alle gewichteten Eingaben, wie in Formel 25 zu sehen ist. [44]

$$u_k = \sum_{j=1}^{m} w_{jk} x_j \tag{25}$$

Auf das Ergebnis u_k wird der dritte Bestandteil angewendet, die Aktivierungsfunktion $\varphi(\cdot)$, welche die Ausgabestärke der Funktion normalisiert. Das Ergebnis u_k wird zudem mit einem Bias b_k angereichert. Dieser ist ein Parameter, welcher die Eingabe von u_k in die Aktivierungsfunktion steuert. Mit der Aktivierungsfunktion berechnet sich die Ausgabe y_k des Neurons k nach der Formel 26.

$$y_k = \varphi(u_k + b_k) \tag{26}$$

Typische Aktivierungsfunktionen sind die Sigmoid-Funktion, ReLU (Rectified Linear Unit) und die Stufenfunktion. Sigmoid transformiert Eingabewerte x auf einen Ausgabewert y im Intervall $y \in [0,1]$. Damit werden auch Werte $x \ll 0$ und $y \gg 1$ auf das Intervall transformiert. Die Stufenfunktion hingegen projiziert x in eine binäre Menge $y \in \{0,1\}$. ReLU ist eine der am häufigsten verwendeten Aktivierungsfunktionen in modernen neuronalen Netzen. Sie hat den Effekt, dass alle negativen Eingaben auf 0 transformiert werden und sonst der Eingabewert unverändert zurückgegeben wird. Abbildung 7 zeigt die drei Funktionen graphisch.

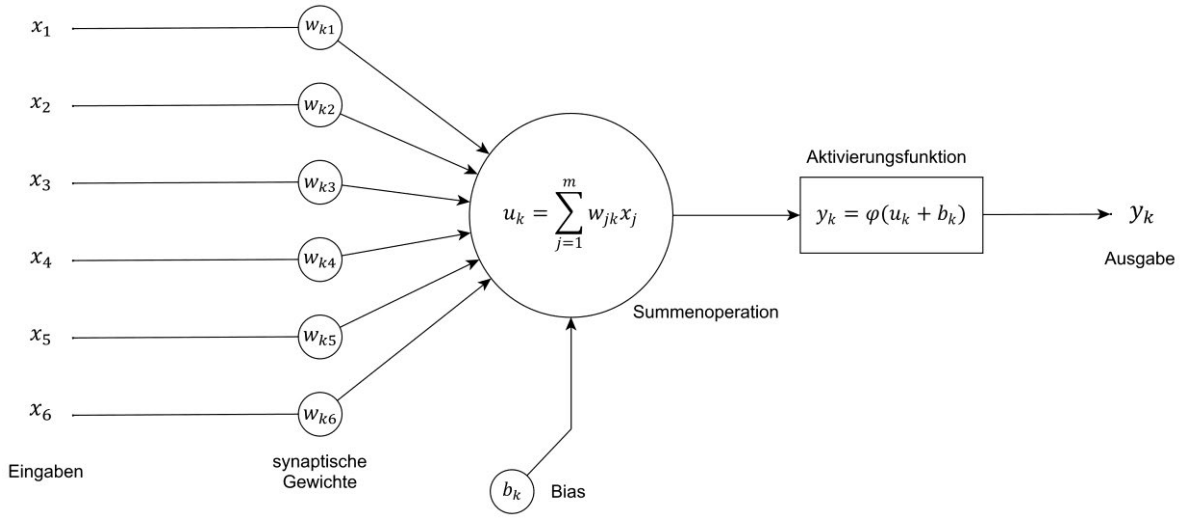

Abbildung 6: Aufbau eines Neurons in einem neuronalen Netzwerk nach [24]

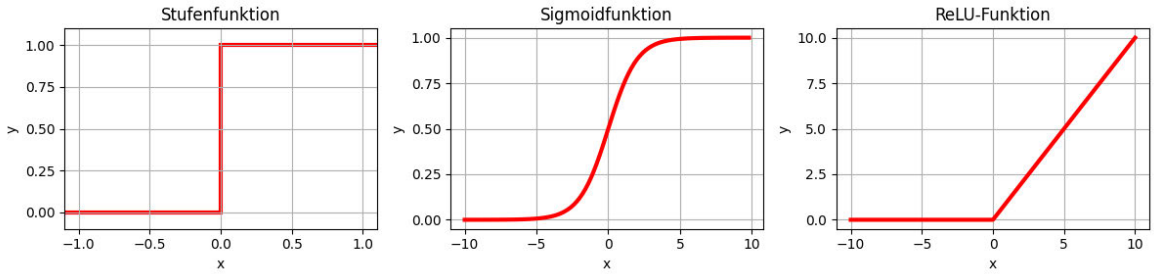

Abbildung 7: Drei typische Aktivierungsfunktionen

3.3.1.2 Lernprozess

Die bisher beschriebene Modellierung beschreibt nur den Fall, dass Eingaben vorwärts durch ein Netz geführt werden und am Ende einen Ausgabewert erzeugen. Damit ist der Lernprozess des Modells nicht beschrieben und kein Lerneffekt zu verzeichnen. Dieser Effekt tritt durch die Rückpropagierung (Backpropagation) ein, wobei die Daten rückwärst durch das Netz geführt werden.

Grundlage für die Optimierung der Gewichte eines Netzes und damit des Lernens ist eine Kostenfunktion (auch Ziel- oder Fehlerfunktion, engl.: loss function). Diese wird nach jedem Trainingsschritt neu ermittelt und zur Berechnung des erzeugten Fehlers durch die Ausgabe des Netzes genutzt. In modernen Ansätzen wird dies durch die Kreuzentropie zwischen den erwarteten und tatsächlichen Ausgabewerten realisiert. Ausgangspunkt ist das Erzeugen einer Vorhersage y des Modells anhand der Parameter θ und der Eingaben x während des Trainings. Der berechnete Wert wird mit dem erwarteten Wert $\mathbb{E}_{(x,y)}$ in der definierten Fehlerfunktion verrechnet und ergibt den zugehörigen Fehler. Damit werden die Gewichte des Netzes angepasst. Die definierte Kostenfunktion wird als $J(\theta)$ bezeichnet und ist durch den Lernprozess zu minimieren. Die Gleichung 27 zeigt exemplarisch das Schema einer möglichen Kostenfunktion. [44]

$$J(\theta) = \mathbb{E}_{(x,y)} L(f(x,\theta), y) \tag{27}$$

Das Minimieren des erzeugten Fehlers wird oft durch das Gradientenabstiegsverfahren umgesetzt. Es werden partielle Ableitungen der Zielfunktion gefunden und der steilste Abstieg der Kostenfunktion gesucht. Somit können für jedes Gewicht w_{kj} neue Werte gefunden werden, die $J(\theta)$ minimieren.

Der Lernprozess basiert auf den vorgegebenen Labeln des Trainingsdatensatzes. Die Trainingsbeispiele zeigen der Ausgabeschicht direkt an, welches Verhalten sie bei jedem einzelnen Beispiel x zeigen soll. Damit beeinflusst das optimale Label y^* das Verhalten der Ausgabeschicht, nicht aber das der verdeckten Schichten. Daher die Bezeichnung als verdeckte Schicht, da sie von außen durch das Training nicht direkt steuerbar und deren Ausgaben nicht ersichtlich sind. Die Gewichte werden rein durch die Optimierung anhand der Zielfunktion beeinflusst.

3.3.1.3 Vertreter zur Feature-Selektion

Auch im Bereich des NLPs sind neuronale Netzwerke State-of-the-Art-Ansätze zur Klassifizierung von Texten. Allerdings gibt es Verfahren, welche sich nicht auf die Klassifikation von Texten, sondern auf die Aufbereitung der Textdarstellung konzentrieren. Einige Vertreter sollen folgend kurz vorgestellt werden.

Grundlage für viele Ansätze in der Feature-Selektion sind Word Embeddings (Worteinbettungen) oder auch „short dense vectors“. Diese stellen jedes Wort als einen Vektor dar mit einer Dimension $50 < d < 1000$ je nach Implementation. Damit weisen sie deutlich weniger Dimensionen auf als Vokabulare, die den meisten Textkorpora zugrunde liegen. [46] Werte, die in den Einbettungsvektoren gespeichert werden, sind für Menschen nicht interpretierbar, daher auch deren Bezeichnung als hidden oder dense Vektoren. Sie zielen außerdem darauf ab, Ähnlichkeiten verschiedener Wörter darzustellen. Dies geschieht im sog. Embedding Space, was ähnlich dem Vektorraummodell aus Kapitel 3.1.4 konzipiert ist. Synonyme oder ähnliche Begriffe werden auch als Vektoren mit ähnlichen Werten eingebettet. Worteinbettungen können statisch oder dynamisch sein.

Statische Modelle legen für jedes Wort einen Vektor fest, welcher fortlaufend als fix angenommen wird [47]. Ein Beispiel dafür ist Word2Vec, welches Wörter anhand eines neuronales Netzwerkes in Vektoren im Embedding Space kodiert. Dessen Ablauf ist wie folgt zu beschreiben:

1. Wähle sequenziell Wörter w_i und dessen Nachbarkontext c als positive Beispiele aus der Eingabesequenz aus.
2. Wähle zufällige andere Wörter aus dem Korpus als negative Beispiele
3. Trainiere binären Klassifikator nach der Frage: „Ist es wahrscheinlich, dass ein Wort c nahe dem Zielwort w auftritt?
4. Nutze die gelernten Gewichte des neuronalen Netzes als Worteinbettungen.

Das durch Mikolov et al. [47] beschriebene Verfahren ermöglicht eine Selbstüberwachung des Algorithmus und vermeidet es Trainingsdaten händisch zu labeln.

Beispiele für dynamische Einbettungsverfahren sind ELMo (Embeddings from Language Models) und BERT, welches im folgenden Kapitel näher beschrieben wird. Die Dynamik der Modelle besteht darin, dass sich die Einbettung abhängig vom Kontext, in dem sie auftreten verändern. Es ermöglicht mit einem Wort dessen umgebenden Kontext zu kodieren [46].

3.3.2 BERT

BERT ist ein Deep Learning Algorithmus, welcher 2019 vom Google Research Team vorgestellt wurde [48]. Besonderheit ist die Einbettung einzelner Wörter, welche abhängig vom umgebenden Kontext ist. Die Autoren der Publikation heben hervor, dass BERT damit Nachteile älterer Modelle mit Vortraining behebt, die teilweise nur den linken Kontext von Wörtern einbeziehen. Dazu zählt z.B. OpenAI GPT [49]. BERT bedeutet Bidirectional Encoding Representations from Transformers (dt.: bidirektionale Kodierungsrepräsentationen von Transformern) und bezieht den beidseitigen Kontext ein.

BERT-Modelle sind standardmäßig vortrainiert auf einem Textkorpus von ca. 3,3 Mrd. Wörtern aus dem BooksCorpus [50] und englischen Wikipedia Artikeln[3]. Das Modell basiert auf einer im Jahr 2017 von Vaswani et al. [51] vorgestellten Architektur namens Transformer. Es verwendet einen Attention-Mechanismus, um globale Abhängigkeiten zwischen Ein- und Ausgabe herzustellen. In den folgenden Unterkapiteln wird die Architektur von BERT dargestellt. Dabei werden Besonderheiten des Algorithmus aufgearbeitet und dessen Training genauer erläutert. Schließlich werden die Vorverarbeitungsschritte für BERT und einige Möglichkeiten zur Feinabstimmung gezeigt.

3.3.2.1 Transformer-Architektur

Die Transformer-Architektur wurde 2017 von Vaswani et al. mit dem Ziel veröffentlicht, eine schnelle und effiziente Lösung für maschinengestützte Übersetzungen zu schaffen. Zu diesem Zeitpunkt wurde diese Aufgabe oftmals von RNNs (Recurrent Neural Networks) und LSTMs (Long-short Term Memory) übernommen. Diese weisen lange Verarbeitungszeiten und Probleme bei langen Sätzen auf.

Transformer bestehen aus zwei Bestandteilen: Kodierer und Dekodierer. Abbildung 8 zeigt die Architektur nach [51]. Jedes der beiden Bestandteile besteht aus einem Set von $N = 6$ identischen Schichten. Der Kodierer teilt sich in die Unterschichten Multi-Head Attention und Feed-Forward Netzwerk auf. Der Dekodierer hat als erste Schicht noch eine maskierte Multi-Head Attention Schicht und ist danach identisch zum Kodierer.

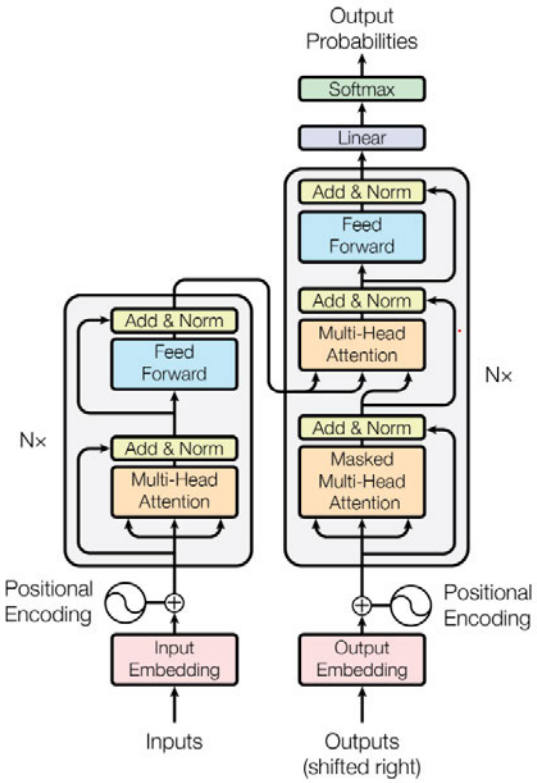

Abbildung 8: Transformer-Architektur. Kodierer (links) und Dekodierer (rechts) nach [48]

[3] https://en.wikipedia.org/wiki/English_Wikipedia

Die Eingaben für die Kodierer sind Worteinbettungen $x = \{x_1, x_2, \dots, x_n\}$ einer Phrase, die in eine andere Sprache zu übersetzen ist. Diese Eingabe wurde bereits durch Algorithmen wie Word2Vec in positionelle Einbettungen kodiert, um die Ähnlichkeit synonymer Wörter zu repräsentieren. Jedes Wort wird folgend durch einen Vektor $x \in \mathbb{R}^H$ gekennzeichnet. H steht für die Hidden Size des Algorithmus und definiert die Dimension der verdeckten Wortvektoren. Durch Vaswani et al. wird $H = 512$ gesetzt.

Die positionellen Einbettungen werden in die Attention-Schicht geleitet. Attention ermöglicht es die Referenzen verschiedener Wörter einer Eingabesequenz zueinander zu erfassen. Ausgabe ist der Vektor $z = \{z_1, z_2, \dots, z_n\}$, wobei $z \in \mathbb{R}^H$ gilt. Nach jeder Schicht in der Transformer-Architektur erfolgt eine Normierung der Ausgabevektoren, mit deren Eingabevektoren. z dient als Eingabe für ein neuronales Feed-Forward Netz mit einer verdeckten Schicht. Darauf erfolgt ebenso eine Normierung des Ausgabevektors.

Die Dekodierung der Vektoren funktioniert auf gleiche Art und Weise, wie die Kodierung. Bei der Übersetzung einer Sequenz in eine andere Sprache sind die Eingaben des Dekodierers die gleiche Sequenz in der Zielsprache. Die Ausgabe des Dekodierers ist eine Wahrscheinlichkeit für das entsprechende Wort im Vokabular der Zielsprache. Transformer stellen in der Trainingsphase eine Relation eines Wortes zwischen zwei Sprachen her. Die zusätzliche maskierte Attention-Schicht funktioniert, wie die übrigen Attention-Schichten mit der Ausnahme, dass sie nur auf linksseitige Wörter des aktuellen Tokens zugreifen kann.

Der letzte Schritt der Dekodierung ist eine lineare Schicht, welche die ausgegebene Matrix des Dekodierers in einen Wortvektor mit der Größe des Vokabulars der Zielsprache übersetzt. Danach beschreibt jedes Element i des Vektors den Score für ein Wort w_i im Vokabular der Zielsprache. Durch das Anwenden einer Softmax-Funktion werden die Scores in Wahrscheinlichkeiten umgewandelt. Das Element mit der höchsten Wahrscheinlichkeit bezeichnet den Index i des Wortes in der Zielsprache. [51]

3.3.2.2 Self-Attention

Das Prinzip der Self-Attention bezieht sich darauf, Wörter in einer Sequenz zu betrachten und zu entscheiden, welches dieser Wörter mit dem aktuell betrachteten Wort die Verbindung eingeht. Abbildung 9 zeigt schematisch die Berechnung der Self-Attention.

Ausgangspunkt für die Berechnung sind die Einbettungsvektoren. Diese werden zu einer Einbettungsmatrix X konkateniert, da Attention mit Matrixoperationen berechnet wird. Der Berechnung liegen drei Gewichtsmatrizen zugrunde: Query-Gewichtsmatrix W_Q, Key-Gewichtsmatrix W_K und Value-Gewichtsmatrix W_V. Sie werden im Laufe des Trainings angepasst. Die Gewichtsmatrizen dienen dazu aus der Einbettungsmatrix durch Matrixmultiplikation jeweils drei weitere Matrizen zu bilden: Query-Matrix Q, Key-Matrix K und Value-Matrix V. Diese werden anschließend mittels Softmax und einer Multiplikation zur Ausgabematrix Z berechnet. Formel 28 zeigt diese Berechnung. Dabei ist d_k die Dimension der Vektoren in Q, K, V, welche standardmäßig auf 64 gesetzt ist. $\sqrt{d_k}$ dient zum Skalieren der Matrixmultiplikation.

$$Z = Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{28}$$

Die bisher beschriebene Verfahrensweise sieht eine skalierte Produkt Attention vor, was bedeutet, dass diese nur einmal berechnet wird. Die Transformer-Architektur nach [51] sieht eine Multi-Head Attention vor. Das wird erreicht, indem für jedes Token in mehreren Heads, also den skalierten Produkten, der Vektor z berechnet wird. Schließlich werden die einzelnen Heads konkateniert und mit einer Gewichtsmatrix W^0 zur finalen Matrix Z multipliziert. Für die Multi-Head Attention gilt die Formel 30.

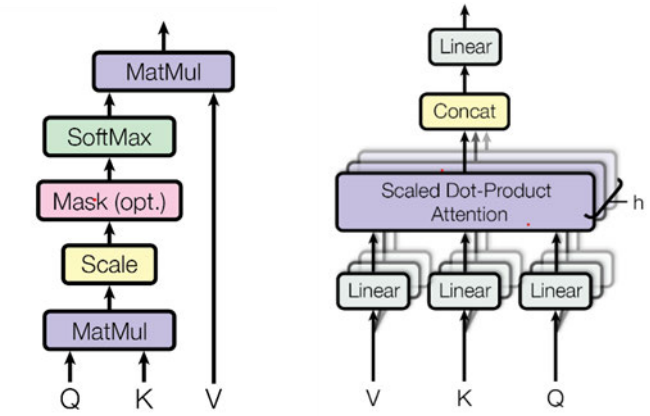

Abbildung 9: Schema der Self-Attention. Die Multihead-Attention (rechts) besteht aus h skalierten Produkt Attentionen (links) [48].

$$head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right) \tag{29}$$

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h)W^0 \tag{30}$$

Die Matrizen W_i^Q, W_i^K und W_i^V sind die Gewichtsmatrizen für die jeweiligen Heads i. h beschreibt die Anzahl der Heads, welche durch Vaswani et al. standardmäßig auf 8 gesetzt wird. [51]

3.3.2.3 Funktionsweise von BERT

BERT basiert grundlegend auf der Transformer-Architektur, wobei das Ziel von BERT nicht die Übersetzung einer Sprache in eine andere ist. Die Aufgabe von BERT ist aussagekräftige Worteinbettung zu generieren mit denen NLP-Task effizient gelöst werden können. Aufgrund dessen beschränkt sich BERT auf den Kodierer der Architektur nach Vaswani et al. [51]. Der Aufbau von BERT ist in Abbildung 10 zu sehen.

Das Modell durchläuft zwei Phasen, bevor das zu lösende Problem bearbeitet wird. Das Vortraining (Pre-Training) dient dazu, dem Modell Sprache zu lernen und Kontexte von Sätzen zu verstehen. Phase zwei, die Feinabstimmung (Fine-Tuning), richtet sich nach dem Task auf dem BERT angewendet werden soll. Sie dient dazu Parameter des Modells auf das Problem auszurichten und domänenspezifische Zusammenhänge zu lernen.

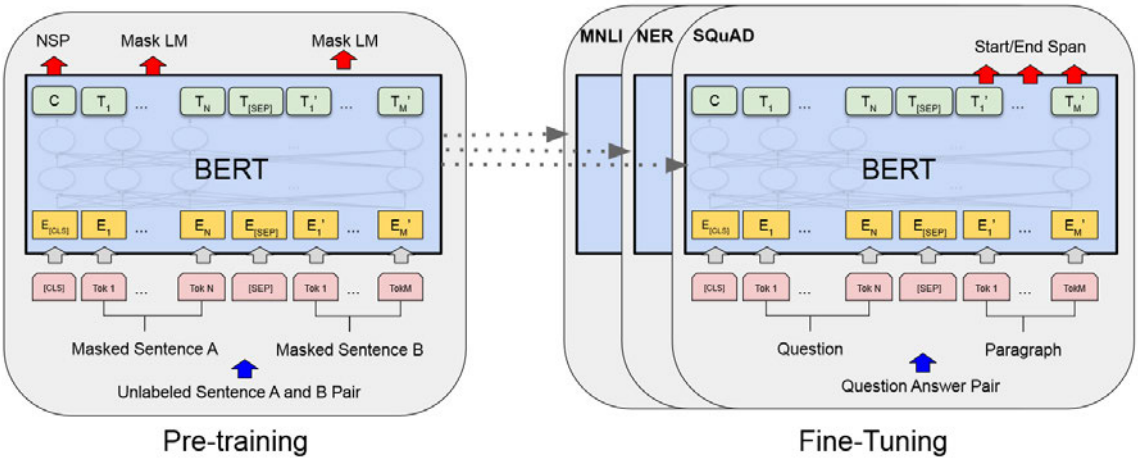

Abbildung 10: Schema von BERT. Das Vortraining (links) besteht aus zwei Aufgaben, die gleichzeitig gelöst werden. Dieses Verfahren ist Ausgangspunkt für jedes BERT-Modell, unabhängig vom zu lösenden Task (rechts) [48]

Die Kodierer werden mit einer Größe L für Anzahl der Layer (Schichten) aneinandergereiht. Ein weiterer Parameter ist H, welcher die Dimension der verdeckten Einbettungen eines Wortes beschreibt. A ist die Anzahl der Attention-Heads, welche gleichzeitig zur Berechnung der Attention genutzt werden. Für BERT sind durch Entwickler Devlin et al. [48] zwei Modelle veröffentlicht wurden. Entsprechenden Parameter werden in Tabelle 7 dargestellt.

Eingabe für BERT ist eine rohe Textsequenz. Diese wird mittels WordPiece in Form von Token zu Worteinbettungen E_i umgewandelt [52]. Jede Sequenz bekommt ein Token [CLS] an erster Stelle eingefügt. Es ist ein spezielles Token, was den Anfang einer Sequenz markiert und später für Next Sentence Prediction zur Klassifikation genutzt wird. Das Token wird nach Abbildung 9 als $C \in \mathbb{R}^H$ bezeichnet. Das letzte Token einer Sequenz ist stets [SEP] für Separator. Außerdem wird das Symbol zur Trennung verwendet, wenn zwei Sätze in einer Sequenz enthalten sind. Alle anderen Tokens werden mit $T_i \in \mathbb{R}^H$ notiert.

Nach der Tokenisierung und der Einbettung durch WordPiece werden die Positionseinbettung und die Einbettung des entsprechenden Segments der Sequenz addiert. Schematisch wird dieser Ablauf in Abbildung 9 dargestellt. Die Positionseinbettungen geben die Stelle i des Tokens in der Sequenz wieder. Das Segment gibt hingegen an, ob sich ein Satz in Segment A oder B befindet, was für Frage-Antwort-Systeme und Next Sentence Prediction von Relevanz ist.

Nach der Vorverarbeitung der Tokens verfährt BERT nach dem gleichen Ablauf, wie er für den Kodierer nach Vaswani et al. beschrieben wird. Die finalen Ausgaben von BERT werden als verdeckte Vektoren eines Tokens (hidden) $h_{Token} = T_i \in \mathbb{R}^H$ bezeichnet und werden für jedes Wort der Eingabesequenz berechnet.

Tabelle 7: Unterschiede der beiden BERT-Modelle nach Devlin et al. nach deren Aufbau und Anzahl der Parameter nach [48]

	Layer L	**Hidden H**	**Attention A**	**Parameter Total**
$BERT_{Base}$	12	768	12	Ca. 110 Mio.
$BERT_{Large}$	24	1024	16	Ca. 340 Mio.

3.3.2.4 Vortraining

Das Vortraining besteht aus zwei Unteraufgaben, welche gleichzeitig durch das Modell unüberwacht durchgeführt werden. Die erste ist das Masked Language Modelling (MLM). Dafür wird ein bestimmter Prozentsatz der WordPiece Einbettungen maskiert durch [MASK]. Devlin et al. verwenden dabei 15% der Wörter. Die Ausgabevektoren des Kodierers der maskierten Tokens werden in eine Softmax-Funktion über das gesamte Vokabular gegeben, um die [MASK]-Tokens zu bestimmen. Damit werden nur die fehlenden Wörter vorhergesagt und gleichzeitig deren Kontext zu anderen Wörtern der Sequenz gelernt. Problematisch ist, dass es bei er Feinabstimmung und darüber hinaus keine [MASK]-Tokens mehr gibt. Um diesen Fall zu umgehen, werden nur 80% der maskierten Tokens tatsächlich maskiert, 10% durch zufällige andere Tokens ersetzt und 10% unverändert verarbeitet. Damit muss BERT bestimmen, ob die Wörter an den Stellen Sinn machen, also wahrscheinlich sind. Somit wird der Kontext in einer Sequenz gelernt.

Die zweite Aufgabe ist die Next Sentence Prediction (NSP), welche sich mit dem Kontext zweier Sätze beschäftigt. Ziel ist es zu klassifizieren, ob ein gegebener Satz B Folgesatz von Satz A ist. Das führende [CLS]-Token wird mit der binären Entscheidung gelabelt. Beide Sätze A und B werden durch das [SEP]-Token getrennt in eine Eingabesequenz geschrieben. Dabei ist in 50% der Fälle Satz B Folgesatz von Satz A und in 50% der Fälle nicht.

Anhand des Vortrainings ist BERT in der Lage, Wörter in einem Satz vorherzusagen, da es den Kontext um das Wort kennt und das wahrscheinlichste Wort aus dem Vokabular einsetzen kann. Andererseits kann BERT entscheiden, ob ein Satz B zu einem vorhergehenden Satz A passt oder nicht und versteht so auch Zusammenhänge über Satzgrenzen hinweg.

3.3.2.5 Feinabstimmung

Nach dem Vortraining von BERT ist es bereits in der Lage NLP-Probleme zu lösen, da es gelernt hat, wie Sprache funktioniert. Allerdings fehlt dem Modell noch domänenspezifisches Wissen, welches mit der Feinabstimmung erzielt wird. So kann BERT auf eine bestimmte Aufgabe fokussiert werden und bessere Ergebnisse erzielen.

Nach Sun et al. [53] gibt es für diesen Schritt verschiedene Strategien. Diese sind in Abbildung 10 durch die farbigen Pfeile dargestellt. BERT kann durch das Verfahren des Vortrainings weiter auf Daten der Domäne trainiert werden, in der das später zu lösende Problem angesiedelt ist. Dazu können genau die Daten des Problems verwendet werden, ähnliche Daten oder Daten verschiedener Domänen. Multitask Finetuning nimmt mehrere Tasks einer Domäne als Grundlage, wohingegen sich Singletask nur auf das zu lösende

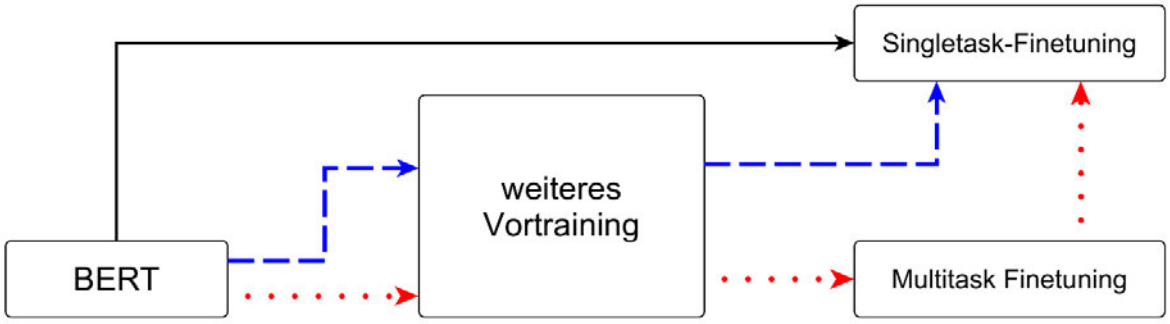

Abbildung 11: Schematischer Ablauf für die Feinabstimmung von BERT. Jede Linienfarbe zeigt eine mögliche Trainingsstrategie des bereits vortrainierten Modells nach [53]

Problem konzentriert. Sun et al. empfehlen ein möglichst ausgiebiges Training des Modells, so empfehlen sie den roten Weg in Abbildung 10. Sie warnen allerdings davor, dass durch zu intensives Training der Algorithmus schnell zur Überanpassung an die Trainingsdaten neigt, besonders im Singletask-Finetuning.

Mit BERT haben Devlin et al. ein Modell entwickelt, welches schnell und effizient genaue Repräsentationen abhängig vom Kontext lernen kann. Viele NLP-Tasks wurden bereits mit BERT durchgeführt und zeigen, dass das Modell hervorragende Ergebnisse bringt. Weiterhin wurden seit der Veröffentlichung der Publikation von Devlin et al. viele spezifizierte und abgeänderte Modelle im Internet verfügbar. Darunter auch für Sprachen außer Englisch und vortrainiert für Aufgaben auf bestimmten Domänen.

4 Experiment Setup

Das folgende Kapitel befasst sich mit der Beschreibung der Experimente auf Basis der Identifikation von gerichteter offensiver Sprache in Twitter Posts nach dem Vorbild von Zampieri et al. [11], [12]. Ziel ist ein Vergleich von Modellen zur Feature-Selektion. Dazu wurde aus den vorgestellten Arten von Ansätzen jeweils ein Vertreter eines Feldes ausgewählt, implementiert und auf den Datensatz angewendet.

Folgend wird der vorliegende Datensatz beschrieben und die Eigenschaften von Tweets analysiert. Mittels dieser Eigenschaften werden Vorverarbeitungsschritte erschlossen, die eine Klassifikation für die Modelle erleichtern sollen. Die Modelle werden mit ihren Parametern vorgestellt und deren Vorverarbeitung beschrieben, um die Reproduzierbarkeit der Ergebnisse zu gewährleisten.

Die Experimente werden auf einem Desktop PC mit Windows 10 64-Bit-Version, einem AMD Ryzen 5 3600 6 Kern Prozessor mit 16GB RAM und einer NVIDIA GeForce RTX 2070 Super mit 8GB dedizierten Speicher durchgeführt.

4.1 Datenbeschreibung

Zum Vergleichen der Ergebnisse wird für alle Modelle grundsätzlich der gleiche Datensatz verwendet. Dieser ist der OLID-Datensatz, welcher mit dem Paper von Zampieri et al. veröffentlicht wurde. Die Sammlung von Tweets erfolgt durch die Autoren anhand von Schlüsselwörtern, die sich besonders auf politische Ereignisse in Amerika beziehen. Der Datensatz besteht aus vorgefertigten Test- und Trainingsdaten für jeden Subtask. Wegen der Beschaffenheit des Codes für HBBO wurden die Trainings- und Testdaten je Subtask konkateniert und in einem Datensatz abgespeichert. Die Aufteilung von Test- und Trainingsdaten erfolgt in der Laufzeit des Programmcodes. Sie wird in allen Ansätzen durch 5-Fold-Kreuzvalidierung umgesetzt. Das Annotationsschema wird unverändert nach der Beschreibung in Punkt 2.2 beibehalten.

4.1.1 Klassenverteilung

OLID ist ein hochqualitativer Datensatz mit einem Umfang von 14.100 englischen Tweets, welcher sich in drei Level aufteilt. Wie in Abbildung 12 dargestellt, ist der Datensatz in allen Subtasks nicht balanciert. Besonders schwerwiegend ist das Ungleichgewicht der Klassen in Subtsask B und C. Es sind deutlich weniger Beispiele für UNT und OTH vorhanden als für die jeweils dominierenden Klassen. Deshalb sind Evaluationsmaße, wie die Accuracy sehr fehleranfällig. Durch diese Gegebenheit werden die Experimente für TFIDF-SVM und BERT jeweils einmal mit balancierten Klassengewichten durchgeführt und einmal ohne diese. Bei allen Experimenten werden keine Sampling-Techniken angewandt, um eine Klasse künstlich zu vermehren und so den Datensatz zu balancieren.

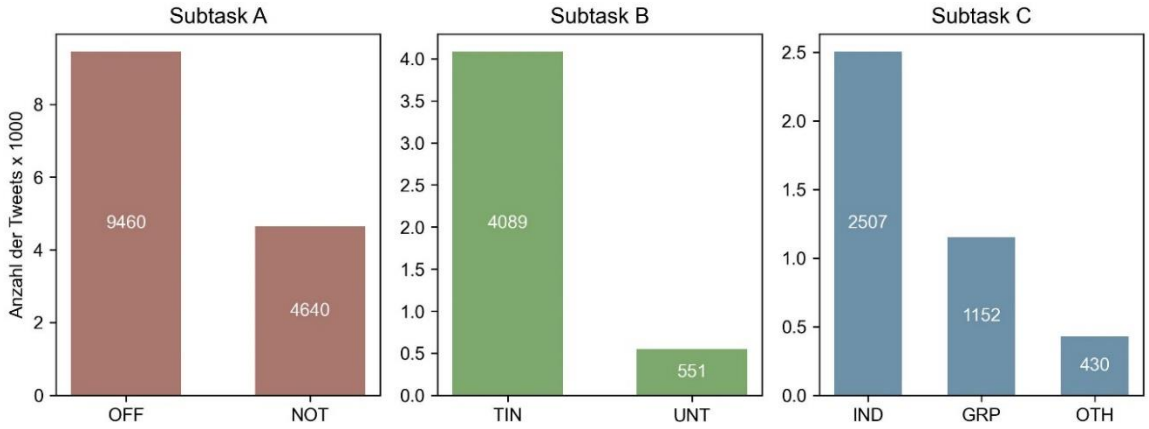

Abbildung 12: Verteilung der Klassen in den drei Subtask

4.1.2 Charakteristika von Tweets

Twitter ist im Bereich der offensiven Sprache eine häufig adressierte Plattform. Wie bereits in Kapitel 2.1 beschrieben, weisen die Autoren Nakov et al. [19] darauf hin, dass Anwendungen auf die Plattform angepasst werden sollen, auf der sie verwendet werden. Trotz der häufigen Verwendung von Twitter als Quelle für Datensätze sind einige Aspekte zu beachten, um möglichst effektive Klassifikationsmodelle zu trainieren.

Nachrichten, die in sozialen Netzwerken ausgetauscht werden, folgen oft nicht der standardisierten Schriftsprache. Baldwin et al. [21] analysierten dieses Phänomen genauer. Sie stellten fest, dass unter anderem Tweets durchschnittlich viele Schreibfehler enthalten, sowie grammatikalische Schwächen aufweisen. Grammatikalische Schwächen sind für Modelle, wie SVM und HBBO unproblematisch, da sie die Reihenfolge der Wörter nicht beachten. BERT hingegen analysiert den bidirektionalen Kontext eines Wortes, wodurch bei grammatikalischen Fehlern die Qualität der Einbettungen beeinträchtigt wird.

Abbildung 13: Beispielhafter Tweet aus OLID (leicht modifiziert zur besseren Veranschaulichung)

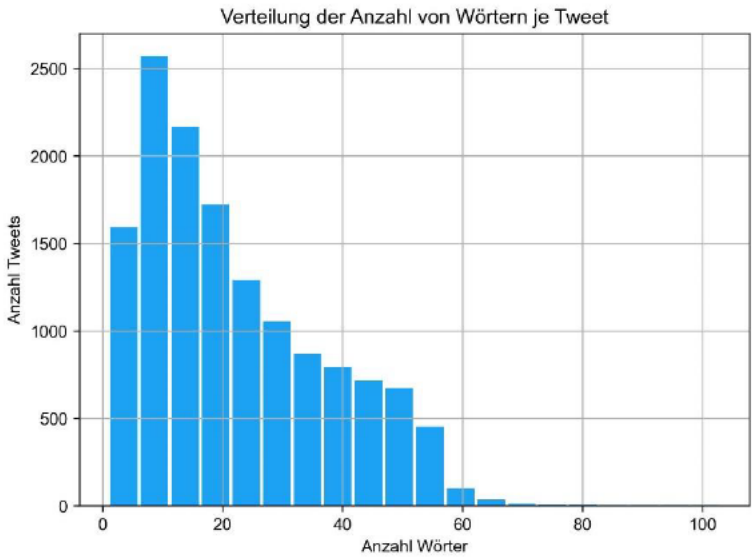

Abbildung 14: Histogramm über die Verteilung der Wortanzahl pro Tweet

Abbildung 13 zeigt beispielhaft einen Tweet, wie er im OLID-Datensatz vorkommt. Anzumerken ist, dass nur der geschriebene Text in OLID enthalten ist. Alle anderen Informationen (Likes, Autor, etc.) werden nicht gespeichert. Besonders auffällig ist, dass die Tweets nur kurze Texte sind. Der kürzeste Tweet im Datensatz besteht aus einem Wort, der längste aus 103. Tweets mit einer Anzahl zwischen 6 und 11 Wörtern sind am häufigsten. Abbildung 14 zeigt ein Histogramm zur Verteilung der Wortanzahl pro Tweet. Allgemein haben die meisten Tweets eine Länge zwischen 5 und 60 Wörtern. Das arithmetische Mittel liegt bei ca. 22 Wörtern pro Tweet.

Besonderheiten von Tweets sind Hashtags und Usermentions (dt.: Nutzererwähnungen). Letztere sind in OLID schon vorverarbeitet und werden aufgrund datenschutzrechtlicher Gegebenheiten durch „@USER“ repräsentiert. Hashtags hingegen bleiben unverändert im Datensatz enthalten. Sie stellen eine Konkatenation von Wörtern dar, die sich in Summe auf ein bestimmtes Thema beziehen. Gekennzeichnet sind diese durch eine führende Raute am Anfang der Wortgruppe. Eine weitere Besonderheit sind Emojis. Für Modelle wie BERT sind sie unbekannte Tokens, die erst in das Vokabular aufgenommen werden müssen. Das verlängert die Trainingszeit und kann besonders bei kleinen Datensätzen dazu führen, dass deren Bedeutung nicht durch das Modell erschlossen werden kann.

4.2 Vorverarbeitung von Texteingaben

Durch die beschriebenen Charakteristika von Nachrichten auf Twitter ergeben sich Möglichkeiten zur Vorverarbeitung der Eingabetexte. Wichtig ist, dass nicht alle in jedem Subtask angewendet werden müssen. Sie lassen sich beliebig kombinieren. Folgend werden die Vorverarbeitungsschritte dargelegt.

4.2.1 Emojis

Emojis sind Piktogramme, welche durch eine bestimmte Zeichenfolge repräsentiert werden. Sie ersetzen in Nachrichten auf sozialen Plattformen Wörter und drücken Emotionen aus. Kontextabhängige Modelle können Emojis schlecht interpretieren, was teilweise zu einer schlechteren Performanz führt.

Eine Möglichkeit um Emojis zu verarbeiten besteht im Entfernen dieser aus dem Datensatz. Damit geht zwar deren Bedeutung verloren, aber die Auswertung durch Modelle kann verbessert werden. Besonders wenn Emojis ironisch verwendet werden, können sie einen falschen Kontext aufzeigen. Auch können Emojis durch deren Bedeutung ersetzt werden. Dafür wurde in den Experimenten das Tool Demoji[4] verwendet.

4.2.2 Hashtags

Für Hashtags besteht ebenfalls die Möglichkeit des Entfernens. Hashtags haben allerdings eine Bedeutung inne, welche für Klassifikationsprobleme entscheidend sein kann. Sie können teils alleinig entscheiden, welche Klasse einem Tweet zugewiesen wird. Ein Hashtag welcher zum Großteil nur in angreifenden Tweets vorkommt ist „#antifa“. Durch das Entfernen des Hashtags wird die Entscheidung für das Modell möglicherweise schwieriger.

Alternativ können Hashtags segmentiert werden. Durch das Tool Wordninja[5] werden die Wortgruppen in einzelne Wörter zerlegt und können besser in das Vokabular des Klassifikators eingebaut werden. So wird das Hashtag „#damnliberals“ zu „damn“ und „liberals“.

4.2.3 Usermentions und URLs

Nutzererwähnungen zielen darauf ab, einen anderen Nutzer direkt mit einem Tweet in Verbindung zu bringen. Auffällig ist, dass in vielen Tweets mehr als ein anderer Nutzer referenziert wird. Aufgrund der Tatsache, dass alle Nutzererwähnungen durch „@USER“ ersetzt werden, kommen oft viele dieser Token hintereinander vor. Das führt zu einer Gewichtung, die ähnlich der eines Stoppwortes ist und vermindert die Bedeutung des Wortes. Auch bei diesem Merkmal besteht die Möglichkeit, es zu entfernen. Eine weitere Möglichkeit, welche sich als effektiv zeigt, ist das Vereinfachen der Nutzererwähnungen. Kommen mehrere hintereinander vor, werden diese auf ein einziges reduziert, indem alle auf das erste der Reihe folgenden gelöscht werden. Treten die Erwähnungen an mehreren Stellen eines Tweets auf, hat das beschriebene Prinzip keinen Effekt.

Für URLs besteht nur die Möglichkeit diese zu entfernen. Sie enthalten wenig Informationen und sind nicht vorteilhaft für die Klassifikation in allen drei Subtasks.

---

[4] https://pypi.org/project/demoji/

[5] https://github.com/keredson/wordninja

4.2.4 Kombination der Vorverarbeitungsschritte

Die Vorverarbeitungsschritte lassen sich in einem Datensatz beliebig miteinander kombinieren und zeigen verschiedene Effekte auf die Evaluationsergebnisse. In Vortests und -experimenten haben sich einige Kombinationen für einzelne Subtasks als sinnvoll bzgl. der Performanz erwiesen. Diese sollen kurz für den Subtask erläutert werden. Die Vorverarbeitungsschritte gelten anschließend für alle Experimente mit allen Modellen und werden für den jeweiligen Subtask immer gleich angewendet.

Subtask A profitiert vom Entfernen von Emojis. Das liegt vermutlich an der ironischen Verwendung dieser. Ebenso hat das Segmentieren von Hashtags und das Vereinfachen der Nutzererwähnungen einen positiven Einfluss auf das Evaluationsergebnis. Das Entfernen von URLs hat nur wenig Einfluss, wird aber dennoch für Subtask A angewendet.

Für Subtask B ist besonders das Vereinfachen der Nutzererwähnungen wichtig. Die Evaluationsmaße werden dadurch merkbar verbessert. Emojis werden durch deren Bedeutung gemäß dem Muster *:<Bedeutung>:* ersetzt, sodass die Bedeutung mit einem speziellen Separator markiert wird. Auch hier werden ULRs entfernt.

In Subtask C werden Emojis sowie URLs entfernt. Des Weiteren findet das Segmentieren von Hashtags Verwendung. Vermutlich kann anhand von Hashtags eine bessere Entscheidung bzgl. des Ziels getroffen werden.

4.3 BERT

Aufgrund der hervorragenden Performanz von BERT in vielen NLP-Tasks wird BERT als Vertreter neuronaler Ansätze für die Experimente herangezogen. BERT stellt zudem mit seiner Veröffentlichung in 2019 ein neues Modell dar und soll deswegen in dieser Arbeit Erwähnung finden.

BERT wird für die durchzuführenden Experimente mit PyCharm 2021.3 in Python 3.9 implementiert. Das vortrainierte Modell wird aus der Transformer-Bibliothek von Hugging Face bezogen. Es handelt sich um das BERT-base-uncased Standardmodell nach der beschriebenen BERT-Architektur mit 12 Encodern, 12 Attention-Heads und einer Hidden Size von 768. Zur Implementierung des neuronalen Netzwerkes und BERT wird Tensorflow 2.9.1 verwendet. Zum Erreichen stabiler Ergebnisse wird eine 5-Fold-Kreuzvalidierung angewandt. So wird auch eine Testgröße von jeweils 20% des zugrundeliegenden Datensatzes je Subtask erreicht. Das jeweils beste Modell eines Subtasks wird für die Klassifikation der Testdaten herangezogen.

Die Klassifikation der Einbettungen von BERT wird mit einem CNN (Convolutional Neural Network) umgesetzt, wie von [48] empfohlen. Es besteht aus zwei Eingabeschichten für die Input-IDs und die Attention-Masken. Auf die Eingabeschicht folgt ein Dropout-Layer mit einer Dropout-Rate von 0.1. Darauf folgt eine verdeckte Schicht mit einer Größe von 32 Neuronen und L1-Regulierung von Kernel und Aktivierung der Neuronen. Die Ausgabeschicht besteht für die Tasks A und B aus jeweils einem Neuron mit Sigmoid-Aktivierung. Für Subtask C besteht diese aus drei Neuronen für je eine der Klassen mit

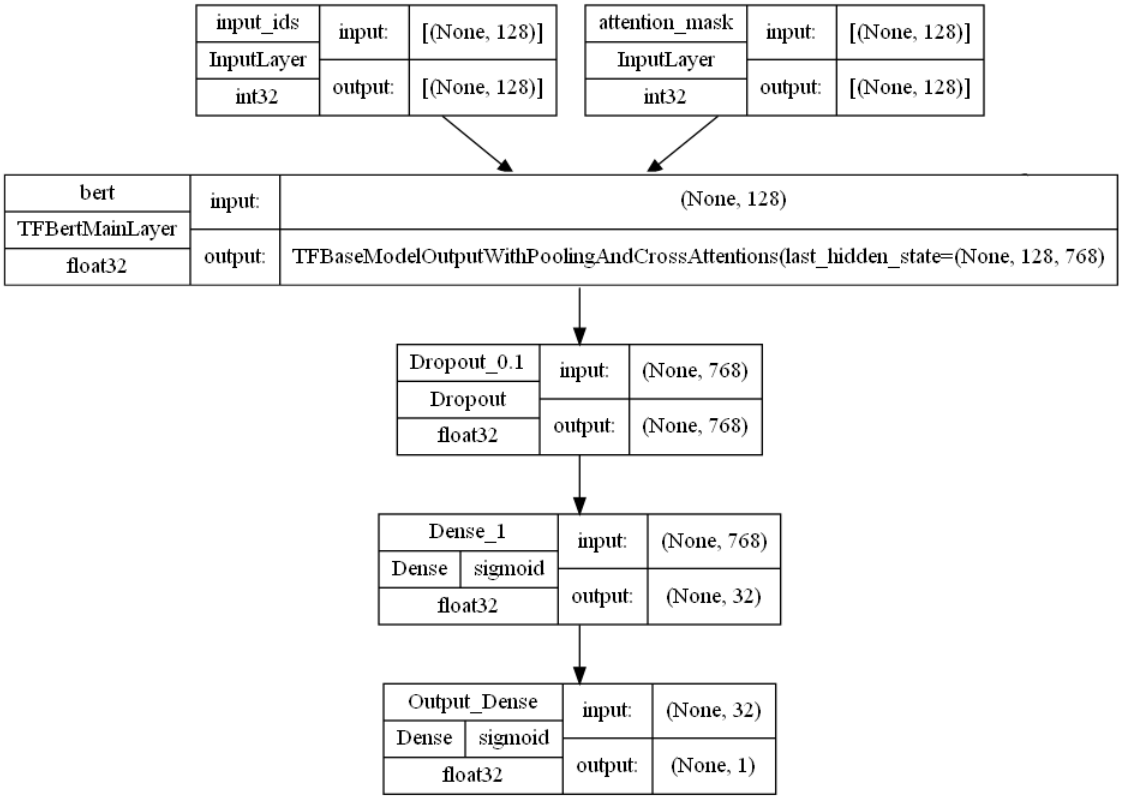

Abbildung 15: Darstellung des Modells für BERT. Die beiden Eingabeschichten geben die Vektoren in BERT, welcher die Embeddings berechnet und anschließend in das CNN leitet.

Softmax-Aktivierung. Für die Optimierung wird der Adam-Optimierer herangezogen. Der erzeugte Fehler wird mit der binären bzw. kategorialen Kreuzentropie berechnet. Eine Visualisierung des Modells von Tensorflow für die Tasks A und B ist in Abbildung 15 zu sehen. Für Subtask C wird die Ausgabe in die Form $(None, 3)$ abgeändert.

4.3.1 Vorverarbeitung

Die Vorverarbeitung von BERT ist mit wenig Aufwand umzusetzen. Zuerst werden die jeweiligen Vorverarbeitungsschritte je Subtask angewandt. Durch Hugging Face wird für alle BERT-Modelle eine Vorverarbeitungsschicht zur Verfügung gestellt. Diese tokenisiert die Texteingaben der vorverarbeiteten Tweets in Input-IDs und Attention-Masken für BERT, welche als Eingabedatensatz zusammen mit der Zielvariable gemappt werden.

4.3.2 Algorithmische Parameter

Ebenso sind wenige algorithmische Parameter festzulegen. Aufgrund der maximalen Länge eines Tweets von 103, wird eine Sequenzlänge von 128 gewählt. Durch die 5-Fold-Kreuzvalidierung sind 80% des Datensatzes Trainingsdaten. Eingaben für das Modell werden in Batches zu je 16 Sample eingeteilt. Tweets von einer Länge mit kleiner als 3 werden aussortiert. Der Adam-Optimierer wird mit einer Lernrate von konstant $4 \cdot 10^{-6}$ über 6 Epochen je Task angewendet. Kernel und Aktivität der Neuronen werden mit einem Faktor von $1 \cdot 10^{-3}$ normalisiert. Das Modell wird einmal mit und einmal ohne Klas-

sengewichte w_C angewendet. Dabei gilt, dass das Gewicht je Klasse indirekt proportional zu deren Vorkommen im Datensatz ist. Je öfter eine Klasse vorkommt, desto geringer ist ihr Gewicht für die Klassifikation. Die Berechnung erfolgt nach der Formel 31. $total$ ist die Gesamtzahl der Tweets im Datensatz und $count(C)$ die Häufigkeit der Klasse C.

$$w_C = \frac{total}{count(C) * 2} \tag{31}$$

4.4 TFIDF-SVM

Eine Support Vektor Maschine mit einer TFIDF-Vektorisierung ist eines der klassischen Modelle. SVMs werden nach wie vor im maschinellen Lernen zur Klassifikation eingesetzt und erzielen gute Ergebnisse, weshalb ein Vergleichsmodelle ein SVM sein wird. Es wird ebenfalls mit PyCharm 2021.3 in Python 3.9 implementiert und in einer 5-Fold-Kreuzvalidierung angewendet.

4.4.1 Vorverarbeitung

Die Tweets werden in Kleinschreibung überführt und per regulärem Ausdruck werden Punktationen entfernt. Mittels POS-Tagging wird Lemmatisierung durchgeführt. Schließlich erfolgt die Berechnung der Dokument-Term-Matrix mit TFIDF-Gewichtung. Für den Subtask A werden Stoppwörter entfernt, welche in B und C erhalten bleiben, da sie das angegriffene Ziel besser spezifizieren. Die weitere Vorverarbeitung der Texte wird, wie in Abschnitt 4.2.4 beschrieben, angewendet.

4.4.2 Algorithmische Parameter

Das SVM verwendet unverändert alle Standardparameter. Ebenso wie BERT wird das SVM je einmal mit Klassengewichten w_C und einmal ohne angewendet. Die Berechnung ist gleich der für BERT. Als Kernel wird der RBF-Kernel (Gauß'sche radiale Basisfunktion) verwendet, da es sich um nicht-lineare Daten handelt. Die 5-Fold-Kreuzvalidierung sorgt für einen Trainingsdatenanteil von 80%.

4.5 HBBO

HBBO ist ein sehr junger Algorithmus, der für Optimierungsprobleme entwickelt wurde. Die Adaption der Hochschule Mittweida für die Feature-Selektion für Textklassifikation aus 2022 muss für das Problem der Detektion gerichteter offensiver Sprache in der vorliegenden Arbeit Anwendung finden.

Der Algorithmus wurde durch Ludwig et al. [42] in R zur Verfügung gestellt. Die Anpassung des Codes wurde mit R 4.2.0 in RStudio 2022.02.3 durchgeführt. Die Eingabetexte werden durch eine Dokument-Term-Matrix repräsentiert, wobei die Wörter nur durch deren Bitvektor repräsentiert werden.

4.5.1 Vorverarbeitung

Die Vorverarbeitung wird nicht in R, sondern in der bereits erwähnten Python-Umgebung umgesetzt. Es wird die Punktation in den Texten entfernt und Lemmatisierung mittels POS-Tagging durchgeführt. Stoppwörter werden auch bei HBBO nur bei Subtask A entfernt und bleiben in B und C erhalten. Vorverarbeitungsschritte bzgl. der Charakteristika von Tweets werden angewandt.

4.5.2 Algorithmische Parameter

HBBO benötigt eine Reihe an Parametern, die auf das Verhalten des Algorithmus Einfluss nehmen. Wie in den anderen Verfahren haben die Holdout-Daten (Testdaten) einen Anteil von 20%. Für die Faltung der Daten in Gruppen werden 18 Dokumente pro Gruppe angenommen. Einzige Ausnahme bildet dabei die Klassifizierung der Klasse OTH im Subtask C, da sonst der Anteil der positiven Samples pro Gruppe zu gering ist. Für die Klassifikation des Labels OTH werden daher 36 Dokumente pro Gruppe angenommen. Für das Training werden pro Fold der 5-Fold-Kreuzvalidierung 1000 Iterationen durchlaufen. Der Experte testet 30 Feature pro Iteration. Sollten weniger Features vorhanden sein, als bezogen werden sollen, werden nur so viel genutzt, wie vorhanden sind. In der Education werden alle Dokumente einbezogen und es ist keine Verschlechterung des F1-Maßes erlaubt. In der Konsultation werden 10 Feature zwischen den Individuen ausgetauscht. Pro Iteration darf nur 1 Dokument je Klasse das Feld wechseln.

5 Ergebnisse und Diskussion

Im folgenden Teil werden die Ergebnisse der beschriebenen Experimente vorgestellt. Jeder Subtask wird separat betrachtet und anschließend diskutiert. Im Diskussionsteil werden aufgetretene Ereignisse beschrieben, welche durch verschiedene Vorverarbeitungsmethoden und Parameteranpassungen beobachtet werden konnten. Des Weiteren soll auch die Laufzeit der Algorithmen mit in Betracht gezogen werden.

5.1 Ergebnisse

Die folgenden Unterkapitel befassen sich mit der Darstellung der Ergebnisse. Wie beschrieben, wird jeweils der beste Fold von BERT und TFIDF-SVM dazu genutzt, die Testdaten zu klassifizieren. Die Auswahl findet anhand des Macro F1-Maßes statt, da bei allen Subtasks ein massives Ungleichgewicht der vorhandenen Klassen zu vermerken ist.

5.1.1 Subtask A

Die Support Vector Machine mit TFIDF-Gewichtung zeigt einen durchschnittlichen Macro F1-Score von 0,665 über alle Folds. Dabei weist dieser eine Standardabweichung von 0,011 auf. Für das gewichtete SVM ergeben sich nach Formel 31 die Klassengewichte $w_{OFF} \approx 1{,}5$ und $w_{NOT} \approx 0{,}75$. Der durchschnittliche Macro F1-Score liegt bei 0,719 mit einer Abweichung von 0,006.

Der Unterschied von gewichteten zu ungewichteten Modell bei BERT ist nur gering. Ohne Gewichte erreicht BERT durchschnittlich 0,775 und mit Gewichten 0,768. Die Standardabweichungen liegen jeweils bei 0,008 und 0,013. Die Klassengewichte der SVM gelten auch für BERT. Für den Verlauf des Trainings wird jeweils für die Trainings- und Testdatenevaluation die Accuracy und die Loss gespeichert. Diese sind in Abbildung 16 für den ersten Fold des gewichteten Modells dargestellt, welcher der Beste für den Subtask A ist.

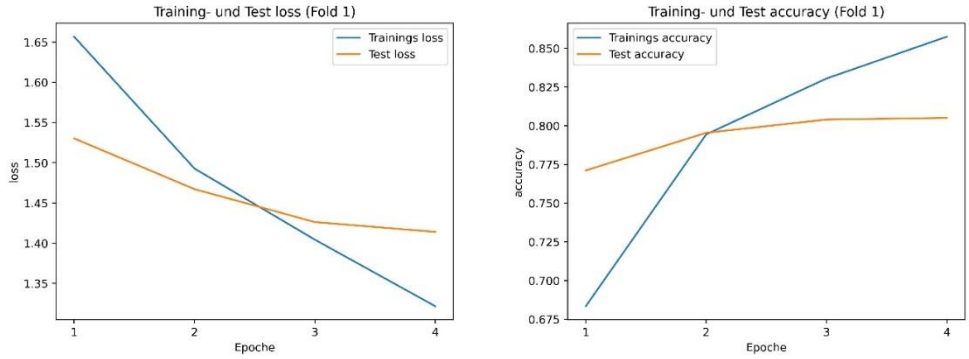

Abbildung 16: Verlauf von BERT auf Trainings- und Testdaten (Subtask A). Links: Loss, Rechts: Accuracy.

HBBO erreicht bei der Identifikation von angreifenden Tweets nur einen F1-Score von 0,495 auf den Testdaten. Während der Trainingsphase werden die F1-Maße des Experten sowie der Durchschnitt aller Individuen gespeichert. Diese sind im Verlauf über 100 Iterationen in Abbildung 17 zu sehen. Die grüne Linie zeigt den Verlauf des F1-Maßes der besten Gruppe je Iteration, während die schwarze den durchschnittlichen F1-Score aller Gruppen darstellt. Das F1-Maß ohne Training von HBBO zeigt die rote horizontale Linie.

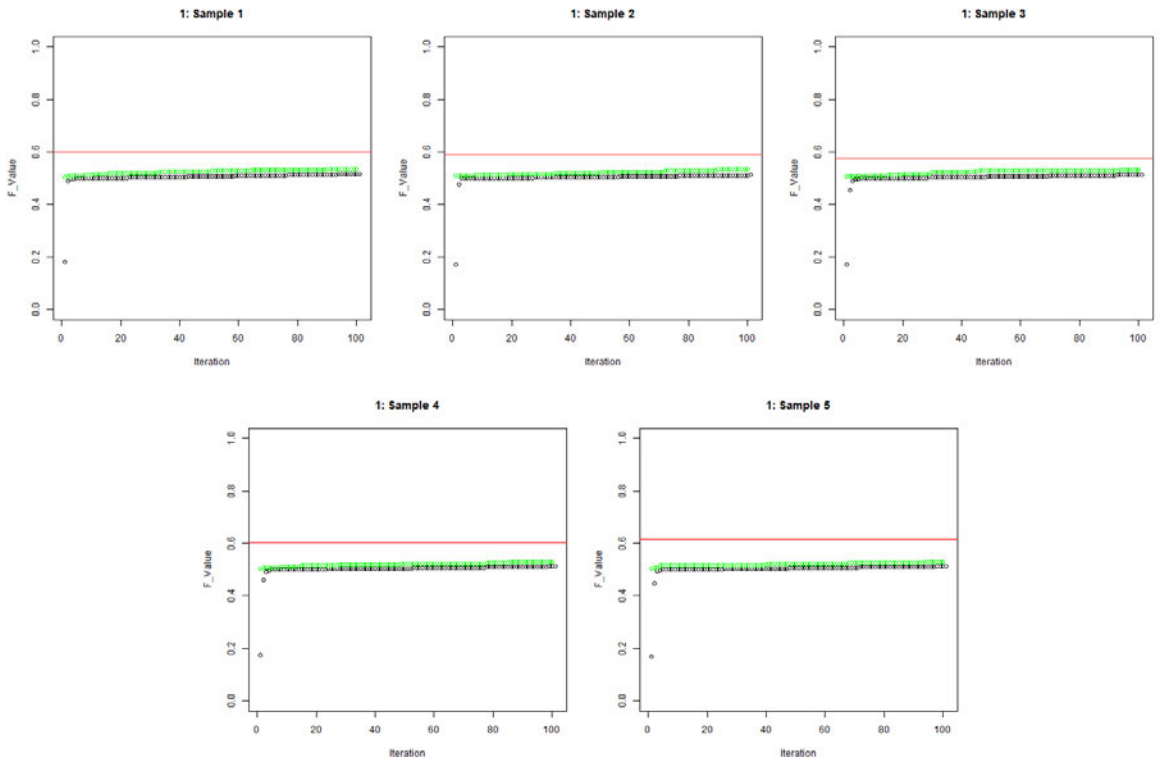

Abbildung 17: Verlauf F1-Score HBBO Training (Subtask A, Klasse OFF). grün: F1-Score der besten Gruppe; schwarz: Durchschnitt F1-Score aller Gruppen; rot: F1-Score ohne Training.

Tabelle 8 zeigt die Ergebnisse aller Modelle tabellarisch für Subtask A. Zum visuellen Vergleich folgt in Abbildung 18 ein ROC-Chart für alle Modelle auf den Testdaten.

Tabelle 8: Evaluationstabelle für Subtask A aller Modelle. Gezeigt werden Precision (P), Recall (R) und F1-Score je Klasse und der gewichtete Durchschnitt. Bestes F1 Macro in fett gedruckt.

	OFF			NOT			Weighted Average			
Modell	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1 Macro**
SVM	0.85	0.36	0.50	0.75	0.97	0.84	0.78	0.76	0.73	0.67
BERT	0.72	0.69	0.70	0.85	0.87	0.86	0.81	0.81	0.81	0.78
HBBO	0.42	0.13	0.20	0.69	0.91	0.79	0.60	0.66	0.60	0.49
SVM gew.	0.71	0.54	0.61	0.80	0.90	0.85	0.78	0.78	0.77	0.73
BERT gew.	0.69	0.75	0.72	0.87	0.83	0.85	0.81	0.81	0.81	**0.78**

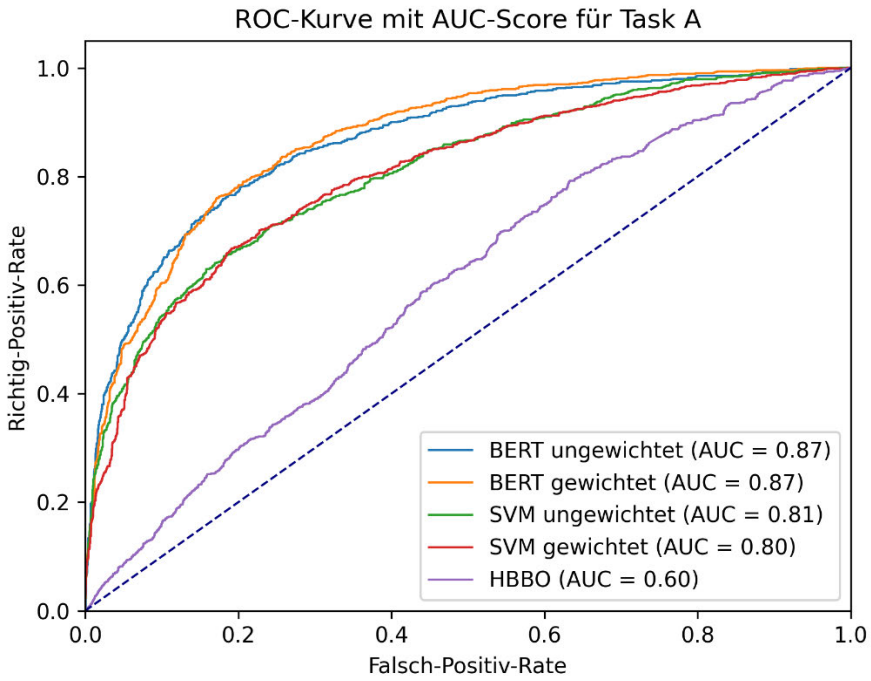

Abbildung 18: ROC-Kurve für Subtask A aller Modelle (Klasse OFF). AUC-Werte unten rechts.

5.1.2 Subtask B

Für die Identifikation von gerichteten Angriffen erreicht die SVM im besten Fold ein Macro F1 von 0,67. Durchschnittlich liegt dieser bei 0,483 mit einer Abweichung von 0,006. Für das gewichtete Modell wurden die Klassengewichte $w_{TIN} \approx 0{,}57$ und $w_{UNT} \approx 4{,}3$. Es zeigt einen durchschnittlichen F1-Score von 0,544 mit einer Abweichung von 0,034.

BERT zeigt jetzt einen deutlichen Unterschied im Training mit und ohne Gewichte. Ohne Gewichte erreicht BERT durchschnittlich 0,469 im F1-Score mit einer Standardabweichung von 0,002. Das gewichtete Modell erreicht durchschnittlich 0,583 mit 0,03 Abweichung. Der Verlauf der Accuracy und der Loss des besten Folds von BERT in Subtask B zeigt Abbildung 19.

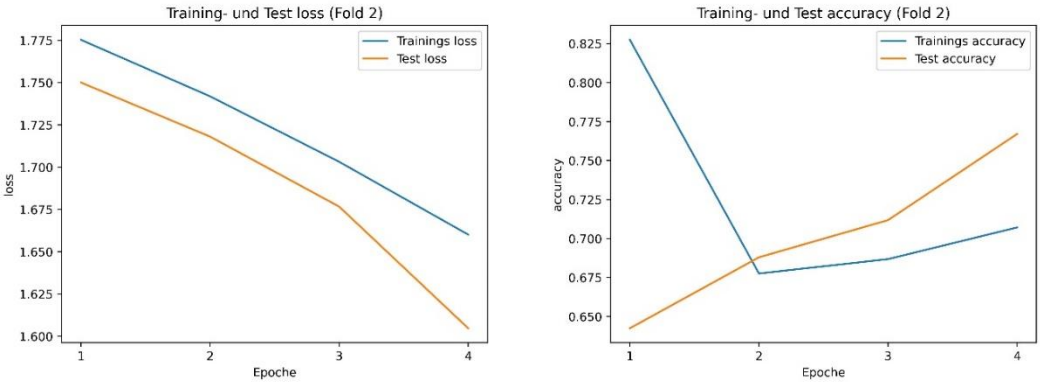

Abbildung 19: Verlauf von BERT auf Trainings- und Testdaten (Subtask B). Links: Kostenfunktion, Rechts: Accuracy.

HBBO zeigt bei der Identifikation gerichteter Angriffe einen F1-Score von 0,495 und ist somit deutlich näher an denen der anderen implementierten Modelle. Abbildung 20 zeigt den Verlauf der F1-Scores von HBBO für die Klasse TIN des Subtasks B.

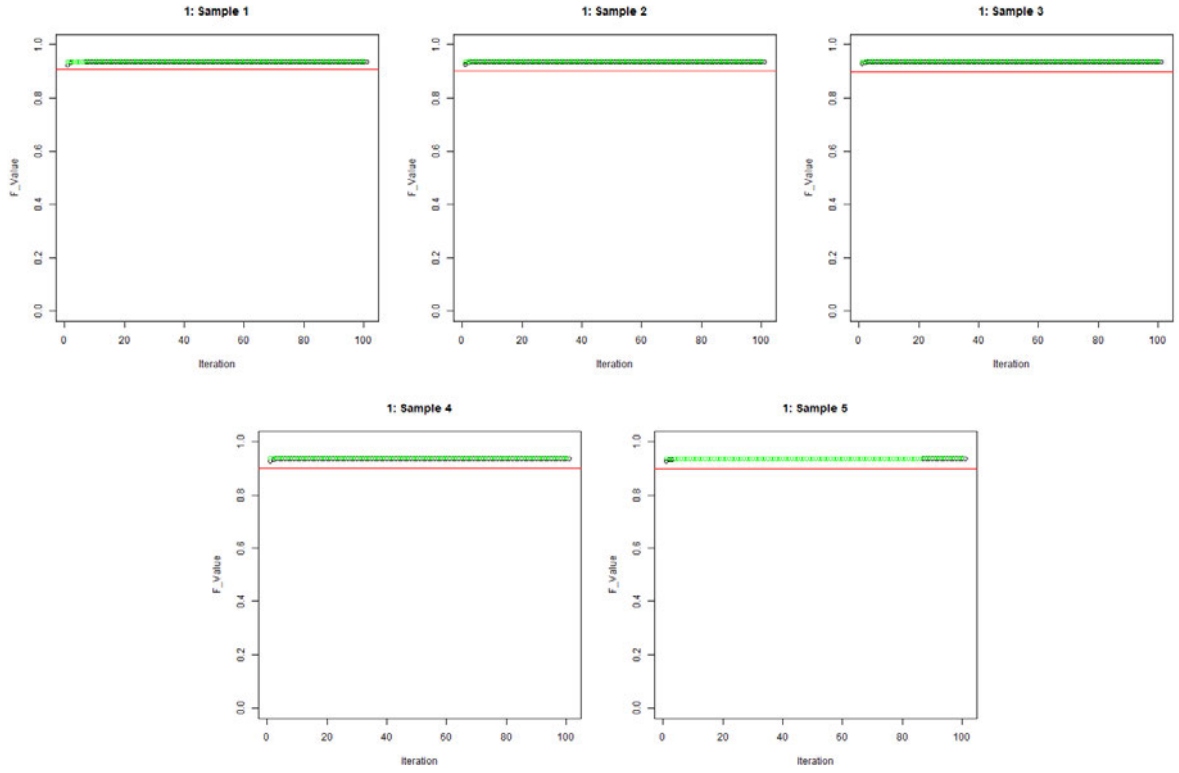

Abbildung 20: Verlauf F1-Score HBBO Training (Subtask B, Klasse TIN). grün: F1-Score der besten Gruppe; schwarz: Durchschnitt F1-Score aller Gruppen; rot: F1-Score ohne Training.

Tabelle 9 zeigt die Evaluationsergebnisse der besten Folds aller Modelle für Subtask B. Abbildung 21 zeigt die ROC-Charts und AUC-Werte aller Modelle im Vergleich.

Tabelle 9: Evaluationstabelle für Subtask B aller Modelle. Gezeigt werden Precision (P), Recall (R) und F1-Score je Klasse und der gewichtete Durchschnitt. Bestes F1 Macro in fett gedruckt.

	TIN			UNT			Weighted Average			
Modell	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1 Macro**
SVM	0.87	1.00	0.93	0.60	0.03	0.05	0.84	0.87	0.82	0.49
BERT	0.89	1.00	0.94	0.00	0.00	0.00	0.79	0.89	0.84	0.47
HBBO	0.90	0.98	0.94	0.16	0.03	0.05	0.82	0.88	0.84	0.49
SVM gew.	0.91	0.98	0.95	0.53	0.17	0.26	0.88	0.90	0.88	0.60
BERT gew.	0.93	0.80	0.86	0.28	0.57	0.38	0.85	0.77	0.80	**0.62**

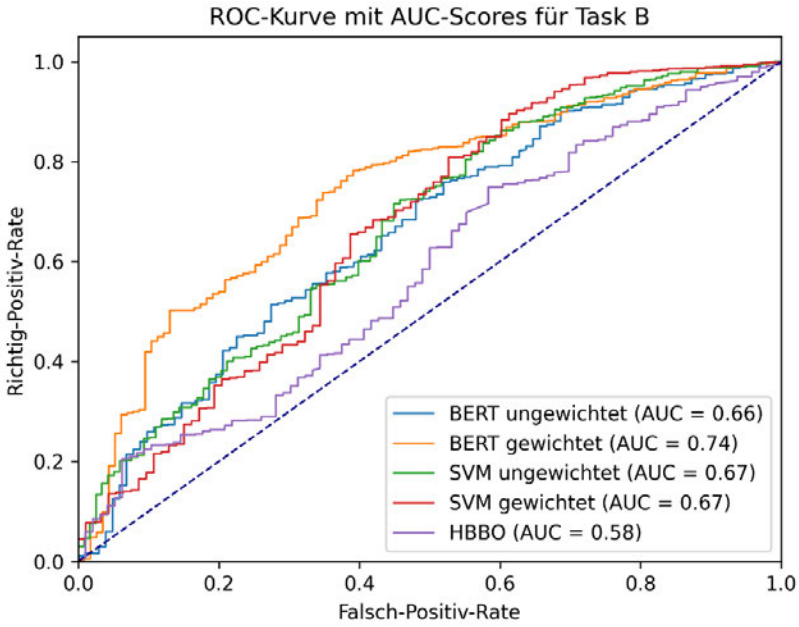

Abbildung 21: ROC-Kurve für Subtask B aller Modelle (Klasse TIN). AUC-Werte unten rechts.

5.1.3 Subtask C

Die SVM erreicht bei der Identifikation des Ziels in Subtask C einen durchschnittlichen F1-Score von 0,439 mit einer Abweichung von 0,018. Die Gewichte für das gewichtete Modell wurden zu folgenden Werten berechnet: $w_{IND} \approx 0{,}82$; $w_{GRP} \approx 1{,}77$ und $w_{OTH} \approx 4{,}75$. Mit den Gewichten zeigt das SVM einen F1-Score von durchschnittlich 0,496 mit einer Standardabweichung von 0,023.

BERT erreicht ohne Gewichte leicht bessere Ergebnisse mit durchschnittlich 0,485 im F1-Maß und 0,008 als Standardabweichung. Die Gewichte sind gleich der des SVMs. Mit diesen erreicht BERT einen höheren F1-Score von durchschnittlich 0,546. Der Wert hat eine Abweichung von 0,012. Accuracy und Loss für den Trainingsverlauf sind in Abbildung 22 dargestellt.

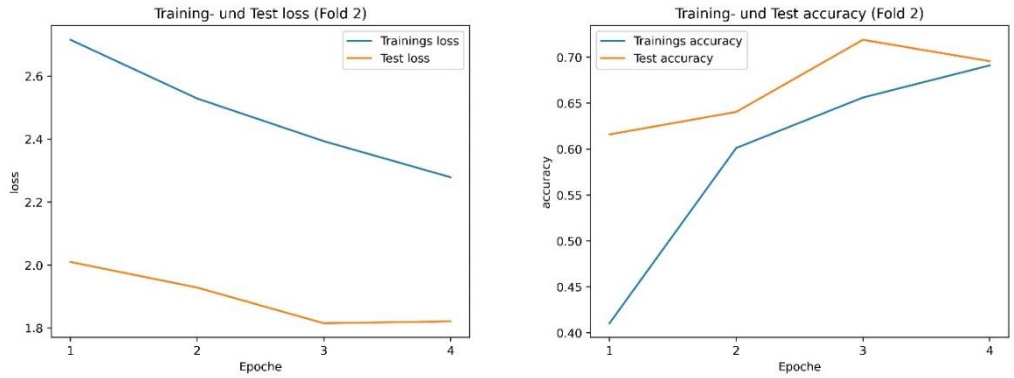

Abbildung 22: Verlauf von BERT auf Trainings- und Testdaten (Subtask C). Links: Kostenfunktion, Rechts: Accuracy.

HBBO muss aufgrund des aktuellen Codedesigns für jede Klasse separat ausgeführt werden. Es trennt jeweils die aktuell betrachtete, positive Klasse von den übrigen ab und benötigt damit 3 Durchführungen. Durch das Zusammenfügen der Ergebnisse ergibt sich insgesamt ein F1-Maß von 0,426 auf den Testdaten. Der Verlauf des F1-Maßes auf den Trainingsdaten wird in den Abbildungen 23 - 25 dargestellt.

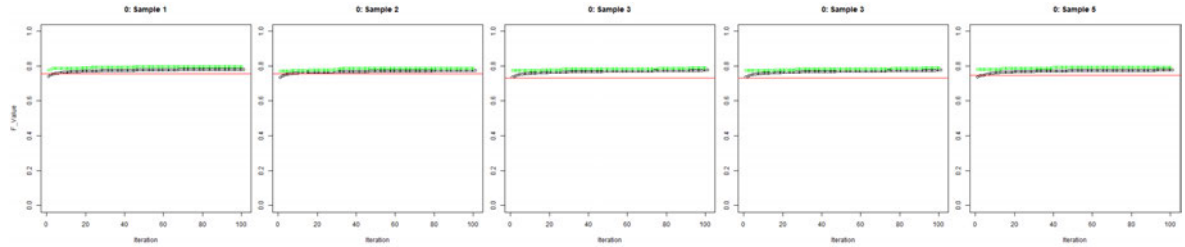

Abbildung 23: Verlauf F1-Score HBBO Training (Subtask C, Klasse IND). grün: F1-Score der besten Gruppe; schwarz: Durchschnitt F1-Score aller Gruppen; rot: F1-Score ohne Training.

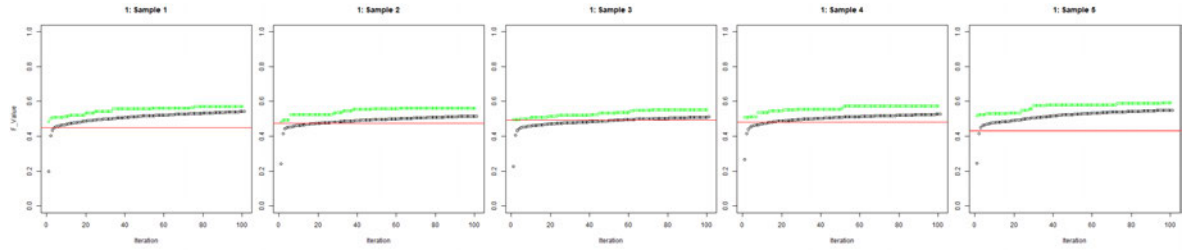

Abbildung 24: Verlauf F1-Score HBBO Training (Subtask C, Klasse GRP). grün: F1-Score der besten Gruppe; schwarz: Durchschnitt F1-Score aller Gruppen; rot: F1-Score ohne Training.

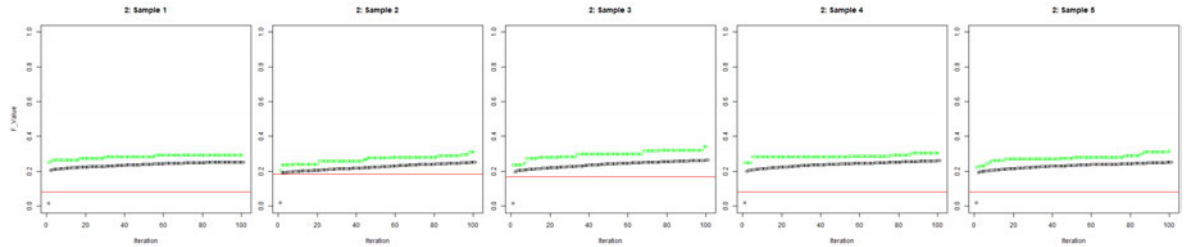

Abbildung 25: Verlauf F1-Score HBBO Training (Subtask C, Klasse OTH). grün: F1-Score der besten Gruppe; schwarz: Durchschnitt F1-Score aller Gruppen; rot: F1-Score ohne Training.

Die Ergebnisse der jeweils besten Folds der Modelle für Subtask C werden in Tabelle 10 dargestellt. Die ROC-Kurven und AUC-Werte sind in Abbildung 26 festgehalten.

Tabelle 10: Evaluationstabelle für Subtask C aller Modelle. Gezeigt werden Precision (P), Recall (R) und F1-Score je Klasse und den gewichteten Durchschnitt. Bestes F1 Macro in fett gedruckt.

	IND			GRP			OTH			Weighted Average			
Modell	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1 Macro**
SVM	0.72	0.93	0.81	0.69	0.48	0.56	0.00	0.00	0.00	0.72	0.84	0.67	0.46
BERT	0.76	0.91	0.83	0.69	0.61	0.65	0.00	0.00	0.00	0.74	0.83	0.70	0.49
HBBO	0.70	0.86	0.78	0.67	0.20	0.31	0.17	0.21	0.19	0.65	0.68	0.64	0.43
SVM gew.	0.80	0.86	0.83	0.62	0.70	0.66	0.44	0.05	0.10	0.74	0.80	0.71	0.53
BERT gew.	0.83	0.80	0.81	0.61	0.67	0.64	0.24	0.24	0.24	0.70	0.69	0.70	**0.57**

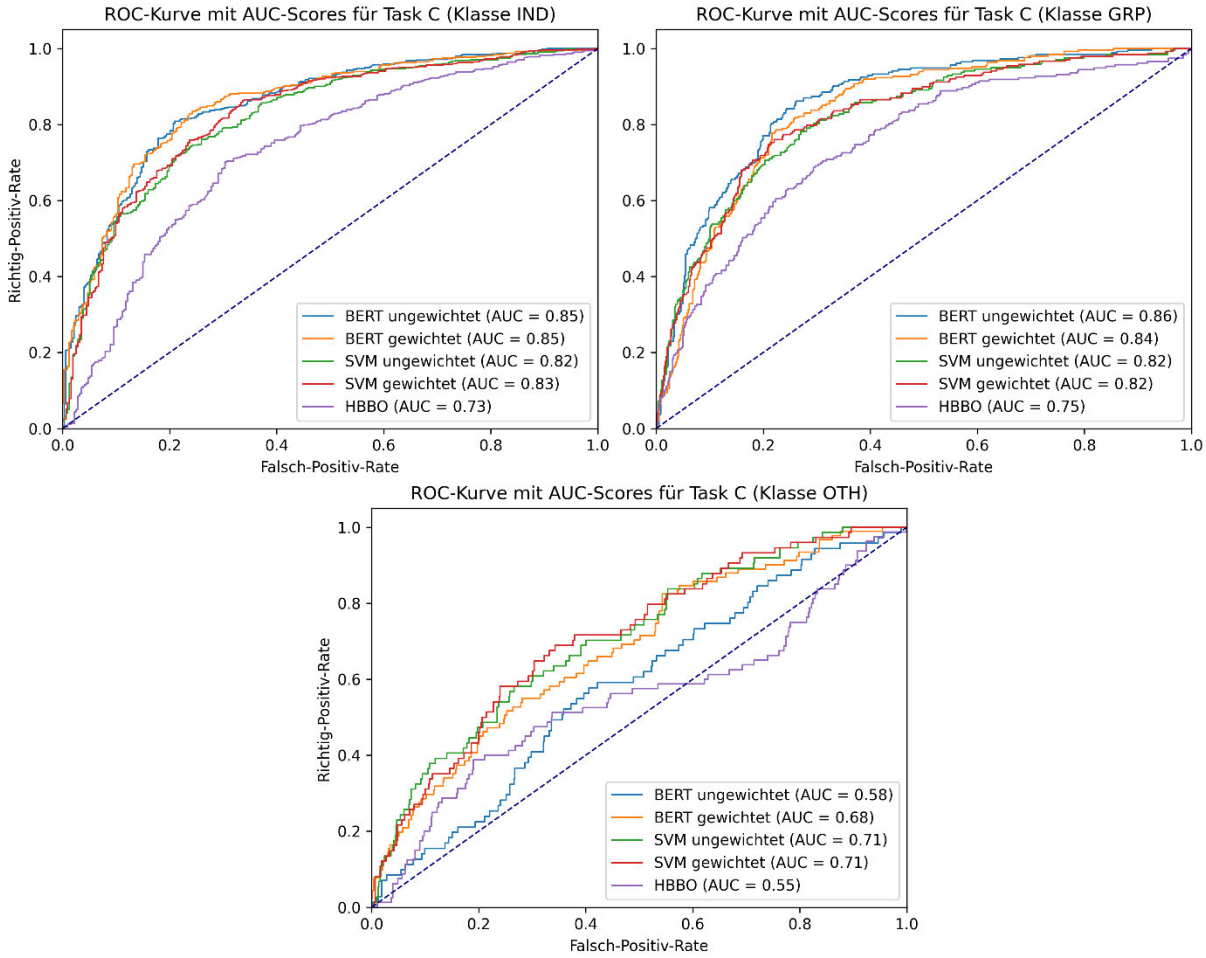

Abbildung 26: ROC-Kurve für Subtask C aller Modelle für alle Klassen. Oben links: Klasse IND, oben rechts: Klasse GRP, unten: Klasse OTH. AUC-Werte im jeweiligen Diagramm unten rechts.

5.2 Diskussion

Im vorherigen Unterabschnitt wurden die Ergebnisse der implementierten Modelle vorgestellt und visualisiert. In den folgenden Unterkapiteln wird der Vergleich der Modelle vorgenommen mit Betracht auf die gezeigten Ergebnisse. Dabei soll ebenso auf deren Laufzeit und Performanz eingegangen werden. Des Weiteren werden Beobachtungen während der Vortests zur Kombination der Vorverarbeitungsschritte aufgezeigt.

5.2.1 Subtask A

Subtask A, die Identifikation von angreifenden Nachrichten, ist der einfachste Teil des Problems gerichteter offensiver Sprache. Zampieri et al. [11], [12] als auch [16] zeigten, dass diese Klassifikationsaufgabe von den getesteten Modellen gut zu bewältigen ist.

In den durchgeführten Experimenten kann sich BERT gegen die anderen Modelle durchsetzen. Unabhängig vom Verwenden der Klassengewichte erzielt BERT bessere Evaluationsmaße als das SVM. Ebenso ist festzustellen, dass das Modell mit und ohne Gewichte ähnliche Ergebnisse liefert. Tendenziell verursachen die Gewichte instabilere und schlechtere Vorhersagen, was anhand des Durchschnittes des F1-Maßes und dessen Standardabweichung zu erkennen ist. Ebenso ähneln sich die ROC-Kurven und AUC-Werte der Klassifikatoren.

In Abbildung 16 kann festgestellt werden, dass die Kostenfunktion der Trainingsdaten geringer wird, als die der Testdaten. Umgekehrt gilt das auch für die Accuracy, welche auf den Trainingsdaten höher ist, als auf den Testdaten. Diese Beobachtung zeigt eine Überanpassung des Modells an die Trainingsdaten an, was bei BERT ein bekanntes Problem ist [53]. Sun et al. empfehlen eine individuelle Lernrate für jede Schicht des BERT-Modells. Alternativ ist es möglich, die Lernrate in jeder Epoche zu verringern oder weniger Epochen zum Training zu durchlaufen. Anhand des Diagramms sind 2 oder 3 Epochen zu verwenden. Die Überanpassung kann durch die größere Menge an Daten in Subtask A begründet sein.

Das SVM wird durch das Anwenden der Gewichte allerdings maßgeblich beeinflusst. Durch deren Verwendung kann der F1-Score um 0,06 gesteigert werden. Das entsteht durch eine bessere Erkennung der minder vertretenen Klasse OFF. Die ROC-Kurven der SVMs zeigen keinen beachtlichen Unterschied, im Gegensatz zu den F1-Maßen.

Signifikant schlechter performt HBBO mit dem niedrigsten F1-Maß von 0,49. Das ist ebenso im ROC-Chart zu sehen, da dessen Kurve deutlich flacher und näher an der Diagonalen ist. Der AUC Klassifikator ist nur 0,1 besser als der einer zufälligen Vergabe der Klassenlabels. Anhand von Abbildung 17 ist zu erkennen, dass während des Trainings kaum ein Lerneffekt stattgefunden hat. Letztlich hat die Klassifikation mit dem gelernten Featureset schlechter performt, als die ohne vorheriges Training.

HBBO nimmt auf dem in Abschnitt 4 beschriebenen System für Subtask A für 100 Iterationen eine Laufzeit von ca. 46h in Anspruch und ist damit am zeitintensivsten. Das SVM benötigt etwa 1:20 Minuten in Addition mit 2 Minuten Vorverarbeitungszeit und BERT über 4 Epochen ca. 14:20 Minuten Laufzeit.

Eine weitere interessante Beobachtung ist, dass die Vorverarbeitungsschritte bezüglich der Charakteristika von Tweets im SVM deutlich weniger Einfluss haben als etwa bei BERT. Während das gewichtete SVM im F1-Maß um nur 0,01 sinkt, wird BERT um ganze 0,19 schlechter, wenn keine der Schritte angewendet werden.

Für Subtask A performt insgesamt BERT am besten. Dabei ist es irrelevant, ob Klassengewichte angewendet werden. Aufgrund der höheren Standardabweichung mit Gewichten, sollte auf die Gewichte verzichtet werden, um stabilere Ergebnisse zu erhalten. Mit weiteren Anpassungen der Anzahl der Epochen und dem setzen von absteigenden Lernraten, kann für BERT noch eine Verbesserung stattfinden.

5.2.2 Subtask B

Subtask B beschäftigt sich mit der Unterscheidung von gerichteten und ungerichteten Angriffen. Aufgrund des immensen Ungleichgewichts der Klassen ist diese Aufgabe schwieriger als Subtask A.

Bei den Modellen ohne Gewichten ist es überraschend im Vergleich zu A, dass HBBO bessere F1-Maße erbringt als die SVM und BERT. Die Hauptursache dafür scheint das Ungleichgewicht zu sein, denn BERT ist es scheinbar schlecht möglich für die minder vertretene Klasse UNT effektive Feature zu finden und diese zu klassifizieren. Das Erkennen der häufigeren Klasse scheint hingegen für alle Modelle kein Problem darzustellen. Mit Gewichten stellt allerdings, wie bei Subtask A auch BERT das beste Modell dar. Besonders zu bemerken ist, dass der Recall der Klasse UNT mit 0,57 deutlich höher ist als der, anderer Modelle.

In Abbildung 19 ist zu erkennen, dass das Problem der Überanpassung bei BERT in Subtask B nicht stattgefunden hat. Das liegt möglicherweise daran, dass deutlich weniger Daten zur Verfügung stehen, bzw. die Feature komplexer sind. Die anfängliche Überschneidung der Accuracy von Test und Training können durch das Ungleichgewicht der Klassen zustande kommen. Im weiteren Verlauf zeigt sie aber das erwartete Verhalten. In diesen Fall kann versucht werden das Training um eine Epoche zu verlängern, da es auch in den anderen Folds zu keiner Überanpassung gekommen ist. Ggf. kann der Trainingseffekt mit einer im Verlauf sinkenden Lernrate noch erhöht werden.

Das SVM zeigt ebenso wie BERT eine deutliche Verbesserung mit Gewichten. Zwar ähneln sich die F1-Maße der beiden Modelle, allerdings sind die AUC-Werte in Abbildung 21 unterschiedlich. Das liegt am deutlich höheren Recall für UNT, den BERT durch die Gewichte erreicht. Das SVM hingegen hat nur einen Recall von 0,17 für UNT und somit eine hohe falsch-positiv-Rate der Klasse TIN.

Trotz des ausgeglichenen Macro F1-Wertes von HBBO ist dessen AUC-Wert niedrig, was durch den gleichen Grund, wie bei der SVM entsteht. Dennoch erkennt HBBO die Klasse UNT besser als BERT. Im Verlaufsplot (Abbildung 20) ist ein Problem zu sehen, welches durch Ludwig et al. [42]beschrieben wurde. Schon nach ca. 10 Iterationen setzt der Lerneffekt aus und die F1-Maße verbessern sich nicht mehr maßgeblich. Dadurch entsteht

ein Plateau im Graphen und das Training könnte an der Stelle abgebrochen werden. Allerdings ist die Klassifikation mit dem gelernten optimalen Featureset besser als ohne dieses.

HBBO benötigt für die gesetzten 100 Iterationen ca. 4,75 h und ist auch in Subtask B am zeitintensivsten. Das SVM benötigt im Durchschnitt 10 s mit 40 s Vorverarbeitungszeit. BERT rechnet über 4 Epochen 4,5 min.

Eine Beobachtung, die in den Vortest gut zu erkennen war, ist die Veränderung der Evaluationsmaße bei Entfernen der Stoppwörter. Dieses Vorgehen ist für viele NLP-Tasks typisch, verschlechtert allerdings die Performanz in den Subtasks B und C. Das gewichtete SVM erreicht mit Stoppwörtern ein F1-Maß von 0,60. Werden diese entfernt erzielt es nur noch 0,53 und ist somit um 0,07 schlechter. Dieser Effekt lässt sich auch bei HBBO beobachten, allerdings in geringerem Maße.

Die Performanz wird bei allen Modellen durch das Vorverarbeiten der Tweets verbessert, aber mit weniger Einfluss als erwartet. Die größten Verbesserungen werden dadurch bei BERT erzielt. Das SVM hat kaum Nutzen davon und HBBO verbessert sich geringfügig. Dieser Effekt ist durch das Vortraining von BERT zu erklären. Es kennt ein Vokabular von ca. 3,3 Mrd. standardisierten Wörtern, wobei keine Emojis und Hashtags vorkommen. Ohne deren Vorverarbeitung muss ein Neuerlernen dieser Token stattfinden, was den Lernprozess, der erzielt werden soll, hemmt. Durch das Umwandeln der Token in Wörter, die BERT kennt, wird dieser Prozess beschleunigt und führt schließlich zu einer besseren Klassifikation.

Anhand des F1-Macros ist BERT gewichtet als bestes Modell zu nennen. Allerdings richtet sich der Task nach der Identifikation gerichteter offensiver Sprache. Daher ist in dem Fall das gewichtete SVM vorzuziehen, da dieses für TIN sehr hohe Precision und Recall hat. Auch HBBO ist nach diesen Kriterien als gut einzuschätzen, allerdings benötigt es ein Vielfaches der Laufzeit des SVMs und hat nur ein F1 Macro von 0,49.

5.2.3 Subtask C

Die finale Identifikation der angegriffenen Ziele ist der schwerste Teil der Erkennung gerichteter angreifender Sprache. Ebenfalls herrscht in den Daten ein massives Ungleichgewicht, was in den Evaluationsergebnissen klar zu erkennen ist.

BERT als auch das SVM ohne Gewichte haben Probleme, die kleinste Klasse OTH zu klassifizieren. Überraschenderweise ist HBBO das einzige Modell ohne Klassengewichte, welches diese erkennt. Trotz dessen schneidet es mit dem geringsten F1-Maß ab. Grund dafür sind die niedrigeren F1-Scores in den anderen Klassen. BERT und SVM erhalten durch das Verwenden der Gewichte eine Performancesteigerung. Durch die Gewichte wird es überhaupt möglich die Klasse OTH zu erkennen. Entgegen der Erwartung, dass durch die Klassengewichte die Klassen IND und GRP schlechtere F1-Maße erreichen, wird die Performanz des SVMs in allen Klassen verbessert.

BERT ist mit als auch ohne Gewichten jeweils das beste Modell. Zu erkennen ist, dass das Anwenden der Gewichte, bessere Ergebnisse liefert, die keine auffällige Instabilität

aufweisen. Die Verlaufsplots zeigen, dass BERT mit 4 Epochen und der Lernrate gut trainiert wird und nicht zur Überanpassung neigt.

HBBO zeigt trotz des niedrigen F1-Maßes das von ihm erwartete Verhalten. Gut in den Abbildungen 23 – 25 zu erkennen ist, dass der Lerneffekt für alle Klassen einsetzt und über 100 Iterationen erhalten bleibt. Zwar für IND nur gering, aber dabei sei anzumerken, dass der anfängliche F1-Score schon sehr hoch ist. Trotz dessen kann der Algorithmus die Scores ohne Training nicht oder nur leicht anheben.

Die ROC-Charts zeigen, dass die Klassen IND und GRP mäßig stabil klassifiziert werden, da sie einen annähernd gleichmäßig bogenartigen Verlauf zeigen. HBBO zeigt große Abweichungen und verläuft geradliniger. BERT ist auch Klassenweise im ROC für IND und GRP der beste Klassifikator. Die Klasse OTH ist auch anhand der ROC-Charts als schwierig erkennbar einzuschätzen. Alle Modelle bis auf die SVM und BERT gewichtet zeigen nahezu geradlinige Verläufe. Anhand dieses Verlaufes ist zu erkennen, dass die Scores, anhand derer der ROC-Chart erstellt wird, sehr unsicher sind und scheinbar zufällig vergeben wurden. BERT hat nicht einmal die Klasse OTH vorhergesagt. Für BERT ist es demnach zwingend erforderlich, Gewichte bei großen Differenzen in der Klassenverteilung zu vergeben.

Das SVM benötigt ca. 12 Sekunden für Training und Evaluation der Testdaten und BERT etwa 4 Minuten. Aufgrund dass HBBO für jede Klasse eine Binärentscheidung durchgeführt werden muss, wird die Laufzeit um ein dreifaches erhöht. Ein Durchlauf benötigt etwa 2,5h und schließlich 7,5h für Training und Evaluation in allen Klassen.

Auch in Subtask C zeigt sich, dass die Vorverarbeitung der Tweets eine positive Auswirkung auf die Ergebnisse haben. Zwar werden weniger Schritte angewendet als in A und B, aber die Performanz wird durch das Verarbeiten von Emojis und Hashtags bereits verbessert. Ebenso ist der gleiche Effekt, wie in Subtask B bezüglich des Entfernens von Stoppwörtern zu vermerken. Das Erhalten dieser verbessert den F1-Score des gewichteten SVMs um durchschnittlich 0,03.

In Subtask C dominiert BERT. Zwar kann es ohne Gewichte, die Klasse OTH nicht erkennen, aber zeigt mit Klassengewichten weitaus bessere Ergebnisse als die anderen Modelle. Ebenso hat es eine moderate Laufzeit und bringt sichere, stabile Ergebnisse.

6 Zusammenfassung und Fazit

Das Gebiet der angreifenden Sprache ist im Bereich des maschinellen Lernens eine weit diskutierte und untersuchte Disziplin, wie eingangs gezeigt wurde. Die vorliegende Arbeit zielt darauf ab, bisher verwendete als auch neue Methoden vorzustellen und zu vergleichen. Angreifende Sprache kann von mathematisch einfachen klassischen Modellen bis hin zu rechenaufwändigen Optimierungsalgorithmen bearbeitet werden. Dabei gibt es grundlegende Unterschiede in der Verarbeitung der Daten als auch bei den Ergebnissen, die erzielt werden. Festzustellen ist, dass es besonders auf die Feature ankommt die zur Klassifikation herangezogen werden. Beispielhaft ist hier der Vergleich zweier Support Vector Machines, die einerseits Feature durch deren Termfrequenz erhalten und andererseits durch Schwarmintelligenz iterativ bestimmte optimale Feature nutzen.

Mit Blick auf die Ergebnisse kann teilweise zwischen dem klassischen SVM und dem neuronalen Ansatz BERT kaum ein Unterschied erkannt werden. Eine Schwierigkeit, die für alle Modelle herausfordernd scheint, ist das Klassenungleichgewicht der Daten, welches in allen Subtasks herrscht. Mit diesem Problem kommen BERT und SVM ohne entsprechendes Gegenwirken mittels Klassengewichten nur schlecht zurecht. HBBO hingegen überzeugt bei größeren Ungleichgewichten wie in Subtask B und C. Beide Male ist HBBO in der Lage die jeweils kleinste Klasse zu identifizieren. Das kommt dadurch, dass eben genau für die Klasse möglichst optimale Feature gefunden werden, welche deskriptiv für diese sind.

BERT benötigt im Gegensatz zu HBBO nur einen Bruchteil dessen Laufzeit, erreicht aber im Task der gerichteten offensive Sprache State-of-the-Art Ergebnisse in allen drei Subtasks. Bemerkenswert ist, dass trotz eines leichten Ungleichgewichts in A hervorragende Ergebnisse erzielt werden und somit die Notwendigkeit der Klassenwichtung entfällt. Diese ist in Subtasks B und C von Nöten, um alle Klassen erfolgreich zu klassifizieren. BERT neigt bei vielen Trainingsdaten schnell zur Überanpassung. Besonders bei Subtask A sollte die Anzahl der Epochen oder die Lernrate verringert werden, um dem entgegen zu steuern.

In den Verlaufsplots von HBBO ist zu sehen, dass der erwartete Lerneffekt einsetzt mit Ausnahme von Subtask B. Trotz dessen zeigt der Algorithmus stets schlechtere F1-Maße als das reguläre SVM mit TFIDF-Vektorisierung. Das kann an der Vektorisierung der Texte liegen. Da HBBO diese in Binärvektoren überführt, fallen die Gewichtungen durch Termfrequenzen weg. Diese sollten in zukünftigen Versionen des Codes eingebaut werden, um feingranularere Feature zu erhalten. Außerdem sollte der Code so angepasst werden, dass jedes Dokument einzeln optimiert wird, anstatt in Gruppen. Trotz teils guter Ergebnisse leidet der Algorithmus an der Dauer der Durchführung. Die anderen getesteten Modelle liefern vergleichbare Ergebnisse in deutlich kürzerer Zeit. Als junger neuartiger Algorithmus ist HBBO dennoch dazu geeignet Feature aus Text zu extrahieren und zur Klassifikation heranzuziehen. Leider ist es im aktuellen Setup nicht in der Lage mit aktuellen State-of-the-Art Ansätzen mitzuhalten.

Allgemein haben die Vorverarbeitungsschritte bezüglich der Charakteristika von Tweets in allen Modellen erfolgreich bessere Ergebnisse gebracht. Besonders das Vorverarbeiten der Hashtags ist effektiv für die Klassifizierungen. Das Entfernen von URLs wurde dauerhaft angewendet und führte stets zu gleichen oder besseren Ergebnissen. Folglich ist das URL-Token nicht sinnvoll einsetzbar für die Klassifizierung der Modelle. Es hat sich gezeigt, dass das Entfernen von Stoppwörtern in Subtask B und C keine Verbesserung bringt, wie in anderen NLP-Task. Wörter, die sich auf Personen beziehen, enthalten wichtige Informationen über die Richtung und das Ziel eines Angriffes.

Für alle drei getesteten Modelle haben sich Stärken im Problem gerichteter offensiver Sprache auf Twitter ergeben. Wie in vielen anderen Arbeiten stellt sich BERT als performanter Ansatz dar, wie viele andere Vertreter neuronaler Modelle. Es ist rein durch das Verständnis des Kontextes einer Nachricht anderen Vorzuziehen. Dennoch ist es für einfache Aufgaben nicht immer notwendig aufwändige neuronale Modelle zu trainieren, wie in den Ergebnissen zu sehen ist. In nur wenigen Sekunden ist ein einfaches SVM in der Lage, gleichwertige Vorhersagen zu treffen. Aufgrund dessen haben klassische Methoden in modernen Klassifikationsproblemen eine Daseinsberechtigung und sind geeignet für das testen neuer Feature. So auch im Verfahren von HBBO, welches ein junger Ansatz ist, welcher mit einigen Verbesserungen einen erfolgreicher Feature-Selektionsalgorithmus zur Verfügung stellt.

Ein Problem, welches noch einmal Erwähnung finden soll, ist die Konzentration vieler wissenschaftlichen Arbeiten auf Twitter Nachrichten. Offensive, verachtende, sexistische und hasserfüllte Sprache ist auf allen Onlineplattformen präsent, wie anhand verschiedener Studien zu erkennen ist [54]–[56]. Es ist nicht nur nötig angreifende Sprache zu erkennen, sondern deren Eigenschaften für die jeweilige Plattform herauszuarbeiten. Das ist ein erster Schritt für eine noch bessere Featureselektion zu Merkmalen angreifender Sprache und deren Zielen.

Weiterhin ist es nötig in zukünftigen Arbeiten mehr und bessere Daten einzusetzen. Ein erster Schritt dazu wurde bereits durch Rosenthal et al. umgesetzt mit der Erweiterung SOLID [20]. Eine weitere Verbesserung dazu ist das Erhalten der Nutzer, die Kommentare verfassen. Dadurch ist es möglich bestimmte Nutzer zu fokussieren und deren Grundverhalten zu definieren, welches in die Klassifikation einfließen kann. Trotz datenschutzrechtlicher Gegebenheiten, ist dieses Feature mit entsprechender Pseudo- bzw. Anonymisierung umsetzbar.

Literatur

[1] H. Watanabe, M. Bouazizi und T. Ohtsuki, „Hate Speech on Twitter: A Pragmatic Approach to Collect Hateful and Offensive Expressions and Perform Hate Speech Detection“, *IEEE Access*, Bd. 6, S. 13825–13835, 2018, doi: 10.1109/ACCESS.2018.2806394.

[2] E. L. Mirsky und H. A. Omar, „Cyberbullying in Adolescents: The Prevalence of Mental Disorders and Suicidal Behavior“, *International Journal of Child and Adolescent Health*, Bd. 8, S. 37–39, 2015.

[3] A. H. Razavi, D. Inkpen, S. Uritsky und S. Matwin, „Offensive Language Detection Using Multi-level Classification“, in *Advances in Artificial Intelligence*, Ottawa, Canada: Springer, 2010, S. 16–27. doi: 10.1007/978-3-642-13059-5_5.

[4] W. Warner und J. Hirschberg, „Detecting Hate Speech on the World Wide Web“, Montreal, Canada, 2012. [Online]. Verfügbar unter: http://info.yahoo.com/legal/us/yahoo/utos/utos-173.html, Zugegriffen am: 01.09.2022.

[5] W. Wang, L. Chen, K. Thirunarayan und A. P. Sheth, „Cursing in english on twitter“, in *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW*, 2014, S. 415–425. doi: 10.1145/2531602.2531734.

[6] C. van Hee, B. Verhoeven, E. Lefever, G. de Pauw, W. Daelemans und V. Hoste, „Guidelines for the Fine-Grained Analysis of Cyberbullying“, 2015.

[7] C. Nobata, J. Tetreault, A. Thomas, Y. Mehdad und Y. Chang, „Abusive language detection in online user content“, in *25th International World Wide Web Conference, WWW 2016*, 2016, S. 145–153. doi: 10.1145/2872427.2883062.

[8] Z. Waseem und D. Hovy, „Hateful Symbols or Hateful People? Predictive Features for Hate Speech Detection on Twitter“, in *Proceedings of the NAACL Student Research Workshop*, Juni 2016, S. 88–93. doi: 10.18653/v1/N16-2013.

[9] T. Davidson, D. Warmsley, M. Macy und I. Weber, „Automated Hate Speech Detection and the Problem of Offensive Language“, März 2017, [Online]. Verfügbar unter: http://arxiv.org/abs/1703.04009, Zugegriffen am: 01.09.2022.

[10] Z. Waseem, T. Davidson, D. Warmsley und I. Weber, „Understanding Abuse: A Typology of Abusive Language Detection Subtasks“, in *Proceedings of the First Workshop on Abusive Language Online*, Aug. 2017, S. 78–84. doi: 10.18653/v1/W17-3012.

[11] M. Zampieri, S. Malmasi, P. Nakov, S. Rosenthal, N. Farra und R. Kumar, „Predicting the type and target of offensive posts in social media“, in *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, 2019, Bd. 1, S. 1415–1420. doi: 10.18653/v1/n19-1144.

[12] M. Zampieri, S. Malmasi, P. Nakov, S. Rosenthal, N. Farra und R. Kumar, „SemEval-2019 Task 6: Identifying and Categorizing Offensive Language in Social Media (OffensEval)“, *Proceedings of the 13th International Workshop on Semantic Evaluation*, S. 75–86, März 2019, [Online]. Verfügbar unter: http://arxiv.org/abs/1903.08983, Zugegriffen am: 01.09.2022.

[13] P. Liu, W. Li und L. Zou, „NULI at SemEval-2019 Task 6: Transfer Learning for Offensive Language Detection using Bidirectional Transformers“, in *Proceedings of the 13th International Workshop on Semantic Evaluation*, Juni 2019, S. 87–91. doi: 10.18653/v1/S19-2011.

[14] J. Han, S. Wu und X. Liu, „jhan014 at SemEval-2019 Task 6: Identifying and Categorizing Offensive Language in Social Media“, in *Proceedings of the 13th International Workshop on Semantic Evaluation*, Juni 2019, S. 652–656. doi: 10.18653/v1/S19-2116.

[15] A. Nikolov und V. Radivchev, „Nikolov-Radivchev at SemEval-2019 Task 6: Offensive Tweet Classification with BERT and Ensembles“, in *Proceedings of the 13th International Workshop on Semantic Evaluation*, Juni 2019, S. 691–695. doi: 10.18653/v1/S19-2123.

[16] H. Mubarak, A. Rashed, K. Darwish, Y. Samih und A. Abdelali, „Arabic Offensive Language on Twitter: Analysis and Experiments“, *Proceedings of the Sixth Arabic Natural Language Processing Workshop*, S. 126–135, Apr. 2020.

[17] Z. Pitenis, M. Zampieri und T. Ranasinghe, „Offensive Language Identification in Greek“, *Proceedings of the 12th Language Resources and Evaluation Conference*, S. 5113–5119, Mai 2020.

[18] G. I. Sigurbergsson und L. Derczynski, „Offensive Language and Hate Speech Detection for Danish“, Aug. 2019, [Online]. Verfügbar unter: http://arxiv.org/abs/1908.04531, Zugegriffen am: 01.09.2022.

[19] P. Nakov, V. Nayak, K. Dent, A. Bhatawdekar, S. Sarwar, M. Hardalov, Y. Dinkov,

D. Zlatkova, G. Bouchard und I. Augenstein, „Detecting Abusive Language on Online Platforms: A Critical Analysis“, Feb. 2021, [Online]. Verfügbar unter: http://arxiv.org/abs/2103.00153, Zugegriffen am: 01.09.2022.

[20] S. Rosenthal, P. Atanasova, G. Karadzhov, M. Zampieri und P. Nakov, „SOLID: A Large-Scale Semi-Supervised Dataset for Offensive Language Identification“, Apr. 2020.

[21] T. Baldwin, P. Cook, M. Lui, A. MacKinlay und L. Wang, „How Noisy Social Media Text, How Diffrnt Social Media Sources?“, 2013. [Online]. Verfügbar unter: http://rankings.big-boards.com, Zugegriffen am: 01.09.2022.

[22] Z. S. Harris, „Distributional Structure“, *WORD*, Bd. 10, Nr. 2–3, S. 146–162, Aug. 1954, doi: 10.1080/00437956.1954.11659520.

[23] C.-F. Tsai, „Bag-of-Words Representation in Image Annotation: A Review“, *ISRN Artificial Intelligence*, Bd. 2012, S. 1–19, Nov. 2012, doi: 10.5402/2012/376804.

[24] J. Brownlee, „Why One-Hot Encode Data in Machine Learning?“, *Machine Learning Mastery*, Juli 28, 2017. [Online]. Verfügbar unter: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/, Zugegriffen am: 01.09.2022.

[25] Y. T. Zhang, L. Gong und Y. C. Wang, „Improved TF-IDF approach for text classification“, *J Zhejiang Univ Sci*, Bd. 6 A, Nr. 1, S. 49–55, Jan. 2005, doi: 10.1631/jzus.2005.A0049.

[26] S. Qaiser und R. Ali, „Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents“, *Int J Comput Appl*, Bd. 181, Nr. 1, S. 25–29, Juli 2018, doi: 10.5120/ijca2018917395.

[27] Z. Yao und C. Ze-wen, „Research on the Construction and Filter Method of Stop-word List in Text Preprocessing“, in *2011 Fourth International Conference on Intelligent Computation Technology and Automation*, März 2011, S. 217–221. doi: 10.1109/ICICTA.2011.64.

[28] J. Ramos, „Using TF-IDF to Determine Word Relevance in Document Queries“, Piscataway, NJ, USA, 2003.

[29] D. L. Lee, Huei Chuang und K. Seamons, „Document ranking and the vector-space model“, *IEEE Softw*, Bd. 14, Nr. 2, S. 67–75, Apr. 1997, doi: 10.1109/52.582976.

[30] M. Mohammed und N. Omar, „Question classification based on Bloom’s taxonomy cognitive domain using modified TF-IDF and word2vec“, *PLoS One*, Bd. 15, Nr. 3, März 2020, doi: 10.1371/journal.pone.0230442.

[31] M. A. Lewis und G. A. Bekey, „The Behavioral Self-organization Of Nanorobots Using Local Rules“, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Juli 1992, S. 1333–1338. doi: 10.1109/IROS.1992.594558.

[32] M. Dorigo, V. Maniezzo und A. Colorni, „Ant system: optimization by a colony of cooperating agents“, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Bd. 26, Nr. 1, S. 29–41, Feb. 1996, doi: 10.1109/3477.484436.

[33] M. Dorigo, M. Birattari und T. Stutzle, „Ant colony optimization“, *IEEE Comput Intell Mag*, Bd. 1, Nr. 4, S. 28–39, Nov. 2006, doi: 10.1109/MCI.2006.329691.

[34] S.-A. Ahmadi, „Human behavior-based optimization: a novel metaheuristic approach to solve complex optimization problems“, *Neural Comput Appl*, Bd. 28, Nr. S1, S. 233–244, Dez. 2017, doi: 10.1007/s00521-016-2334-4.

[35] P.-P. Grassé, „La reconstruction du nid et les coordinations interindividuelles chezBellicositermes natalensis etCubitermes sp. la théorie de la stigmergie: Essai d’interprétation du comportement des termites constructeurs“, *Insectes Soc*, Bd. 6, Nr. 1, S. 41–80, März 1959, doi: 10.1007/BF02223791.

[36] M. Dorigo und L. M. Gambardella, „Ant colony system: a cooperative learning approach to the traveling salesman problem“, *IEEE Transactions on Evolutionary Computation*, Bd. 1, Nr. 1, S. 53–66, Apr. 1997, doi: 10.1109/4235.585892.

[37] B. Bullnheimer, R. F. Hartl und C. Strauss, „Applying the ANT System to the Vehicle Routing Problem“, in *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Boston, MA: Springer US, 1999, S. 285–296. doi: 10.1007/978-1-4615-5775-3_20.

[38] R. Schoonderwoerd, O. E. Holland, J. L. Bruten und L. J. M. Rothkrantz, „Ant-Based Load Balancing in Telecommunications Networks“, *Adaptive Behavior*, Bd. 5, Nr. 2, S. 169–207, Jan. 1997, doi: 10.1177/105971239700500203.

[39] H. Nezamabadi-pour, S. Saryazdi und E. Rashedi, „Edge detection using ant algorithms“, *Soft comput*, Bd. 10, Nr. 7, S. 623–628, Mai 2006, doi: 10.1007/s00500-005-0511-y.

[40] J. Xi, M. Spranger und D. Labudde, „Music Event Detection Leveraging Feature Selection based on Ant Colony Optimization“, Juni 2020.

[41] R. Soto, B. Crawford, F. Gonzalez, E. Vega, C. Castro und F. Paredes, „Solving the Manufacturing Cell Design Problem Using Human Behavior-Based Algorithm Supported by Autonomous Search“, *IEEE Access*, Bd. 7, S. 132228–132239, Sep. 2019, doi: 10.1109/ACCESS.2019.2940012.

[42] A. Ludwig, J. Felser, J. Xi, D. Labudde und M. Spranger, „FoSIL at CheckThat! 2022: Using Human Behaviour-Based Optimization for Text Classification“, Mittweida, Deutschland, 2022. [Online]. Verfügbar unter: https://www.researchgate.net/publication/363272661, Zugegriffen am: 27.11.2022.

[43] F. Rosenblatt, „The perceptron: A probabilistic model for information storage and organization in the brain.“, *Psychol Rev*, Bd. 65, Nr. 6, S. 386–408, 1958, doi: 10.1037/h0042519.

[44] I. Goodfellow, Y. Bengio und A. Courville, *Deep Learning: Das umfassende Handbuch. Grundlagen, aktuelle Verfahren, Algorithmen, neue Forschungsansätze*, Bd. 1. Frechen, Germany: mitp-Verlag, 2016.

[45] S. S. Haykin, *Neural networks and learning machines*. Upper Saddle River, New Jersey, USA: Pearson Education, 2009.

[46] D. Jurafsky und J. H. Martin, *Speech and Language Processing*, Bd. 3. 2020.

[47] T. Mikolov, I. Sutskever, K. Chen, G. Corrado und J. Dean, „Distributed Representations of Words and Phrases and their Compositionality“, Okt. 2013.

[48] J. Devlin, M.-W. Chang, K. Lee und K. Toutanova, „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding“, *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, S. 4171–4186, Okt. 2018.

[49] A. Radford, K. Narasimhan, T. Salimans und I. Sutskever, „Improving Language Understanding by Generative Pre-Training“, 2018.

[50] Y. Zhu *u. a.*, „Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books“, Juni 2015.

[51] A. Vaswani *u. a.*, „Attention Is All You Need“, *31st Conference on Neural Information Processing Systems NIPS 2017*, Juni 2017, [Online]. Verfügbar unter: http://arxiv.org/abs/1706.03762, Zugegriffen am: 01.09.2022.

[52] Y. Wu *u. a.*, „Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation“, Sep. 2016, [Online]. Verfügbar unter: http://arxiv.org/abs/1609.08144, Zugegriffen am: 01.09.2022.

[53] C. Sun, X. Qiu, Y. Xu und X. Huang, „How to Fine-Tune BERT for Text Classification?“, *China International Conference on Chinese Computational Linguistics*, S. 194–206, Mai 2019.

[54] S. Madisetty und M. S. Sesarkar, „Aggression Detection in Social Media using Deep Neural Networks“, *Proceedings of the First Workshop on Trolling, Aggression*

and Cyberbullying (TRAC-2018), S. 120–127, Aug. 2018.

[55] A. Jiang, X. Yang, Y. Liu und A. Zubiaga, „SWSR: A Chinese Dataset and Lexicon for Online Sexism Detection“, Aug. 2021.

[56] S. Frenda, B. Ghanem, M. Montes-y-Gómez und P. Rosso, „Online Hate Speech against Women: Automatic Identification of Misogyny and Sexism on Twitter“, *Journal of Intelligent & Fuzzy Systems*, Bd. 36, Nr. 5, S. 4743–4752, Mai 2019, doi: 10.3233/JIFS-179023.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Oelsnitz/Erz., den 30.09.2022

Tim Wetterau