The Quest(ion) of Implicit Hate Speech Detection

Florian Kankowski
3974289
florian.kankowski@uni-bielefeld.de

Abstract

Das Problem von *hate speech detection* ist in NLP-Kreisen seit einigen Jahren stark im Umlauf. Als verhältnismäßig neue Teildisziplin existieren noch einige Grundlagenfragen, zu denen kein einschlägiger Konsens besteht. Häufig diskutiert sind die Definition von Hassrede und ihre Abgrenzung von anderen Formen von Rede. Die vorliegende Arbeit befasst sich mit der Art und Weise, wie Hass in der Rede kodiert wird, und beleuchtet die Frage, ob implizite – oder koverte – *hate speech* sich kategoriell von expliziter unterscheidet und für NLP-Zwecke als eigenständiges, separates Problem aufgefasst werden sollte. Um diese Frage zu beantworten, wird ein möglichst kompetenter Classifier für implizite Hate speech konstruiert und vergleichend geprüft. Damit wird gezeigt, dass messbare Unterschiede zwischen *implicit* und *explicit hate speech detection* existieren.[1]

1 Einführung

Automatic hate speech detection ist ein bekanntes Problem, welches nicht nur akademisch interessant ist, sondern auch in der „echten Welt“ immer wichtiger wird. Besonders für automatische Moderation in Internetforen oder auf sozialen Medien, wo aufgrund der großen Nutzerbasis eine menschliche Moderation unmöglich ist, ist der Bedarf für robuste automatisierte Systeme groß.

Viel Aufmerksamkeit ist dem Begriff *hate speech* selbst zugekommen, der Suche nach einem Konsens für die Definition und den Möglichkeiten und Schwierigkeiten für automatische Erkennung von *hate speech*.

Die meisten der bisherigen Arbeiten widmen sich *hate speech* allgemein, meistens sogar offener Hassrede, bzw. *explicit hate* (zum Beispiel in Davidson et al., 2017). Aber auch die Thematik von *implicit hate* wird immer interessanter und wird zum Beispiel dort beleuchtet, wo die Typologie von Hass analysiert wird (Waseem et al., 2017).

Diese Arbeit befasst sich mit der Überlegung, ob *hate speech* eine monolithische Kategorie von Rede darstellt, welche zum Beispiel von automatischen Systemen oder Moderationsalgorithmen holistisch detektiert werden sollte. Genauer wird die Fragestellung beleuchtet, ob implizite und explizite Hassrede als eigenständige, getrennte Problemstellungen aufgefasst werden sollten.

Diese Überlegungen könnten überall dort nützlich sein, wo *automatic hate speech detection* betrieben wird: Sind nämlich explizite und implizite Hassrede als eigenständige Probleme zu verstehen, wäre es womöglich sinnvoll, dafür getrennt automatische Erkennungssysteme einzurichten und ML-Modelle zu trainieren.

2 Explizit vs. Implizit

Genau wie für den Begriff *hate speech* an sich ist es wichtig, eine Abgrenzung von impliziter und expliziter Hassrede zu definieren.

Grundsätzlich wird in *explicit hate speech* die Botschaft auf overte, offensichtliche Art geäußert, während sie im Falle von *implicit hate speech* auf die eine oder andere Art verschleiert wird. Die Herausforderung hierbei liegt darin, dass es viele verschiedene, teils sehr unterschiedliche Methoden gibt, eine Botschaft zu verschleiern. So kann eine Aussage zum Beispiel mit Ironie oder metaphorischer Sprache mehrdeutig gemacht werden, durch die Verwendung von falsch geschriebenen oder abgewandelten Begriffen obfuskiert werden oder mithilfe von *inside-jokes* oder gruppenspezifischen Referenzen für Außenstehende unzugänglich gemacht werden.

Diese Mehrdimensionalität, mit der *implicit hate speech* ausgedrückt werden kann, ist die große Herausforderung für die automatische Erkennung

[1] Die Daten dieser Arbeit sind auf Github abrufbar.

und eine große Motivation, sie von der vergleichsweise einfach einzugrenzenden *explicit hate speech* abgrenzen zu wollen.

In dieser Arbeit sollen zwei Datensätze betrachtet werden, die sich diesem Phänomen widmen. „Latent Hatred: A Benchmark for Understanding Implicit Hate Speech“ (ElSherief et al., 2021) beinhaltet Twitter-Posts mit impliziten, expliziten und nicht hasserfüllten Tweets. Sie definieren die Domäne von *implicit hate* folgendermaßen:

> „Implicit hate speech is a subclass of hate speech defined by the use of coded or indirect language such as sarcasm, metaphor and circumlocution to disparage a protected group or individual, or to convey prejudicial and harmful views about them.“
>
> (ElSherief et al., 2021, S. 2)

Der zweite Datensatz stammt aus „Introducing the Gab Hate Corpus: Defining and applying hate-based rhetoric to social media posts at scale“ (Kennedy et al., 2018), dessen Gab-Posts unter anderem danach kategorisiert sind, ob ihre Botschaft implizit oder explizit kodiert ist. Die Differenzierung zwischen diesen Kategorien wird dort beschrieben als

> „[...] roughly analogous to the distinction in linguistics and semiotics between denotation, the literal meaning of a term or symbol, and connotation, its sociocultural associations. [...] Implicit rhetoric is most often an invocation of derogatory beliefs, sentiments, or threats which are accessible through cultural knowledge.“
>
> (Kennedy et al., 2018, S. 15)

Die beiden Definitionen sind nicht deckungsgleich (die erste legt einen größeren Schwerpunkt auf sprachliche Mittel, die zweite auf kulturelle Phänomene, um eine Aussage implizit zu gestalten), aber sie sind miteinander vereinbar.

3 Durchführung

Die übergreifende Fragestellung dieser Arbeit befasst sich mit dem Unterschied zwischen impliziter und expliziter *hate speech*. Um diese Frage zu beantworten, soll ein Classifier geschaffen werden, welcher darauf ausgerichtet ist, coverte *hate speech* zu erkennen. Das Ziel ist, diesen Classifier für explizite *hate speech detection* zu verwenden und aus der Performance darin Rückschlüsse für die Fragestellung zu ziehen.

Daraus ergibt sich neben der eigentlichen Fragestellung eine zweite – eher technische – Fragestellung, die sich damit befasst, wie ein solcher Classifier sinnvollerweise gebaut werden kann. Der Anspruch dieses Classifiers im Sinne der Arbeitshypothese ist es, lediglich eine solide Baseline mit akzeptablen *scores* zur Verfügung zu stellen, die messbare Ergebnisse liefert. Eine möglichst optimale Performance des finalen Modells ist aber auch für sich gesehen eine interessante und wertvolle Aufgabe. Sollte sich nämlich herausstellen, dass es tatsächlich sinnvoll sein kann, implizite und explizite *hate speech detection* als disjunkte Probleme aufzufassen, so können die hier angestellten Erkenntnisse und Ergebnisse genutzt werden, um darauf aufbauend in der Zukunft bessere Modelle zu erschaffen. Diese Arbeit verfolgt also zwei Ziele: Die Beantwortung einer wissenschaftlichen und einer technischen Frage.

Der größte Teil der Arbeit befasst sich mit den technischen Aspekten des Classifier-Erschaffens.

In einem ersten Schritt soll ein grober Überblick über die vorhandenen Methoden und Systeme geschaffen werden. Auch wenn es reizvoll ist, auf *state-of-the-art*-Methoden wie BERT oder andere neuronale Netze zurückzugreifen, werden hier nur simple ML-Modelle betrachtet. Dies hat zwei hauptsächliche Gründe. Erstens sind simple Systeme üblicherweise relativ transparent und erlauben Einsicht in ihre Arbeitsweise, sodass es möglich ist, Features, Performance und Entscheidungsgründe zu analysieren. Zweitens sind solche Systeme verhältnismäßig wenig komputationell anspruchsvoll. Dies erlaubt es, viele verschiedene Systeme, Hyperparameterkonfigurationen und Featuresets zu vergleichen, ohne dafür zu viel Rechenzeit aufwenden zu müssen.

Ein eindeutiger Nachteil dabei ist, dass das Potential solcher Modelle nicht mit dem neuronaler Systeme mithalten kann. Es ist intuitiv anzunehmen, dass das Problem von *implicit hate* stark nonlinear und sehr tiefliegend ist, was Modelle wie *logistic regression classifiers* vor Probleme stellen kann. Da allerdings für viele NLP-Anwendungen auch simple Modelle leistungsstark genug sind (wie zum Beispiel in Davidson et al., 2017 selbst), soll dieser Nachteil nicht zu sorgenbereitend sein.

Neben dem Modell selbst ist auch die Wahl der Features sehr relevant. Als Fundament sollen *vec-*

tor embeddings als semantische Repräsentation der Texte verwendet werden. Wie bei den Modellen auch gibt es hier mehrere Möglichkeiten zur Auswahl. Ziel des ersten Schrittes ist es, ein ML-Modell und ein *embedding* zu finden, die eine möglichst gute Leistung zeigen und mit denen im weiteren Verlauf weitergearbeitet werden kann.

Darüber hinaus sollen weitere Features, die sich mit den oberflächlichen oder graphemischen Eigenschaften der Texte befassen, in das System eingebaut und getestet werden.

Nachdem ML-Modell und Featureset ausgewählt werden, sollen sie tiefergehend analysiert und optimiert werden durch Anwendung von Hyperparameter-Finetuning und Feature-Analyse.

Anschließend werden weitere potenzielle Features, sowohl solche aus vorhandener Literatur als auch eigens ersonnene, implementiert, verbessert und getestet. Der Prozess wird in den jeweiligen Abschnitten genauer beschrieben.

Das Ergebnis ist ein möglichst optimierter Classifier für *implicit hate speech*. Dieser kann genutzt werden, um *out-of-domain*-Daten zu klassifizieren und einer Antwort auf die Ausgangsfrage näher zu kommen.

3.1 Erstes Setup

3.1.1 Daten

Beide eingangs gewählten Datensätze scheinen sinnvoll für die vorliegende Aufgabe. Aber sowohl die Daten aus ElSherief et al. (2021) wie auch die aus Kennedy et al. (2018) sind *multi-class-annotated* und enthalten darüber hinaus noch weitere annotierte Labels, die die Posts feiner klassifizieren, zum Beispiel nach Zielgruppe oder Art der Hassrede. Um die Domäne des Problems möglichst einzugrenzen, soll jedoch lediglich eine binäre Klassifizierung *implicit hate* und *non-hate* angestellt werden.

Für die Twitter-Posts aus ElSherief et al. (2021) werden alle Tweets hinausgefiltert, die als `explicit_hate` gelabelt sind. Alle anderen Kategorien werden ignoriert. Somit ergibt sich ein nutzbarer Datensatz mit passenden, binären Gold-Labeln. Von insgesamt 21.481 Posts sind 20.391 nutzbar. Davon sind 7.100 positiv (*hate*) und 13.291 negativ. Somit ist der Datensatz im Verhältnis von fast 1:2 unbalanciert. Für das Trainieren des Modells sollte das kein Problem darstellen, jedoch muss der Umstand bei der Bewertung der Classifier beachtet werden – besonders bei der *accuracy*-Metrik –, da die *minority class* die interessantere ist.

Die Gab-Posts aus Kennedy et al. (2018) werden ähnlich gehandhabt. Explizit vs. implizit wird in einer eigenen Kategorie festgehalten, die gefiltert werden kann. Zwar unterscheidet der Datensatz verschiedene Arten von *hate speech*, aber für diese Zwecke werden sie alle in eine positive Kategorie kollabiert. Ein Unterschied zum anderen Datensatz ist, dass hier alle Annationen der einzelnen *annotater* einzeln aufgeführt sind. Um den Datensatz zu vereinheitlichen, werden alle Duplikate herausgefiltert, sodass der letztendliche Datensatz 25.119 negative und 2.210 positive Posts enthält, was sehr unbalanciert und wenig ideal ist. Da der Datensatz nicht besonders groß ist, können allerdings Methoden wie *downsampling* sinnvollerweise nicht angewendet werden.

3.1.2 Embeddings

Es ist wünschenswert, einen Classifier erschaffen zu können, der möglichst stark und verlässlich ist unter Einsatz von möglichst wenigen, aber dafür prägnanten Features. *Vector embeddings* sind hierbei ein sehr nützliches Werkzeug, da sie in der Lage sind, vielfältige sprachliche Informationen zu repräsentieren, denn neben Wortsemantik erfassen sie auch grammatische und pragmatische Aspekte von Wörtern.

Ein Vorteil gegenüber zum Beispiel TF-IDF-Modellen zur Wortrepräsentationen ist die verhältnismäßig geringe Dimensionalität und die damit einhergehende große Informationsdichte. Nützlich sind ebenfalls vortrainierte *embeddings*, basierend auf vielen Milliarden Datenpunkten, die semantische Repräsentation auch für im *training set* nicht vorkommende Wörter enthalten können.

Vector embeddings haben auch Nachteile – zum Beispiel können sie üblicherweise keine Kontexteffekte wie Synonymien erfassen –, doch für die vorliegende Problematik sind sie gut geeignet. Für dieses Projekt werden drei verschiedene Embeddings verglichen.

GloVe (Pennington et al., 2014) (*Glo*bal *Ve*ctors) generiert für jedes Wort aus einem Korpus eine Vektorrepräsentation auf Basis globaler Kookkurrenzen zwischen Wörtern. Für diese Arbeit wird eine vortrainierte GloVe-Implementation verwendet, die auf Twitter-Daten trainiert und auf 50 Dimensionen reduziert wurde. Wörter, die nicht im Korpus erscheinen, werden von GloVe nicht erfasst und darunter fallen auch flektierte Formen. Alle

Texte werden daher normalisiert, tokenisiert und gestemmt.

Ein Nachteil von GloVe ist, dass ein Vektor pro Wort vorliegt und Texte eine variable Länge besitzen, die Featuresets aber eine feste Länge benötigen. Deswegen wird für jeden Text ein einzelner Vektor generiert, indem pro Dimension der Durchschnitt aller Dimensionen errechnet wird. Um dadurch potenziell aufkommendem *noise* entgegenzuwirken, werden Stopwörter hinausgefiltert.

Dieses Problem tritt bei dem **Doc2Vec**-Modell (Le and Mikolov, 2014) nicht auf. Doc2Vec, eine Weiterentwicklung von **Word2Vec** für supraphrasale Einheiten, stellt für jeden Text eine Vektorrepräsentation von fester Länge zur Verfügung. Ein Nachteil von Doc2Vec ist, dass das Modell domänenspezifisch ist, sodass keine vortrainierte Implementation verwendet werden kann. Somit ist die Qualität der Vektoren abhängig von den zur Verfügung steheneden Trainingsdaten.

Für diese Zwecke wäre es möglich, beide Korpora zu vereinen und ein einheitliches Doc2Vec-Modell zu trainieren, doch für Testzwecke wurden sie separat behandelt. Trainiert wurde auf 500 Dimensionen mit einem Kontextfenster von 3 Wörtern in beide Richtungen. Der hier verwendete Trainingsalgorithmus ist *distributed bag of words* (DBOW).

Zuletzt wird **Fasttext** (Bojanowski et al., 2016) geprüft. Ähnlich wie in GloVe stellt Fasttext eine Vektorrepräsentation pro Wort zur Verfügung. Darüber hinaus lernt der Fasttext-Algorithmus jedoch *subword*-Informationen, die es erlauben, auch für im Trainingskorpus unbekannte Wörter einen Vektor zu generieren, zum Beispiel für Komposita, Schreibfehler oder neologistische Derivationen. Für implizite *hate speech* kann dies besonders dort interessant sein, wo hasserfüllte Wörter durch absichtliches Falschschreiben obfuskiert werden.

3.1.3 ML-Modelle

Fünf verschiedene – relativ simple – Modelle werden miteinander verglichen. **Logistische Regression** ist ein schneller und überschaubarer Algorithmus, der häufig für NLP-Tasks verwendet wird (wie zum Beispiel bei Davidson et al. (2017)). Ein Vorteil von logistischer Regression ist die hohe Interpretierbarkeit: Die Gewichtung von jeder Einzelkomponente ist direkt einsehbar, was elementar ist, um die Features zu bewerten. Da es sich bei *implicit hate speech detection* allerdings womöglich um ein stark nonlineares Problem handelt, könnte es sein, dass ein linearer Classifier wie logistische Regression nicht mächtig genug ist.

Ähnlich wie logistische Regression sind **Decision Trees** sehr transparente Systeme, da sie erlauben, den Klassifizierungsprozess im generierten Entscheidungsbaum, welcher im Zuge des Trainings aufgebaut wird, direkt einzusehen. So können auch die Gewichtungen der einzelnen Features sowie die Entropie der fertigen Struktur ermittelt werden. Für manche Anwendungen sind Decision Trees leistungsstärker als logistische Regression, doch sie neigen dazu, schnell auf die Trainingsdaten zu overfitten. Hier verwendet werden ein Decision Tree Classifier und ein Decision Tree Regressor.

Dieses Problem liegt bei **Random Forests** weniger vor. Hierbei handelt es sich um ein Ensemble-System, bei dem mehrere Decision Trees konstruiert werden, welche gemeinsam zum Klassifizieren verwendet werden. Üblicherweise zeigen Random Forests daher bessere Ergebnisse als ein einzelner Decision Tree und sie sind darüber hinaus robuster, wenn man sie auf Fremddaten anwendet. Die Interpretierbarkeit von Decision Trees geht allerdings verloren.

Latent Dirichlet Allocation (LDA) ist, ähnlich wie logistische Regression, ein lineares Modell, was üblicherweise eher bei *multiclass*-Problemen präferiert ist. LDA ist für dieses Problem ein interessantes Benchmark, da sie ein sehr simples System ist, was einige reduktive Annahmen über die Daten macht. Zeigt LDA robuste und gute Ergebnisse, wäre das ein Indiz dafür, dass das Problem doch nicht allzu nonlinear oder komplex ist.

Zuletzt wird eine **Support Vector Machine** (SVM) betrachtet. Dieses Modell konstruiert, ähnlich wie logistische Regression, eine Hyperplane im Featurespace, um die Klassifizierung durchzuführen. Eine SVM optimiert jedoch darauf, eine möglichst hohe Trennbarkeit der Klassen herbeizuführen, was Overfitting entgegenwirkt. Ebenfalls sind SVMs besser ausgestattet, um mit nichtlinearen Klassengrenzen umzugehen. Ein großer Nachteil ist der im Vergleich zu allen anderen vorgestellten Methoden gewaltige Rechenaufwand für das Fitting.

3.1.4 Weitere Features

Neben den semantischen *embeddings* werden den Systemen einige weitere Features zur Verfügung gestellt, welche möglicherweise sinnvoll für die Aufgabe sein können. Dabei handelt es sich sowohl

um oberflächliche Eigenschaften des Textes, welche die *embeddings* nicht erfassen können, nämlich Satzzeichenfrequenz und Verwendung von Wörtern in *ALL CAPS*, als auch um Features, die spezifisch für Online-Posts relevant sind, nämlich Verwendung von Emojis, Hashtags und Mentions / Retweets (nur für Twitter anwendbar).

Diese Features wurden nicht gewählt, weil sie domänenspezifisch für *hate speech* anwendbar sind, sondern weil sie die Systeme mit weiteren Textinformationen anreichern, die zum Inferieren nützlich sein können.

3.2 Vergleich von Modellen

Für eine grobe Übersicht werden alle ML-Modelle in allen Kombinationen gegeneinander getestet. Das bedeutet, dass jedes Modell mit jedem *embedding* für jeden der beiden Datensätze trainiert und getestet wird, je einmal mit und ohne die weiteren Features (aus 3.1.4). Damit ergeben sich $5 \times 3 \times 2 = 30$ verschiedene Kombinationen pro Datensatz.

Die Systeme wurden mit einem 4:1-Train-Test-Split getestet ohne Verwendung von Cross-Validation oder Bootstrapping, da lediglich ein erster Benchmark angelegt werden sollte. Für jedes ML-Modell wurde ein grobes Finetuning der Hyperparameter betrieben (was eventuell ein Oxymoron ist), damit der Vergleich zumindest einigermaßen aussagekräftig ist.

Da beide Datensätze nicht balanciert sind, ist die *accuracy* wenig aussagekräftig, weswegen auch *precision*- und *recall*-Scores gemessen wurden.

Tabelle 1 zeigt die Ergebnisse für die Daten aus ElSherief et al. (2021). Auf den ersten Blick ist ersichtlich, dass bereits die *accuracy*-Scores wenig ideal sind. Eine *random baseline* gibt etwa 50% *accuracy* und *recall* und etwa 35% *precision*, sodass alle Classifier in 2 von 3 Kategorien schlechter abschneiden als eine Zufallswahl. Es ist ein deutlicher Trend, dass *precision* deutlich höher ausfällt als *recall*. Das deutet darauf hin, dass die Systeme eher selten – und zwar deutlich zu selten – ein positives Label vorhersagen, dies dafür jedoch auch selten falsch ist.

Ein solches verhalten ist bei *implicit hate speech* nicht unerwartet, da die subtile Art, wie Hass ausgedrückt werden kann, sich häufig nur kaum von normaler Kommunikation abhebt, was die Diskriminierung für ein automatisches System sehr schwer macht.

Ein Vergleich der ML-Modelle zeigt, dass die Decision Tree und SVM-Algorithmen in allen Kategorien am besten abschneiden. Interessant ist die schlechte Leistung von logistischer Regression und die verhältnismäßig robuste Leistung von LDA für ein scheinbar so komplexes Problem. Dass der Random Forest schlechtere Performance zeigt als die Decision Trees, ist an sich nicht verwunderlich, da die bessere Performance von Random Forests sich üblicherweise erst bei der Anwendung auf Fremddaten zeigt, aber die *recall*-Scores im einstelligen Prozentbereich sind dennoch auffällig schlecht.

Ein Vergleich zwischen den *embeddings* zeigt nur geringfügige Unterschiede. Keine Repräsentation ist holistisch besser als eine andere, aber insgesamt zeigt Fasttext die besten Ergebnisse, was an der höheren Anzahl an Dimensionen im Vergleich zu GloVe oder der Möglichkeit, *subword*-Strukturen analysieren zu können, liegen mag.

Der Vergleich zwischen den Modellen nur mit *embeddings* und sowohl mit *embeddings* plus weiteren Features ist interessant, da für Aufgaben, die sich mit subtilen sprachlichen Phänomenen befassen, solche eher oberflächlichen Features wenig aussagekräftig oder sogar schädlich sein können. Vidgen and Yasseri (2020) zeigen zum Beispiel, dass bei einem Classifier für die Diskriminierung von starker und schwacher Islamophobie das optimale Featureset nur aus den *embeddings* besteht. In Tabelle 1 lässt sich ablesen, dass die weiteren Features nicht signifikant bessere, manchmal sogar schlechtere Classifier erzeugen.

Insgesamt scheint hier ein Decision Tree auf Basis von Fasttext und weiteren Features das beste Modell zu sein, aber auch eine SVM mit dem gleichen Featureset ist vielversprechend.

Tabelle 2 zeigt die Ergebnisse für die Daten aus Kennedy et al. (2018). Die gleichen Trends zeichnen sich hier nicht ab. Die Classifier haben scheinbar große Probleme mit den Daten und machen gelegentlich gar keine positiven Voraussagungen für das gesamte Testset. Da der Datensatz allerdings so unbalanciert ist, herrscht trotzdem eine große *accuracy*. Dies könnte auch der Grund für die niedrige Performance sein.

Einen möglichen weiteren Grund für die schlechten Metriken nennen Kennedy et al. (2018) selbst:

> „The low [inter annotator] agreement for Implicit/Explicit rhetoric casts doubts on their usefulness and informativeness."
>
> (Kennedy et al., 2018, S. 22)

		Nur *embeddings*			*embeddings* + weitere Features		
Modell		***accuracy***	***precision***	***recall***	***accuracy***	***precision***	***recall***
GloVe	Logistische Regression	.365	.659	.035	.372	.710	.046
	Random Forest	.369	.750	.033	.369	.783	.032
	Decision Tree Classifier	.464	.694	.303	.450	.676	.285
	Decision Tree Regressor	.449	.670	.290	.455	.681	.294
	LDA	.365	.674	.033	.372	.723	.044
	SVM	.496	.683	.412	*.508*	*.690*	*.434*
Doc2Vec	Logistische Regression	.391	.750	.085	.404	.768	.109
	Random Forest	.377	.684	.066	.388	.716	.086
	Decision Tree Classifier	.460	.680	.308	.450	.673	.289
	Decision Tree Regressor	*.466*	*.686*	*.318*	.453	.681	.289
	LDA	.400	.764	.103	.405	.756	.116
	SVM	.460	.656	.346	.427	.625	.283
Fasttext	Logistische Regression	.425	.783	.151	.427	.782	.155
	Random Forest	.370	.758	.037	.367	.728	.032
	Decision Tree Classifier	.468	.688	.321	***.472***	***.693***	***.327***
	Decision Tree Regressor	.471	.686	.332	.470	.688	.328
	LDA	.426	.787	.152	.429	.795	.156
	SVM	.451	.664	.303	.451	.673	.292

Tabelle 1: Scores der Classifier auf Basis der Daten aus ElSherief et al. (2021). Die in **fett** markierte Kombination zeigt die besten Ergebnisse insgesamt. In *kursiv* sind die jeweils besten Classifier für jedes *embedding* markiert.

		Nur *embeddings*			*embeddings* + weitere Features		
Modell		***accuracy***	***precision***	***recall***	***accuracy***	***precision***	***recall***
GloVe	Logistische Regression	***.909***	***.500***	***.004***	***.909***	***.500***	***.004***
	Random Forest	.909	.000	.000	.909	.000	.000
	Decision Tree Classifier	.832	.100	.107	.826	.103	.119
	Decision Tree Regressor	.831	.102	.111	.821	.091	.109
	LDA	.908	.375	.012	.909	.583	.014
	SVM	.847	.064	.050	*.850*	*.085*	*.066*
Doc2Vec	Logistische Regression	.909	.200	.002	.908	.182	.004
	Random Forest	*.909*	*.400*	*.004*	.908	.222	.004
	Decision Tree Classifier	.828	.116	.135	.832	.129	.147
	Decision Tree Regressor	.830	.106	.117	.826	.122	.149
	LDA	.906	.179	.010	.904	.171	.014
	SVM	.859	.141	.109	.855	.134	.109
Fasttext	Logistische Regression	.908	.348	.016	*.907*	*.355*	*.022*
	Random Forest	.909	.000	.000	.909	.000	.000
	Decision Tree Classifier	.817	.107	.139	.826	.121	.147
	Decision Tree Regressor	.815	.103	.135	.828	.106	.121
	LDA	.904	.262	.032	.904	.294	.040
	SVM	.848	.081	.064	.860	.116	.083

Tabelle 2: Scores der Classifier auf Basis der Daten aus Kennedy et al. (2018). Die in **fett** markierte Kombination zeigt die besten Ergebnisse insgesamt. In *kursiv* sind die jeweils besten Classifier für jedes *embedding* markiert.

Diese ersten Experimente deuten also darauf hin, dass dieser Datensatz möglicherweise doch nicht gut geeignet für diese Aufgabe ist. Daher werden die Daten aus Kennedy et al. (2018) in dieser Arbeit nicht weiterverwendet.

3.3 Optimierung

Der große Vorteil eines Decision Trees – Interpretierbarkeit – geht verloren, wenn der Baum genügend groß ist. Der im ersten Experiment generierte Classifier stellt einen sehr komplexen, tiefen, verzweigten Baum dar, wie Abbildung 1 zeigt.

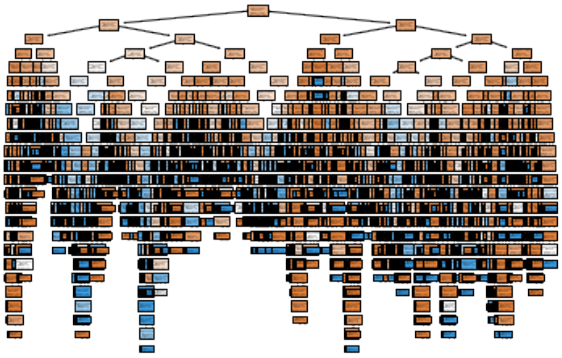

Abbildung 1: Visualisierung des besten Decision Tree Classifiers aus 3.2.

Ein Decision Tree, der ohne Restriktion wachsen darf, leidet üblicherweise an starkem Overfitting. Um dem entgegenzuwirken, wurde Hyperparameter-Finetuning betrieben, und zwar die maximale Tiefe des Baumes sowie der Threshold für einen Knoten, um sich zu spalten. Dazu wurden in einem ersten Schritt „per Hand“ grob sinnvolle Bereiche für die Hyperparameter ermittelt, in denen mithilfe einer exhaustiven Suche die optimale Kombination gefunden wurde.

Der resultierende Baum (Abbildung 2) ist noch immer hoch komplex, jedoch deutlich kompakter. Allerdings ist der neue Classifier auch weniger stark als der alte (siehe Tabelle 3), weswegen es sinnvoll erscheint, die SVM ebenfalls zu optimieren, um dort noch bessere Scores zu erlangen. So oder so bleibt der hier erzeugte Decision Tree allerdings ein sinnvolles Werkzeug für eine spätere Feature-Analyse.

Wie in 3.1.3 bereits erwähnt ist die SVM das rechen- und damit zeitintensivste Modell, weswegen Hyperparameter-Finetuning hier ein aufwendiger Prozess ist. Dieser Umstand wird nur noch verstärkt dadurch, dass die einzelnen Parameter wesentlich weniger unabhängig voneinander sind

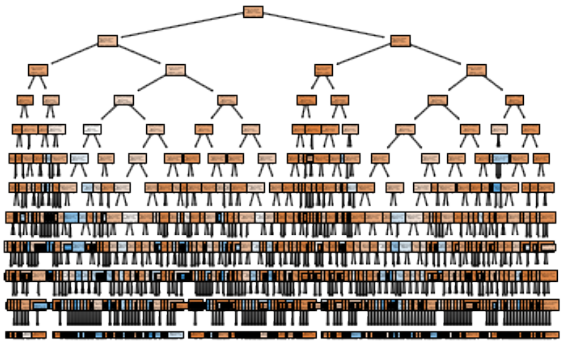

Abbildung 2: Visualisierung des gesäuberten Decision Tree Classifiers aus 3.3.

als beim Decision Tree, sodass sie nicht einzeln angepasst werden können. Da in diesem Prozess drei Parameter der SVM optimiert wurden (die L2-Regularisierung, der Kernel-Koeffizient sowie der Kerneltyp), öffnet sich ein großer Parameterraum an möglichen Werten, in dem nach der besten Kombination gesucht werden muss.

Dafür eignet sich zum Beispiel ein *Bayes' search*, der sich innerhalb eines Parameterraums inkrementell einem optimalen Ergebnis annähert. Eine andere gute Möglichkeit ist ein *random search*, jedoch konnten aus technischen Gründen beide Methoden nicht angewendet werden.

Stattdessen wurde erst ein grober *exhaustive grid search* durchgeführt, um den Parameterraum einzuschränken, und mit einem zweiten *grid search* darin das beste Arrangement gefunden.

Dabei wurde die Beobachtung aus Tabelle 1 mit einbezogen, dass die Systeme tendenziell eher einen höheren *precision*- als *recall*-Wert erzielen. Die verschiedenen Parameterkonstellationen wurden daher auf Basis eines *f1.5*-Scores bewertet. Frühere Tests haben gezeigt, dass eine *f1*-Bewertung oder eine reine *accuracy*-Bewertung dazu führen, dass die Modelle eine sehr hohe *precision*, aber eine extrem niedrige *recall* besitzen, was für das vorliegende Problem sehr unerwünscht ist. Mit einer Optimierung nach *f1.5* treten diese Probleme nicht auf.

Darüber hinaus wird jede Kombination von Hyperparametern unter Einbeziehung von *3-fold cross-validation* getestet, um Outliern entgegenzuwirken und die Metriken zu stabilisieren.

Tabelle 3 zeigt die Ergebnisse des Optimierungsprozesses. Die optimierte SVM ist als einziger Classifier besser in allen drei Metriken als die Random Baseline. Dies ist immerhin ein vielversprechen-

des Ergebnis und es scheint sinnvoll, mit diesem Modell weiterzuverfahren.

Modell	***acc.***	***prec.***	***rec.***
Random Baseline	.507	.354	.505
Decision Tree einfach	.472	.693	.327
Decision Tree optimiert	.670	.535	.427
SVM einfach	.451	.673	.292
SVM optimiert	.732	.626	.549

Tabelle 3: Scores der verschiedenen Classifier bis 3.3.

3.4 Feature-Analyse

Das Modell verfügt über insgesamt 312 Features, von denen 300 die Dimensionen der *embeddings* sind. Wie sich bereits gezeigt hat, scheinen die zusätzlichen Features keinen großen Einfluss auf die Qualität der Modelle zu haben, potenziell sogar einen leicht negativen.

Der optimierte Decision Tree Classifier aus 3.3 erlaubt es, die Wichtigkeit jedes einzelnen Features einzusehen. Aufgrund der Constraints an die Baumtiefe und der Knotenaufspaltung ist es nicht verwunderlich zu sehen, dass 120 der Features gar keinen Einfluss auf den Classifier haben, etwa 38%. Nur zwei der nicht-*embeddings*-Features sind für den Baum relevant, nämlich `is retweet` und `number of ,`. Letzteres ist sogar das relevanteste Feature überhaupt, was sehr interessant ist. Ein Grund dafür konnte leider nicht gefunden werden.

Eine Varianzanalyse der Features zeigt eine 0-Varianz von Emojis und Anführungszeichen. Tatsächlich scheinen Emojis im Datensatz von ElSherief et al. (2021) herausgefiltert zu sein, so wie andere Unicode-Zeichen auch. Anführungszeichen hingegen werden im Preprocessing bereinigt, da sie als Substitionszeichen für eine große Menge anderer Entitäten eingesetzt werden und somit nur unnötigen Noise darstellen. Beide Features sollten daher entfernt werden.

Die Varianz der *embeddings* zeigt einige, wenige deutliche Peaks. Dies deutet darauf hin, dass es vermutlich semantische Informationen in den varianzstarken Komponenten gibt, die für die Aufgabe relevant sind. Einige Dimensionen zeigen kaum Varianz und es könnte sinnvoll sein, sie zu entfernen, da sie nur eine geringe Informativität besitzen und quasi Noise sind.

Eine Bewertung der Features nach Pearsons Korrelationsskoeffizient bestätigt, dass Emojis und Anführungszeichen ungeeignet sind, gibt aber keine weitere starke Auskunft über die anderen Features. Es scheint sinnvoll zu sein, für den weiteren Verlauf der Arbeit bis die niedrigvariaten Komponenten von Fasttext aufs Erste zu behalten und auch die weiteren Features zu benutzen.

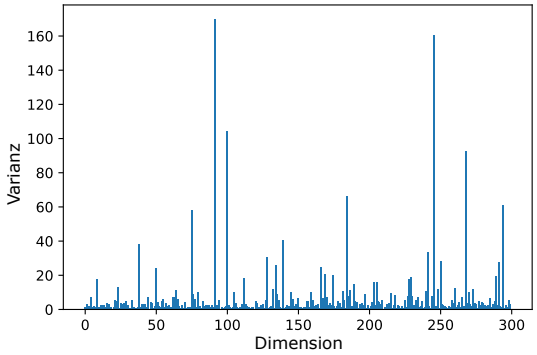

Abbildung 3: Varianz der Komponenten der Fasttext-*embeddings* in den Posts aus ElSherief et al. (2021).

3.5 Fasttext-Finetuning

Die Fasttext-Implementation, die bisher benutzt wurde, ist auf mehreren Milliarden Datenpunkten trainiert und ist somit bereits sehr informativ. Aber es besteht die Möglichkeit, dass die Aussagekraft der Vektoren durch Finetuning mit domänenspezifischen Daten präzisiert werden kann.

Die Daten liegen in einem Format vor, das das Weitertrainieren erlaubt. Da die Gab-Posts aus Kennedy et al. (2018) bisher keine guten Ergebnisse für die Classifier geliefert haben, ist es sicherer, nur die Twitter-Daten aus ElSherief et al. (2021) zu verwenden.

Um eventuellen Problemen vorzubeugen, werden für das Finetuning die gleichen Hyperparameter genommen, mit denen die Vektoren ursprünglich trainiert wurden (und zwar ein Kontextfenster von 5 Wörtern unter Verwendung des CBOW-Algorithmus). Lediglich die Anzahl an *epochs* ist offen. Da es keine klare Antwort gibt, wie viele *epochs* optimal sind, werden mehrere *embeddings* durch Finetuning erstellt und jeweils in einem Classifier implementiert, deren Scores verglichen werden können.

Insgesamt wurden Fasttext-Implementationen mit 1, 5, 25, 150, 625 und 3125 trainiert und getestet. Dies hat etwa 5 Stunden Rechenzeit in Anspruch genommen mit dem Ergebnis, dass Finetuning keinen signifikanten Einfluss auf die Perfor-

mance des Classifiers nimmt. Einzig die Implementation mit 150 epochs zeigt leicht bessere Scores als die Standard-Vektoren, stellt sich im weiteren Verlauf der Arbeit allerdings als schlechter heraus, sodass letzten Endes die Finetuning-Modelle nicht weiterverwendet werden.

3.6 Weitere Features

Die SVM zeigt zu diesem Zeitpunkt bereits bessere Ergebnisse als eine Random Baseline, besitzt jedoch noch immer viel Luft nach oben. Weitere Features, die komplexere, schwerer zu erfassende Eigenschaften der Texte erfassen, könnten nützlich sein, um die Leistung noch zu verbessern.

3.6.1 Hatefulness Metrik

Eine Eigenschaft von *semantic embeddings* ist es, gewisse Tendenzen oder Neigungen natürlichsprachlicher Systeme zu lernen, replizieren und zu verstärken. So erfassen *embeddings* zum Beispiel die Konnotation von Begriffen, einem Geschlecht zugewiesen zu werden. Diese Effekte sind messbar und üblicherweise unerwünscht. Allerdings besteht die Möglichkeit, diesen Umstand zu nutzen, um eine „*Hatefulness*-Komponente“ in den Embeddings zu identifizieren, die bei der Klassifizierungsaufgabe nützlich sein kann.

Es gibt unterschiedliche Methoden, diesen semantischen *bias* zu klassifizieren und zu messen. Hier wird sich der Methode von Sweeney and Najafian (2019) in einer leichten Abwandlung bedient. Deren Modell, *Relative Negative Sentiment Bias* (RNSB) basiert auf der Annahme, dass in einem nicht-voreingestellten, neutralen System die Tendenz verschiedener – idealerweise wertneutraler – Bezeichnungsbegriffe (wie zum Beispiel Volksgruppen, Religionen), Ähnlichkeit zu negativ konnotierten Begriffen zu besitzen, einer gleichmäßigen Zufallsverteilung folgen sollte. Ist dies nicht der Fall, so weist das System einen positiven *bias* gegenüber manchen Bezeichnungen auf und einen negativen gegenüber anderen.

Um diesen *bias* zu messen, trainieren Sweeney and Najafian (2019) einen *logistic regression classifier*, welcher auf einem Korpus von positiven und negativen Wörtern trainiert wird und somit positiv und negativ konnotierte Wortvektoren diskriminieren kann. Der Classifier wird in deren Arbeit genutzt, um im weiteren Verlauf den RNSB-Score eines Systems zu ermitteln. Dies ist hier jedoch nicht von Interesse. Stattdessen ist der Score des Classifiers interessant. Dieser könnte nämlich verwendet werden, um eine Hatefulness-Komponente von Wörtern oder Sätzen zu ermitteln.

Dafür wird das Setup aus Sweeney and Najafian (2019) leicht verändert. Statt zwischen positiven und negativen Wörtern soll zwischen positiven und hasserfüllten Wörtern unterschieden werden. Dafür muss der Trainingskorpus jedoch angepasst werden, denn für gute Ergebnisse werden sehr viele Trainingswörter benötigt.

Hierfür wurde die API von The Weaponized Word[2] benutzt, um ein großes Lexikon an mit Hass assoziierten Wörtern zu laden. Dieses wurde minimal gefiltert – Begriffe, die aus mehr als einem Wort bestehen, wurden entfernt, da diese von Fasttext womöglich nicht gut erfasst werden – und dann anstelle von Negativbegriffen als Trainingslexikon für den *logistic regression classifier* verwendet. Die Implementation des Classifiers mit passenden Hyperparametern wurde aus Badilla et al. (2020) entnommen, die ihren Code frei zur Verfügung stellen.

Der Classifier stellt nun zwei Features für die SVM zur Verfügung: einmal die Hatefulness-Metrik für den Gesamtvektor eines Satzes und das maximal hasserfüllte Wort des Satzes.

3.6.2 Emotion Intensity analysis

Es erscheint selbstverständlich, dass Hass häufig im Zusammenhang mit *sentiment* und Emotionalität steht. VADER (Hutto and Gilbert, 2015) ist ein regelbasiertes System, das einen Text mit vier verschiedenen Scores bewertet.

Die `compound`-Metrik bewertet das *Sentiment* des Satzes insgesamt und reicht von sehr positiv bis sehr negativ. Drei weitere Scores bemessen die Intensität – beziehungsweise das Verhältnis – von positiven, negativen und neutralen Begriffen innerhalb des Satzes. Eine Aussage kann zum Beispiel mehrere sehr positive Begriffe beinhalten, aber insgesamt eine negative Bewertung darstellen.

Für die SVM erscheint es sinnvoll, alle vier Scores mit aufzunehmen, da sie *sentiment* insgesamt, aber auch die emotionale Intensität auf Wortebene erfassen. Besonders interessant für die SVM könnte hier der Fall sein, wenn das offensichtliche *sentiment* sich sehr stark von der emotionalen Intensität unterscheidet.

3.6.3 Features aus Davidson et al. (2017)

Der Classifier aus Davidson et al. (2017) ist mit einigen Features ausgestattet, die den hier verwen-

[2]Daten stammen von The Weaponized Word (https://weaponizedword.org/)

detet „Oberflächenfeatures“ sehr ähneln, zum Beispiel Anzahl an Wörtern oder durchschnittliche Silbenanzahl pro Wort.

Diese Feautres erscheinen auch für die vorliegende Aufgabe sinnvoll, und da sie bei Davidson et al. (2017) gute Ergebnise gezeigt haben, werden sie hier auch implementiert. Insgesamt sind dies 8 weitere numerische Features.

Modell	***acc.***	***prec.***	***rec.***
Random Baseline	.507	.354	.505
SVM einfach	.451	.673	.292
SVM optimiert	.732	.626	.549
Fasttext finetune	.737	.648	.537
SVM neue Features	.736	.642	.549
SVM final	.722	.605	.582

Tabelle 4: Scores der verschiedenen SVMs bis 4.

4 Finales Modell

Das Modell, wie es in 3.3 bereitstand, ist eine SVM mit optimierten Hyperparametern, trainiert auf Basis von den Daten aus ElSherief et al. (2021) mit einem Featureset bestehend aus Fasttext-*embeddings* sowie 12 weiteren Features.

Dieses Modell ist nun erweitert um 13 weitere Features und zwei der alten Features sind entfernt. Die domainspezifisch gefinetuneten *embeddings* wurden zugunsten der Standardvektoren verworfen. Wie in Tabelle 4 zu sehen ist, haben die neuen Features die Performance der SVM nur marginal gesteigert. Eine Analyse der Feature-Wichtigkeiten mithilfe eines neu trainierten Decision Trees zeigt, dass die meisten neuen Oberflächenfeatures aus Davidson et al. (2017) nicht im Entscheidungsprozess verwendet werden, einige wenige (die Anzahl der Zeichen und Wörter insgesamt) aber schon. Auch die Hatefulness-Metrik ist für das System nützlich. Dies ist interessant, da dieser Score – genau wie die meisten anderen nicht-*embeddings*-Features – nur einen geringen Varianzkoeffizienten aufweisen.

Eine Analyse auf Pearson's Korrelationskoeffizienten zeigt weiterhin auf, dass diese Features nützlich sein können. Auch hier scheint es eher der Fall zu sein, dass manche Dimensionen der Fasttext-Vektoren eine geringe Informativität aufweisen.

Um das Modell zu finalisieren, wurde ein letztes Hyperparameter-Finetuning betrieben. Die Scores in Tabelle 4 zeigen dafür eine schlechtere *accuracy* und *precision*, was daran liegt, dass weiterhin auf einen möglichst hohen *f1.5*-Wert optimiert wurde. Im finalen Modell haben sich *precision* und *recall* beinahe dem gleichen Wert angenähert.

5 Vergleich und Diskussion

Die fertige SVM kann nun genutzt werden, um sich der Forschungsfrage zu widmen. Geplant war es, den Classifier zu nutzen, um zwei *out-of-domain*-Datensätze zu klassifieren: Ein anderer Datensatz mit *implicit hate* – um Overfitting auf die Trainingsdaten zu prüfen – und einer mit *explicit hate*. Der zweite ist wesentlich interessanter. Zeigt das System nämlich eine gute Performance für expliziten Hass, obwohl es auf implizitem trainiert wurde, deutet es darauf hin, dass beide Probleme sehr ähnlich sind. Zeigt es hingegen schlechte Ergebnisse (und ist sichergestellt, dass kein zu starkes Overfitting stattgefunden hat), wäre es ein erstes, vorsichtiges Argument dafür, dass es sich um separat zu behandelnde Kategorien handelt.

Der Datensatz mit impliziten Daten ist der aus Caselli et al. (2020), welcher neu-annotierte Texte aus Zampieri et al. (2019) enthält. Der Datensatz enthält weitere Daten, welche für die Zwecke hier nicht relevant sind, aber problemlos herausgefiltert werden können. Problemhaft ist allerdings, dass die Daten nicht öffentlich zugänglich sind und eine Zugangsanfrage (bisher) nicht beantwortet wurde. Deswegen konnte dieser Vergleich leider nicht stattfinden. Ein anderer Datensatz mit guten impliziten Daten für eine Baseline scheint nicht zu existieren. Hierfür die gefilterten Posts von Kennedy et al. (2018) zu verwenden, ist ebenfalls keine gute Idee, da diese sich bereits für diese Zwecke als wenig kohärent gezeigt haben.

Interessanter ist ohnehin der Vergleich mit *explicit hate*. Der Datensatz von Davidson et al. (2017) unterscheidet drei Klassen, die hier auf zwei reduziert werden, damit wie immer nur *hate* und *non-hate* diskriminiert werden.

Tabelle 5 zeigt die Ergebnisse des Classifiers. Die SVM ist im Durchschnitt der Scores knapp 4% besser als die Random Baseline, was kaum signifikant ist.

5.1 Was sagen diese Werte aus?

Eine sichere Aussage kann hier nicht angestellt werden, da die implizite Benchmark weggefallen ist. Es ist also ein starkes Overfitting der SVM nicht ausgeschlossen. Sollte dies aber nicht der Fall sein, so weisen die miserablen Scores des Classi-

Modell	*acc.*	*prec.*	*rec.*
(Caselli et al., 2020)			
Random Baseline	-	-	-
SVM Modell	-	-	-
(Davidson et al., 2017)			
Random Baseline	.508	.259	.497
SVM Modell	.493	.267	.561

Tabelle 5: Scores der finalen SVM auf den *out-of-domain*-Daten

fiers darauf hin, dass nur eine geringe Ähnlichkeit zwischen den Phänomenen von impliziter und expliziter Hassrede existiert. Die Scores sind besser als eine Random-Baseline, jedoch nur marginal, sodass nicht ausgeschlossen ist, dass der Classifier kaum nützliches Wissen aus dem Training hier anwenden kann.

Insgesamt kann also kein sicherer Schluss gezogen werden. Die Datenlage suggeriert, dass implizite und explizite Hassrede tatsächlich kategorisch verschieden sind, doch diese Daten sind nicht sehr robust. Sollten robustere Daten für diese These erscheinen, würde dies bedeuten, dass es nützlich für Systeme zur automatischen Erkennung von Hassinhalten es sinnvoll wäre, die Aufgaben von *implicit* und *explicit hate* getrennt zu behandeln (und zum Beispiel zwei verschiedene Systeme für je eine Art von Inhalten zu trainieren).

Auch muss Vorsicht gewaltet werden, da die Definition für *hate speech* aus Davidson et al. (2017) eine andere ist als für die Daten, auf die die SVM trainiert wurde. Dadurch können auch Fehler durch Inkohärenz auftreten.

5.2 Weitere Forschung

Wie erwartet konnte die Hauptfrage der Arbeit nicht mit Sicherheit beantwortet werden. Die Fragestellung ist recht komplex und bedarf weiterer Forschung. Zukünftige Arbeiten in diese Richtung könnten zum Beispiel folgende Aspekte aufgreifen:

Für die Untersuchung des Unterschieds zwischen explizit und implizit hasserfüllten Inhalten sind mehr und bessere Daten notwendig. Aktuell verfügbare Korpora scheitern an der Trennschärfe zwischen diesen Kategorien, was ihre Informativität verringert. Eine bessere Definition des Begriffs von *implicit hate*, der eine genauere Eingrenzung des Phänomens ermöglicht, könnte dabei helfen, sauberere Daten zu erschaffen.

Die Frage nach Overfitting der finalen SVM konnte nicht beantwortet werden. Mit einem Vergleich des Systems auf anderen Datensätzen könnten die *out-of-domain*-Scores besser kontextualisiert und eingeschätzt werden.

Von der technischen Seite wurde die Performance der SVM im Laufe der Arbeit immer besser, doch es gibt noch Luft nach oben. Weitere Schritte könnten unternommen werden, um die Leistung des Systems zu verbessern. Bereits adressiert wurden die wenig informativen Dimensionen von Fasttext. Hier einen strengen Filter für die Feature-Selektion vorzusetzen (K-Best mit Spearson-r scheint vielversprechend), könnte den Noise verringern und für ein besseres Fitting sorgen.

Eine andere offene Frage ist die Erstellung des Satzvektors aus den Fasttext-*embeddings*. Bisher wurde lediglich der Durchschnitt aller Wortvektoren genommen, doch dies ist vermutlich nicht ideal. Eine andere Möglichkeit wäre es zum Beispiel, die Wortvektoren mit TF-IDF-Scores zu gewichten und dann zu vereinen. Etwas aufwendiger, aber potenziell interessant wäre ein System wie in Boom et al. (2016), welches die Fasttext-Vektoren nutzt, um eine semantische Repräsentation für kurze Texte zu erlernen.

Referenzen

Pablo Badilla, Felipe Bravo-Marquez, and Jorge Pérez. 2020. Wefe: The word embeddings fairness evaluation framework. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 430–436. International Joint Conferences on Artificial Intelligence Organization. Main track.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. 2016. Enriching word vectors with subword information. *CoRR*, abs/1607.04606.

Cedric De Boom, Steven Van Canneyt, Thomas Demeester, and Bart Dhoedt. 2016. Representation learning for very short texts using weighted word embedding aggregation. *CoRR*, abs/1607.00570.

Tommaso Caselli, Valerio Basile, Jelena Mitrović, Inga Kartoziya, and Michael Granitzer. 2020. I feel offended, don't be abusive! implicit/explicit messages in offensive and abusive language. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 6193–6202, Marseille, France. European Language Resources Association.

Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. 2017. Automated hate speech detection and the problem of offensive language. In

Proceedings of the 11th International AAAI Conference on Web and Social Media, ICWSM '17, pages 512–515.

Mai ElSherief, Caleb Ziems, David Muchlinski, Vaishnavi Anupindi, Jordyn Seybolt, Munmun De Choudhury, and Diyi Yang. 2021. Latent hatred: A benchmark for understanding implicit hate speech. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 345–363, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

C.J. Hutto and Eric Gilbert. 2015. Vader: A parsimonious rule-based model for sentiment analysis of social media text.

Brendan Kennedy, Mohammad Atari, Aida M Davani, Leigh Yeh, Ali Omrani, Yehsong Kim, Kris Coombs, Shreya Havaldar, Gwenyth Portillo-Wightman, Elaine Gonzalez, and et al. 2018. Introducing the gab hate corpus: Defining and applying hate-based rhetoric to social media posts at scale.

Quoc V. Le and Tomás Mikolov. 2014. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Chris Sweeney and Maryam Najafian. 2019. A transparent framework for evaluating unintended demographic bias in word embeddings. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1662–1667, Florence, Italy. Association for Computational Linguistics.

Bertie Vidgen and Taha Yasseri. 2020. Detecting weak and strong islamophobic hate speech on social media. *Journal of Information Technology & Politics*, 17(1):66–78.

Zeerak Waseem, Thomas Davidson, Dana Warmsley, and Ingmar Weber. 2017. Understanding abuse: A typology of abusive language detection subtasks. In *Proceedings of the First Workshop on Abusive Language Online*, pages 78–84, Vancouver, BC, Canada. Association for Computational Linguistics.

Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. 2019. SemEval-2019 task 6: Identifying and categorizing offensive language in social media (OffensEval). In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 75–86, Minneapolis, Minnesota, USA. Association for Computational Linguistics.