UNIVERSITÄT
LEIPZIG

Institut für Informatik
Fakultät für Mathematik und Informatik
Abteilung Datenbanken

Privatsphäre-erhaltende Sentiment Analyse auf Twitter

Bachelorarbeit

vorgelegt von:
Felix Vogel

Matrikelnummer:
3782744

Betreuer:
Prof. Dr. Erhard Rahm
Lucas Lange

Zusammenfassung

Das Ermitteln von Haltungen und Emotionen in den sozialen Medien ist unter anderem für Unternehmen vorteilhaft, um die allgemeine Meinung bezüglich eines Produktes beurteilen zu können. Dieser Prozess kann durch die Machine Learning (ML) Methoden aus dem Bereich der Sentiment Analyse erreicht werden. Da einige Posts auf Social Media private Informationen der Nutzer beinhalten und auch nach dem Löschen in externen Datensätzen gespeichert sein können, entwickeln wir das erste privatsphäre-erhaltende Modell für die Sentiment Analyse mittels Differential Privacy (DP) auf einem Twitter-Datensatz, um sensible Daten der Nutzer zu schützen.

Der Einsatz von DP führt in ML-Modellen allgemein zu einer reduzierten Genauigkeit. Die anfängliche Entwicklung eines nicht-privaten Modells, genannt Baseline, erlaubt uns daher den Vergleich der weiteren Nutzbarkeit unserer privaten Ergebnisse. Diese Baseline erreicht unter Berücksichtigung verschiedener Experimente eine maximale Genauigkeit von 81.89%. Die hier erreichte Genauigkeit bewegt sich somit im Bereich der Ergebnisse verwandter Arbeiten. In der Vorverarbeitung wird eine signifikante Verbesserung durch das Entfernen von Stoppwörtern erreicht. Eine neue Erkenntnis ist, dass das Transformieren von Wortkontraktionen in einem Unigramm-Modell einen nachteiligen Effekt hervorbringt.

Wir beurteilen die Nutzbarkeit des privatsphäre-erhaltenden Modells durch den Test verschiedener DP-Level mittels der verbleibenden Genauigkeit. Im Ergebnis erfahren unsere privaten DP-Modelle einen Genauigkeitsverlust zwischen 2.51% und 4.74% in Relation zur Baseline. Damit ist der in diesem Anwendungsfall resultierende Kompromiss zwischen verringerter Genauigkeit und erhöhter Privatsphäre gering und das entwickelte Modell unter dem Vorteil, sensible Trainingsdaten vor Extraktion durch beispielsweise Membership Inference Attacks zu schützen, zuverlässig einsetzbar.

Felix Vogel
3782744

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

BoW	bag-of-words
DP	Differential Privacy
DP-SGD	Differential Privacy Stochastic Gradient Descent
KNN	Künstliches Neuronales Netz
LR	Logistische Regression
MIA	Membership Inference Attack
ML	Machine Learning
MNB	Multinomialer Naive Bayes
NLP	Natural Language Processing
PPML	Privacy Preserving Machine Learning
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine

1. Einleitung

1.1. Motivation

In einer Welt, wo Kundenmeinungen über Produkte überwiegend durch Social Media Posts geteilt werden, ist es von großem Vorteil für Unternehmen, wenn diese Posts hinsichtlich ihrer Haltung zu einem Produkt analysiert werden können, wodurch unter anderem eine bessere Evaluation von Marketingstrategien erfolgen kann. Denn persönliche Produktbewertungen auf Social Media beeinflussen das Verhalten der Konsumenten in Bezug auf dieses Produkt. [1]

Durch die große Menge an Daten ist es nicht realistisch, diese von Hand zu klassifizieren, weshalb versucht wird, ein Verfahren zu entwickeln, welches dies automatisiert. Was jedoch für Menschen in den meisten Fällen nach kurzer Zeit erkennbar ist, stellt sich für eine Maschine als Herausforderung dar: das korrekte Ermitteln von Empfindungen innerhalb eines Textes. Dieser Prozess ist unter dem Namen der Sentiment Analyse bekannt [2]. Unter Einsetzung von Machine Learning (ML) wird versucht, einer Maschine die Festlegung dieser Grundstimmung zu vermitteln. Dabei wird einem Text das passende Sentiment zugeordnet, wie in Abbildung 1.1 dargestellt wird.

my whole body feels itchy and like its on fire	negativ
... 31 days until I leave for Ontario!!	positiv
this week is not going as i had hoped	negativ
@special72 Thanks, Special 72!	positiv

Abbildung 1.1.: Tweets (links) mit ihrem zugeordneten Sentiment (rechts) aus dem in dieser Arbeit verwendeten Datensatz.

Twitter ist ein soziales Netzwerk, bei dem Personen kurze Inhalte, wie beispielsweise Videos, Fotos und Texte, als sogenannte Tweets posten können. Viele dieser Nutzer veröffentlichen dabei private Informationen über sich und andere Personen [3]. Es ist möglich, Daten mittels der Twitter-API[1] zu sammeln und zu speichern, was dazu führt, dass selbst wenn sich ein Nutzer dazu entschließt, den Post aufgrund enthaltener sensibler Informationen zu löschen, dies nicht die vollständige Datenvernichtung impliziert, da eine Speicherung in einem externen Datensatz oder ML-Modell fortbestehen kann. Diese ML-Modelle bringen die Gefahr der Erkennung solcher Tweets durch einen Membership Inference Attack (MIA) [4] mit sich. Selbst nach Anonymisierung des Autors kann der Tweet Informationen wie Adressen, Alter oder Geschlecht anderer Personen enthalten. Diese könnten genügen, um diese zu identifizieren, was einen Verstoß gegen die Privatsphäre darstellt.

Beispielsweise veröffentlichte AOL im Jahr 2006 die Online-Suchanfragen von anonymisierten Nutzern [5]. Jones et al. [6] gelang es daraufhin, viele dieser Personen anhand der gegebenen Informationen in den Suchanfragen zu identifizieren.

Aus den genannten Gründen der Extraktion von Tweets aus ML-Modellen und der möglichen Identifikation der Nutzer über diese Texte, ist es wichtig, privatsphäre-erhaltende ML-Modelle zu entwickeln, die keine Rückschlüsse auf die verwendeten Trainingsdaten zulassen. Erfüllung dessen ist die Aufgabe des Privacy Preserving Machine Learnings (PPML) [7].

[1] https://developer.twitter.com/en/docs/twitter-api

1.2. Ziele

Ziel dieser wissenschaftlichen Arbeit soll es sein, einen privatsphäre-erhaltenden Ansatz für die Sentiment Analyse auf einem Twitter Datensatz zu entwickeln. Dafür soll zuerst eine grundlegende nicht-private Sentiment Analyse, also eine Baseline, implementiert werden. Um den Einfluss verschiedener Metriken und Vorverarbeitungsmethoden beurteilen zu können, sollen unterschiedliche Baseline-Modelle miteinander verglichen werden. Eines dieser Modelle soll dann in ein Modell auf Basis von Differential Privacy (DP) überführt werden. Am Ende der Arbeit soll die Nutzbarkeit des entstanden Modells unter Einfluss der Privatisierung beurteilt werden.

1.3. Struktur

Ein Verständnis über grundlegende verwendete Verfahren wird in Kapitel 2 vermittelt. Insbesondere wird auf die Funktionsweise der Sentiment Analyse, die Vorverarbeitung der Daten sowie privatsphäre-erhaltendes Maschinelles Lernen eingegangen. Darauf folgt in Kapitel 3 eine Aufschlüsselung verwandter Arbeiten, also diese, die auf dem Gebiet der Sentiment Analyse und der privatsphäre-erhaltenden Sentiment Analyse veröffentlicht wurden. In Kapitel 4 wird der Implementierungsablauf dargestellt. Zuerst wird Bezug auf die grundlegenden Methoden und die Hard- und Softwareumgebung genommen. Weiterhin wird der verwendete Datensatz sowie dessen Vorverarbeitung und Vektorisierung dargestellt. Danach folgt der Ablauf der Entwicklung der Baseline und der privaten Modelle mit dabei entstandenen Problemen und ihren Lösungen. Die Auswertung der resultierenden Ergebnisse wird in Kapitel 5 thematisiert, wo die Baseline-Modelle untereinander und mit verwandter Arbeit verglichen werden. Weiterhin erfolgt die Gegenüberstellung der privaten Modelle sowie der Vergleich von Baseline und privatem Modell. Kapitel 6 umfasst ein Fazit dieser Arbeit sowie mögliche Ansätze für zukünftige Arbeiten.

2. Hintergrund

2.1. Funktionsweise der Sentiment Analyse

Grundlage der Sentiment Analyse ist es, das Sentiment beziehungsweise die Stimmung aus einem Text zu extrahieren [8]. Um moderne künstliche intelligente Systeme weiterentwickeln zu können, ist es nötig, ihnen ein Verständnis über menschliche Emotionen zu vermitteln, denn diese spielen in der Beziehung zwischen Menschen eine wichtige Rolle. Wenn das Ziel ist, die Denkweise einer künstlichen Intelligenz der eines Menschen anzunähern, ist die Sentiment Analyse also notwendig, um dies zu erreichen.

Dies führte dazu, dass die Sentiment Analyse ein großer Bestandteil der Forschung auf dem Gebiet des Natural Language Processing (NLP) wurde. In dieser Arbeit beziehen wir uns auf den statistischen Ansatz, bei dem es zum Einsatz von ML-Verfahren kommt. Jedoch muss erwähnt werden, dass die Sentiment Analyse nicht von der Forschung über die Grundlage von Emotionen auf dem Teilgebiet der Psychologie getrennt werden kann. Wichtige Begriffe im Bereich des NLP sind *Dokument*, gleichbedeutend mit Text, und das *Korpus*, welches als eine Menge von Dokumenten definiert wird [9].

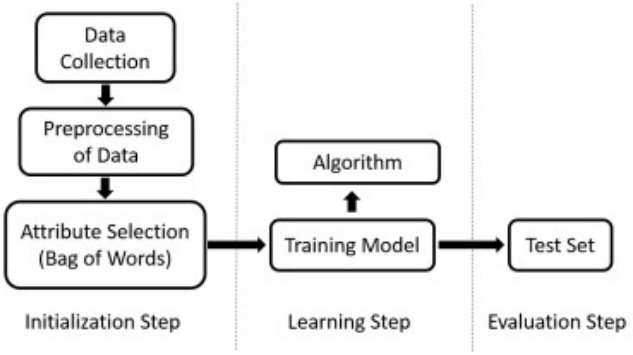


Abbildung 2.1.: Schritte der Sentiment Analyse. Abbildung aus [10] von Angiani et al.

Da der statistische Ansatz nur mit großen Datensätzen ausreichend gut funktioniert, muss erst eine solche Datenmenge gesammelt werden [8]. Der Ablauf besteht nun darin, dass zuerst Trainingsdaten von Hand oder mittels anderer Methoden klassifiziert werden, was bedeutet, dass den Texten ihr jeweiliges Sentiment zugeordnet wird. In Abbildung 2.1 wird ersichtlich, dass nun ein ML-Modell mittels dieser Daten trainiert werden kann, wobei Muster erkannt werden. Nach dem Trainingsprozess ist es in der Lage, Vorhersagen über das Sentiment neuer Texte zu treffen, welche dem Modell vorher unbekannt waren.

Somit gehört die Sentiment Analyse zum Teilgebiet des Supervised Learnings [11], bei dem Modelle mit einem Datensatz trainiert werden, wo die Zielvariable, in diesem Fall das Sentiment, bereits bekannt ist.

Vor diesem Trainingsprozess ist jedoch eine Vorverarbeitung des Textes notwendig. Außerdem muss er in eine Form gebracht werden, mit der das Modell lernen kann. Dabei müssen Attribute, soge-

nannte Features, aus dem Korpus extrahiert werden und in einem bag-of-words (BoW) Modell zusammengefasst werden [10].

Verschiedene Algorithmen existieren auf dem Gebiet des maschinellen Lernens, welche auch im Trainingsprozess der Sentiment Analyse ihren Einsatz finden. Darunter zählen Naive Bayes, Support Vector Machines (SVM), Logistische Regression (LR) sowie das künstliche neuronale Netz (KNN). [8]

Im letzten Schritt folgt eine Evaluierung des Modells, in dem Vorhersagen des Sentiments auf vorher unbekannten Texten getroffen werden. Dabei existieren verschiedene Metriken zur Bestimmung der Performance eines Modells, beispielsweise lässt sich die Vorhersagegenauigkeit aus dem Verhältnis der Anzahl korrekt klassifizierter Daten zu der Gesamtmenge aller Klassifizierungen berechnen.

2.2. Vorverarbeitung

Um das Korpus in eine Form zu bringen, mit welcher es möglich ist, ein ML-Modell zu trainieren, muss es einen Prozess durchlaufen, der im Folgenden erläutert wird.

2.2.1. Säuberung des Textes

Für das Säubern des Textes gibt es verschiedene Verfahren, die nacheinander durchgeführt werden. Sinn dieser Vorverarbeitung ist es, den Text in eine einheitliche Form zu bringen, um den Featureraum zu reduzieren. Es ist vorteilhaft für den Klassifikationsprozess, wenn semantisch gleiche Wörter mit unterschiedlicher Schreibweise einheitlich vorkommen. [10]

Zunächst werden störende Elemente, wie URLs, Benutzernamen und Hashtags aus dem Text entfernt. Tabulatoren und Zeilenumbrüche werden ebenfalls gelöscht, bevor Satzzeichen annulliert werden. Da auf Twitter formlose Sprache den Hauptteil der Tweets ausmacht, kommt es vor, dass zur besonderen Betonung eines Wortes, bestimmte Buchstaben mehr als nötig nacheinander verwendet werden. Dadurch entstehen unterschiedliche Varianten eines Wortes, welche jedoch die gleiche semantische Bedeutung haben (beispielsweise *cool* und *coooool*), weshalb es notwendig ist, diese zu normalisieren. Dabei werden die sich wiederholenden Buchstaben so gekürzt, dass Wortvarianten einheitlich betrachtet werden können. Falls Emoticons vorkommen, müssen diese in ihr Sentiment gruppiert und im Text durch einen Bezeichner (beispielsweise *happy_smiley*) ersetzt werden. Weiterhin werden alle Großbuchstaben in Kleinbuchstaben umgewandelt. [10]

Das Ignorieren von Wortnegationen (*don't, can't* etc.) ist einer der häufigsten Gründe von fehlerhafter Klassifikation [10]. Einerseits können diese mit einem Schlüsselwort, zum Beispiel *not* ersetzt werden. Weiterhin ist es möglich, diese Negationen in ihre Wortbestandteile aufzulösen, beispielsweise *don't* zu *do not*. Nach dem gleichen Prinzip werden allgemeine Wortkontraktionen, also das Zusammenziehen einzelner Wörter zu einem Wort, transformiert, wie zum Beispiel *I'm* zu *I am* [12].

Stemming bezeichnet das Entfernen von Suffixen eines Wortes, so dass ein Wortstamm entsteht [13]. Wörter mit dem gleichen Wortstamm werden nun zu einem Wort gruppiert und mit diesem

Repräsentanten ersetzt, wodurch der Featureraum weiter reduziert werden kann. [10]. Die verwandte Methode der Lemmatisierung ist eine Weiterentwicklung des Stemmings und thematisiert den Gruppierungsprozess der verschiedenen Formen eines Wortes zur Gewährleistung der einheitlichen Analyse dieser Wörter [13]. Der Unterschied dieser beiden Verfahren liegt darin, dass die Lemmatisierung zusätzlich die Bedeutung des Wortes betrachtet, wodurch Wortsteigerungen erkannt und transformiert werden können [13].

Weiterhin ist es möglich, inkorrekt geschriebene Wörter zu erkennen und zu korrigieren oder umgangssprachliche Ausdrücke in formale Ausdrücke zu transformieren [10]. Wörter, welche keine Bedeutung für den Gesamttext enthalten, werden Stoppwörter genannt und entfernt [14].

2.2.2. Vektorisierung des Textes

Das gesäuberte Korpus muss nun so repräsentiert werden, dass es ML-Algorithmen als Eingabe verwenden können. Da diese Algorithmen nicht mit rohem Text arbeiten können, muss dieser in numerische Werte oder Vektoren umgewandelt werden [15]. Dafür wird er in ein BoW-Modell umgewandelt.

Ein BoW-Modell wird genutzt, um Features aus dem Rohtext zu gewinnen. Um dieses Modell generieren zu können, sind Informationen über das gegebene Vokabular, also die vorkommenden Wörter, und über die Metrik, welche das Vorkommen dieser Wörter bewertet, notwendig. Dabei wird die Reihenfolge der Dokumente im Korpus vernachlässigt. [15]

Auf der einen Seite sind einzelne Wörter als kleinste linguistische Einheit geeignet, den Featureraum darzustellen. Jedoch bleibt hierbei der Kontext, der durch die Aneinanderreihung dieser Wörter entsteht, unbeachtet. [8]

Für ein Dokument werden nun alle vorkommenden Wörter bestimmt, welche das Vokabular definieren. Mit jedem betrachteten Dokument wird dieses erweitert. Ist das Vokabular vollständig, können alle Dokumente in Wortvektoren umgewandelt werden. Dabei werden die Wörter des Textes mit dem Gesamtvokabular abgeglichen und für jedes Wort, welches darin vorkommt, ein Wert im Vektor gesetzt. [15]

Abbildung 2.2 zeigt ein beispielhaftes BoW-Modell. Das Vokabular ist vordefiniert und besteht aus nur 4 Wörtern, auf die die beiden Dokumente auf der linken Seite abgebildet werden. Es entstehen zwei Wortvektoren auf der rechten Seite, die im BoW-Modell gespeichert werden und die Dokumente repräsentieren.

Für die Werte innerhalb der Wortvektoren gibt es verschiedene Möglichkeiten, die sich durch die genutzte Metrik auszeichnen [15]. Diese beinhalten die Existenz im Text, wo für jedes Wort der Wert 0 (existiert nicht) oder 1 (existiert) gesetzt wird. Weiterhin wird der Häufigkeitswert angeführt, wo das Vorkommen des Wortes quantifiziert wird.

Innerhalb des BoW-Modells kann es vorkommen, dass Wörter, obwohl sie häufig vorkommen, im Vergleich zu selteneren Wörtern keinen höheren Informationsgehalt besitzen. Durch Normalisieren des Häufigkeitswertes kann diese Problematik umgangen werden. [15]

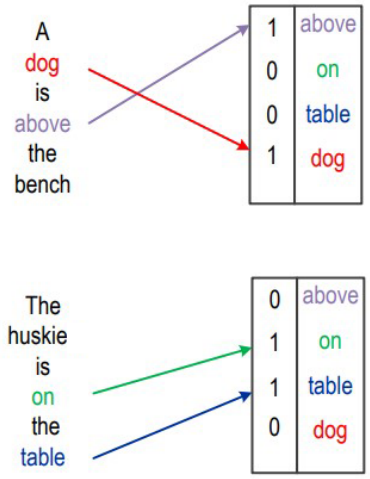


Abbildung 2.2.: BoW für zwei Beispieldokumente. Abbildung (bearbeitet) aus [16] von Zhao et al.

Diese Metrik nennt sich *Term Frequency – Inverse Document Frequency*, kurz TF-IDF. Die Funktionsweise wird durch die folgenden Definitionen ersichtlich: [17, 18]

$$tf(t,d) = \frac{\text{Anzahl von Wort } t \text{ in Dokument } d}{\text{Wortanzahl in Dokument } d} \tag{2.1}$$

$$df(t) = \text{Anzahl von Dokumenten } d\text{, so dass gilt: } t \in d \tag{2.2}$$

$$idf(t,D) = ln(\frac{N_D}{df(t)}) \text{ mit } N_D = \text{Gesamtanzahl aller Dokumente} \tag{2.3}$$

$$tf - idf(t,d,D) = tf(t,d) \cdot idf(t,D) \tag{2.4}$$

Der Häufigkeitswert des Wortes im betrachteten Dokument wird also zu dem Häufigkeitswert in allen Dokumenten ins Verhältnis gesetzt. Dabei gibt *idf(t,D)* die Seltenheit des Wortes im kompletten Textkorpus an [15]. Je seltener ein Wort ist, desto höher ist sein idf-Wert. Wörter mit einem hohen tf-idf-Wert, der durch einen hohen idf-Wert enstehen kann, sind also für das betrachtete Dokument sehr aussagekräftig. Dieser Logik folgend, haben Wörter, die in dem gesamten Korpus häufig vorkommen und nicht übermäßig im aktuellen Dokument erscheinen, also insgesamt einen niedrigen tf-idf-Wert haben, eine geringe Bedeutung beziehungsweise Aussagekraft für das betrachtete Dokument.

Um den Featureraum weiter zu minimieren, existiert die Option, Wörter zu gruppieren. Dieser Ansatz basiert auf sogenannten n-Grammen [9]. Jedes Wort ist ein Gramm und n gibt an, wie viele Wörter gruppiert werden. Beispielsweise werden bei einem 2-Gramm-Modell beziehungsweise Bigramm-Modell jeweils zwei Wörter miteinander gruppiert.

Nun wird auch klar, weshalb es unter anderem wichtig ist, den Text zu säubern. Falls dies nicht geschieht, enthält das Vokabular sehr viele Wörter und das entstehende BoW-Modell ist mit einem hohen Speicheraufwand verbunden. Die Grenzen eines BoW-Modells liegen also in der Handhabung der Vokabulargröße und darin, dass die Vernachlässigung der Reihenfolge der Wörter innerhalb eines Textes unvermeidlich zur Ignorierung des Kontexts führt [15].

2.3. Logistische Regression als künstliches neuronales Netz

Die LR als Algorithmus kann durch ein entsprechendes KNN beschrieben werden. Der Aufbau ist in Abbildung 2.3 dargestellt und besteht darin, dass keine verborgenen Schichten existieren und insgesamt ein einziges Neuron vorhanden ist, welches für die Ausgabe verantwortlich ist [19].

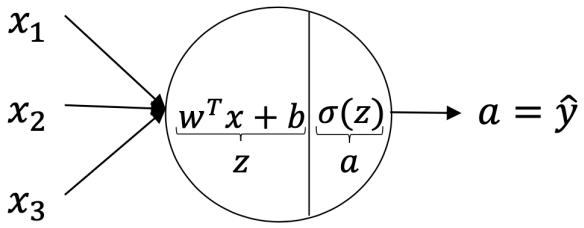


Abbildung 2.3.: Aufbau des KNN für die LR. Abbildung aus [19] von Liquet et al.

Mit w als Gewichtsvektor, x als Eingabevektor und b als Bias ist die Summierungsfunktion z in Ausdruck 2.5 [19] definiert.

$$z = w^T x + b = w_1 x_1 + ... + w_\mathrm{n} x_\mathrm{n} + b \tag{2.5}$$

Dabei entspricht die Aktivierungsfunktion des Ausgabeneurons der sigmoiden Funktion in Ausdruck 2.6, wie sie auch bei der LR eingesetzt wird. Diese gibt für ein beliebiges z einen Wert im Intervall (0, 1) zurück, welches die Wahrscheinlichkeit angibt, dass z zu der Klasse 1 gehört. Somit lässt sich der Ausgabewert des KNN durch den Ausdruck 2.7 ermitteln.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.6}$$

$$\hat{y} = P(y = 1 \mid x) = \sigma(w^T x + b) \tag{2.7}$$

Der Wert $\hat{y}$ muss folglich auf eine Klasse abgebildet werden, wobei ein Schwellenwert genutzt wird. Bei dessen Über- beziehungsweise Unterschreitung erfolgt die Zuordnung zur Klasse. Beim Trainingsprozess wird nun das Ziel verfolgt, die Gewichte und den Bias so anzupassen, dass $\hat{y}$ sich bestmöglich dem tatsächlichen Wert y annähert. Um diese Anpassung vornehmen zu können, muss die Entfernung des vorhergesagten vom tatsächlichen Wert berechnet werden, wobei es zur Nutzung einer Kosten- oder Verlustfunktion kommt. Für die LR existiert dafür die Cross-Entropy Kostenfunktion L für ein Trainingsexemplar, aus welcher sich die Verlustfunktion als Durchschnittswert aller n Exemplare für den gesamten Trainingsdatensatz mit J ergibt [19]:

$$L(\hat{y}, y) = -(y \cdot log(\hat{y}) + (1 - y) \cdot log(1 - \hat{y})) \tag{2.8}$$

$$J(w) = \frac{1}{n} \sum_{i=1}^{n} L(\hat{y}_\mathrm{i}, y_\mathrm{i}) \tag{2.9}$$

Um die Gewichte unter Berücksichtigung des Fehlers anzupassen, kommt der Stochastic Gradient Descent Algorithmus (SGD) [20] zum Einsatz. Dieser findet iterativ die Gewichte zur Minimierung von J. Durch die Minimierung der Verlustfunktion wird also der Abstand zwischen vorhergesagtem und tatsächlichem Wert minimiert, was zu einem genaueren Modell führt. Mit jeder Iteration werden die Gewichte nacheinander wiederholt über den Gradienten unter Einbeziehung des Fehlers und einer Lernrate α, nach dem in Ausdruck 2.10 [19] gegebenem Schema, angepasst. Der Algorithmus wird in Abbildung 2.4 veranschaulicht.

$$w_{t+1} = w_t - \alpha \frac{\partial J(w_t)}{\partial w_t} = w_t - \alpha((\hat{y}_i - y_i)x_i) \tag{2.10}$$

Dabei gibt die Lernrate die Größe der Schritte an, mit denen die Näherung an das Minimum erfolgt. Sobald der Gradient annähernd den Wert 0 erreicht, terminiert der Algorithmus, denn das Minimum der Kostenfunktion J wurde erreicht. Da die Verlustfunktion bei der LR konvex ist, ist SGD in der Lage, das globale Minimum zu finden. Der Unterschied zum allgemeinen Gradient Descent besteht darin, dass beim SGD der Gradient für kleine Datenstücke (Batches) bestimmt werden muss und nicht bei jeder Iteration über alle Trainingsexemplare, was sich vor allem für große Datensätze durch die Reduzierung der Berechnungszeit eignet. [19]

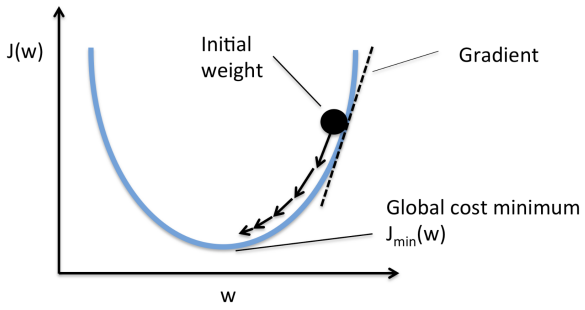


Abbildung 2.4.: Veranschaulichung des SGD Algorithmus. Abbildung aus [21] von Rachka.

Durch Backpropagation bei dem KNN, bei welchem die Fehler rückwärts geleitet werden und über die Berechnung der Gradienten zur Gewichtsanpassung führen, wird gewährleistet, dass die Eingabedaten des Trainingsdatensatzes mit jeder Lerniteration genauer auf die Ausgabeklassen abgebildet werden [19].

Durch die Verwendung von $\sigma(z)$ als Aktivierungsfunktion und der Minimierung der Cross Entropy Verlustfunktion durch SGD vereint das vorgestellte KNN die Funktionsweise der LR sowie die Implementierungsvorteile eines KNN, die dadurch entstehen, dass bekannte ML-Bibliotheken, unter anderem TensorFlow und Keras, auf die Entwicklung dieser ausgelegt sind.

2.4. Privatsphäre-erhaltendes Machinelles Lernen

2.4.1. Notwendigkeit und Nutzen

Jedes ML-Modell passt seine Parameter beim Lernprozess an den Trainingsdatensatz an, so dass die Eingabedaten bestmöglich auf die zugehörigen Klassen abgebildet werden. Nutzt man diese Kenntnis, ist es möglich, zu bestimmen, ob ein gewisser Datenpunkt dem verwendeten Trainingsdatensatz angehört. Diese Eigenschaft wird Membership Inference [4] genannt. Denn Tatsache ist, dass Modelle eine höhere Vorhersagegenauigkeit haben, wenn sie auf den Trainingsdaten evaluiert werden, als auf Testdaten, die vorher unbekannt waren [4]. Dieser Aspekt wird bei einem MIA ausgenutzt, indem über den ausgegebenen Konfidenzwert des Modells bei einem bestimmten Datenpunkt, also der Wert, der angibt, mit welcher Wahrscheinlichkeit der Datenpunkt zu der vorhergesagten Klasse gehört, darauf geschlossen wird, ob dieser im Trainingsdatensatz enthalten war. Durch Testen des Modells mit verschiedenen Daten und gleichzeitiger Beobachtung der resultierenden Konfidenzwerte, kann festgestellt werden, ob der getestete Datenpunkt im Trainingsdatensatz vorkam [4].

Problematisch ist dies vor allem, wenn es um Datensätze geht, wo bereits die Information der Zugehörigkeit eine Verletzung der Privatsphäre darstellt. Falls es beispielsweise medizinische Studien gibt, wo der Trainingsdatensatz nur Personen enthält, die an einer bestimmten Krankheit leiden und dies im Trainieren von ML-Modellen resultiert, kann über MIAs durch die Feststellung der Zugehörigkeit der Person zum Datensatz, die Information gewonnen werden, dass diese mit einer hohen Wahrscheinlichkeit an der Krankheit leidet [22].

Aber auch in anderen Datensätzen kann dies zum Problem werden. Shokri et al. [4] stellten einen Ansatz vor, bei dem zufällige Datenpunkte generiert wurden. Diese wurden an das ML-Modell übergeben und anhand der Konfidenzwerte festgestellt, ob der generierte Datenpunkt tatsächlich existiert und im Trainingsdatensatz enthalten ist, was dann der Fall ist, wenn ein sehr hoher Konfidenzwert resultiert. Somit können tatsächliche Datenpunkte aus dem Trainingsdatensatz generiert werden.

Bezogen auf den Twitter-Datensatz bedeutet dies, dass es theoretisch möglich ist, über MIAs Texte der Tweets aus dem Trainingsdatensatz, welche sensible Informationen enthalten können, zu gewinnen. Solche Tweets können unter anderem Daten über Urlaubsaufenthalte, Alkohol und Krankheiten enthalten [3]. Erhält ein Angreifer solche Informationen, entsteht dadurch eine Verletzung der Privatsphäre des Nutzers. Denn trotz der Tatsache, dass die Nutzernamen im Datensatz zum Training des Modells zur Sentiment Analyse nicht enthalten sind, sondern nur die Polarität sowie der Text, ist es dennoch möglich, einen Menschen allein über den Text zu identifizieren. Durch Angabe von Namen, Standorten und anderen persönlichen Informationen im Tweet können Verfahren wie Record Linkage [23] eingesetzt werden, bei dem Einträge über verschiedene Datensätze miteinander verknüpft werden, und somit eine Identifikation ermöglichen.

Eine große Problematik besteht auch darin, dass sich nicht jeder Nutzer auf Twitter bewusst ist, dass auch nach Entfernen des Tweets, die Daten trotzdem bei Dritten gespeichert sein können. Das in dieser Arbeit entstehende ML-Modell sowie der verwendete Datensatz ist dafür das beste Beispiel. Twitter bietet zwar an, Tweets wieder zu löschen, jedoch impliziert dies nicht die vollständige

Vernichtung dieser Daten. Wenn sich also ein Nutzer dazu entschließt, einen Tweet zu entfernen, weil ihm die darin veröffentlichten Informationen nachträglich zu sehr in die Privatsphäre eingreifen, ist es möglich, dass diese ohne das Wissen des Nutzers an einem anderen Ort gespeichert bleiben. Werden diese sensiblen Daten für einen Angreifer erhältlich, und führen sie zu einer Identifizierung der Person, stellt dies ein Privatsphäreproblem dar. Um dies zu vermeiden, kommt PPML zum Einsatz.

2.4.2. Differential Privacy

Um solche Privatsphäreprobleme bei ML-Modellen zu verhindern, ist das Konzept der DP [24] entstanden. Hier liegt der Ansatz zu Grunde, dass die Ausgabe einer Anfrage an einen Datensatz annähernd übereinstimmt, unabhängig davon, ob ein individueller Datenpunkt enthalten ist oder nicht. Somit schützt DP die Personen, deren Daten einem Datensatz angehören vor dem Schaden, der nicht entstehen würde, wenn sie nicht Teil dieser Datenmenge wären. Das resultierende Privatsphärerisiko soll also, frei von der Zugehörigkeit zu einem Datensatz, gleich sein.

Die Definition der DP wird durch den Ausdruck 2.11 deutlich [24].

$$Pr[M(x) \in S] \leq e^{\epsilon} \cdot Pr[M(y) \in S] + \delta \tag{2.11}$$

Dabei sind x und y zwei Datensätze, die sich in maximal einem Datenpunkt unterscheiden. M ist ein Algorithmus, der auf den Datensatz angewandt wird. $Pr[M(x) \in S]$ gibt die Wahrscheinlichkeit an, dass die Ausgabe von M in einer Menge S liegt. Wenn der Ausdruck für alle x und y sowie für alle S gilt, gewährleistet der Algorithmus M die DP mit dem Parametertupel (ε, δ). Dies wird (ε, δ) - DP genannt.

Dabei gibt ε das Privatsphärebudget an, entspricht also einer Quantifizierung des erlaubten Unterschieds der Wahrscheinlichkeit der Ausgabe, wenn ein einziger Datenpunkt hinzugefügt oder entfernt wird. Ersichtlich wird, je höher ε ist, desto mehr Unterschied zwischen den Ausgabewahrscheinlichkeiten wird erlaubt, was bedeutet, dass die Privatisierung abnimmt. Umgekehrt bedeutet dies, je geringer ε, desto höher die Privatisierung.

Der Parameter δ bezieht sich auf das grundlegende Risiko, dass es zu einem Privatsphäreproblem kommt, während die Faustregel besagt, dass $\delta < \frac{1}{N}$ gelten sollte, wobei sich N auf die Anzahl der Datenpunkte bezieht [24].

Somit ist gewährleistet, dass die Wahrscheinlichkeit der Ausgabe von M auf einem Datensatz ähnlich ist, unabhängig davon, ob dieser Datensatz einen individuellen Eintrag enthält oder nicht. Somit bietet die DP einen Schutz gegen den MIA, welcher genau diese unterschiedlichen Ausgaben ausnutzen würde [4]. Im Gegensatz zur Anonymisierung von Datensätzen werden durch DP ebenfalls Record Linkage Attacken unterbunden [24].

Um DP im Bereich des ML anzuwenden, gibt es verschiedene Methoden. Der Grundgedanke ist, ein Rauschen, welches aus statistischen Verteilungen, beispielsweise Laplace- oder Gauß-Verteilung, abgeleitet wird, hinzuzufügen. Dies geschieht zu unterschiedlichen Zeitpunkten bei der Entwicklung

eines Modells. DP kann auf die Eingabedaten, auf den Optimisierungsalgorithmus beim Training, die Kostenberechnung oder die resultierenden Parameter angewandt werden. [25]

Ein Beispiel für eine DP-Methode, die während des Trainingsprozesses hinzugefügt wird, ist der Differential Privacy Stochastic Gradient Descent Algorithmus (DP-SGD). Dieser erweitert den Optimisierungsalgorithmus SGD um DP, wobei die Gradienten mittels ihrer L2 Norm beschnitten werden sowie ein gaußsches Rauschen hinzugefügt wird, um eine Privatisierung des Modells sicherzustellen [25].

Durch das Hinzufügen des Rauschens sinkt die Genauigkeit des Modells, es steigt jedoch der Privatisierungsgrad [25]. Somit entsteht ein Kompromiss zwischen der Privatsphäre und der Genauigkeit, wo abgeschätzt werden muss, wie viel Privatisierung ohne den Verlust der Anwendbarkeit des Modells möglich ist.

3. Verwandte Arbeiten

Als Vorbild für die Implementierung der Baseline wird auf die Arbeit von Go et al. [26] zurückgegriffen, da dort der genutzte Datensatz generiert wird und erste Vorverarbeitungstechniken vorgestellt werden. Die ausgewerteten Klassifikationsalgorithmen beinhalten Naive Bayes, SVM, Maximum Entropy sowie ein Verfahren, bei dem Schlüsselwörter genutzt werden, um eine Vorhersage zu treffen. Unter der Verwendung von Unigrammen und der Voraussetzung, dass Nutzernamen und URLs entfernt sowie sich wiederholende Buchstaben transformiert werden, erreicht Naive Bayes eine Genauigkeit von 81.3%, der Schlüsselwort-Klassifikator 65.2%, Maximum Entropy 80.5% und SVM 82.2%. Zusätzlich zu den von Go et al. genannten Vorverarbeitungsverfahren, werden in dieser Bachelorarbeit andere Techniken getestet, so dass eine Auswertung des Einflusses des Vorverarbeitungsprozesses auf die Performance des Modells gewährleistet wird.

In verwandten Arbeiten bezüglich der Vorverarbeitung bei der Sentiment Analyse wird meist der gemeinsame Einfluss der genutzten Methoden betrachtet [27, 14, 28, 29]. Alam et al. [27] schlagen vor, dass in zukünftigen Arbeiten der Effekt jeder einzelnen Vorverarbeitungsmethode getrennt ausgewertet werden sollte, was in dieser Bachelorarbeit getan wird.

Einen Ansatz zur Privatisierung der Sentiment Analyse liefert die Arbeit von Alatrista-Salas et al. [30], in der durch die Repräsentation von Textdokumenten mittels Bloomfiltern ein privatsphäreerhaltendes Modell geschaffen werden soll. Vor dem Trainingsprozess erhält jedes Wort im BoW-Modell eine Position im Bloomfilter, welche durch Nutzung von Hashfunktionen ermittelt wird. Resultierend wird die Privatsphäre der Nutzerdaten gewahrt, ohne dabei einen großen Performanceverlust zu verzeichnen. Dieser Vorgang unterscheidet sich also grundlegend von DP, da kein statistisches Rauschen hinzugefügt wird.

Dass jedoch die Nutzung von Bloomfiltern zur Wahrung der Privatsphäre nicht verlässlich ist, belegt die Arbeit von Christen et al. [31]. In dieser wurde festgestellt, dass es durch gezielte Kryptoanalyse-Attacken möglich ist, die sensiblen Informationen, kodiert durch die Bloomfilter, zu re-identifizieren. Mittels der Zusammenführung von häufig vorkommenden Bitpositionen in den Bloomfiltern können Klartextwerte deanonymisiert werden. Weiterhin wird keine Privatisierung des resultierenden Modells, sondern nur des Trainingsprozesses, gewährleistet, weshalb Angriffe auf das Modell durch diesen Ansatz nicht vermindert werden.

Aufbauend auf dieser Arbeit entstand eine Publikation [32], die beweist, dass solche Angriffe auf Bloomfilter durch den Einsatz von DP verhindert werden können. Ranbaduge et al. [32] stellten fest, dass durch die Veränderung von Bits innerhalb der Bloomfilter, gemäß dem Rauschen der DP, Klartextwerte nicht mehr verlässlich bestimmt werden können, wenn der Angriff von Christen et al. durchgeführt wird. Es wird ebenfalls angenommen, dass DP MIAs auf das resultierende Modell vermindern kann [4]. Dies zeigt, dass die DP den Bloomfiltern in Bezug auf die Sicherheit der Privatisierung je nach Anwendungsfall überlegen sein kann, was auch den Grund darstellt, weshalb DP in dieser Bachelorarbeit als Privatisierungstechnik genutzt wird.

4. Sentiment Analyse auf dem Datensatz

4.1. Methoden

Dieser Abschnitt beschreibt die Methoden zur Generierung und Auswertung der Ergebnisse.

Der verwendete Datensatz mit 1.6 Millionen Tweets wurde mittels Distant Supervision generiert [26]. Um den Einfluss der Textvorverarbeitung auf die Genauigkeit der Sentiment Analyse auszuwerten, werden 5 Versuchsreihen, die jeweils unterschiedliche Vorverarbeitungsmethoden beinhalten, auf jedes Dokument des Datensatzes angewandt.

Bei der Anfertigung der BoW-Modelle werden drei verschiedene Metriken genutzt, um die Auswirkung dieser auswerten zu können: der absolute Häufigkeitswert, TF-IDF sowie der Wahrheitswert bezüglich der Existenz des Wortes. Für jede Kombination von Vorverarbeitungsreihe und Metrik werden nun drei verschiedene Algorithmen für das ML-Modell verwendet: Multinomialer Naive Bayes (MNB), SVM sowie die LR. Somit entstehen 45 verschiedene Kombinationen, welche als Grundlage für die Modelle dienen, die die Baseline darstellen. Jedes Modell entsteht durch eine mögliche Kombination aus Vorverarbeitungsreihe, Metrik und Algorithmus. Die entstehenden Modelle werden untereinander hinsichtlich ihrer Genauigkeit auf den Testdaten verglichen.

Um das privatsphäre-erhaltende Modell zu entwickeln, wird DP implementiert. Da die LR den SGD-Algorithmus zum Optimieren der Verlustfunktion verwendet, wird das LR-Modell mit der besten Performance unter Nutzung von DP-SGD zu einem DP-Modell weiterentwickelt. Dabei werden verschiedene ε-Werte verwendet (0, 0.1, 1 und 10). Die Modelle, die unter Einsatz dieser Werte enstehen, werden untereinander sowie mit der Baseline verglichen. Diese Gegenüberstellung erfolgt ebenfalls mittels der Vorhersagegenauigkeit auf dem Testdatensatz.

4.2. Technologien

Dieser Teil bezieht sich auf den verwendeten Computer sowie die grundlegenden Bibliotheken, die für alle Experimente und Analysen genutzt wurden.

Der in dieser Arbeit vorrangig verwendete Computer ist mit 16 GB RAM und einer NVIDIA GeForce RTX 2070 ausgestattet. Um komplexere Berechnungen, vorrangig in Bezug auf das LR-Modell, parallel ausführen zu können, wurden für diese Arbeit zusätzlich Ressourcen des Universitätsrechenzentrums Leipzig genutzt. Die hierbei genutzte Maschine verfügt über 32 GB RAM und einer NVIDIA Tesla V100 GPU.

Der gesamte Quellcode zum Nachvollziehen der Ergebnisse befindet sich in einem öffentlichen Github Repository[2] in Form von Jupyter Notebooks[3] mit einem entsprechenden Leitfaden, der das Ausführen gewährleistet. Als grundlegende Programmiersprache in dieser Arbeit kommt Python[4]

[2] https://github.com/felix2246/dp-sent-analysis-twitter
[3] https://jupyter.org/
[4] https://www.python.org/

zum Einsatz. Scikit-learn[5] ist eine Bibliothek und stellt bekannte Algorithmen für maschinelles Lernen in Python bereit [33]. In dieser Arbeit werden Naive Bayes sowie SVM mittels scikit-learn implementiert.

Für die Implementierung der LR kommt die Bibliothek TensorFlow[6] zum Einsatz [34]. Dies hat den Hauptgrund, dass für TensorFlow eine Erweiterung für den Einsatz von DP entwickelt wurde. Zusätzlich kommt es zur Verwendung von Keras[7], welches auf TensorFlow aufbaut. Die Bibliothek, mit der es möglich ist, den Optimizer der Baseline so zu verändern, dass DP garantiert werden kann, nennt sich TensorFlow Privacy[8] [35]. Hiermit ist es ebenfalls möglich, die Metriken der Privatsphäre zu berechnen.

Um reproduzierbare Ergebnisse beim Vorverarbeiten der Daten und beim Trainingsprozess der Modelle zu gewährleisten, wird mittels einer Seedfunktion ein fester Seed mit dem Wert 42 festgelegt. Dieser gewährleistet, dass beim mehrmaligen Ausführen des Quelltextes dieselben Zufallszahlen generiert werden. Die Zufallszahlen, die beim Training enstehen sowie verwendet werden, sind also bei jeder Ausführung gleich und es kommt bei den gleichen Eingabedaten zum gleichen resultierenden Modell. Folglich wird ein deterministischer Prozess gewährleistet, welcher sich für die spätere Auswertung und den Vergleich verschiedener Modelle sowie verschiedener Vorverarbeitungsmethoden eignet.

4.3. Datensatz

Der in dieser Arbeit verwendete Datensatz wurde im Rahmen einer wissenschaftlichen Publikation an der Universität in Stanford von Go et al. [26] generiert und ist frei zum Download[9] verfügbar. Er besteht aus einem Trainings- und Testdatensatz. Insgesamt enthält der Trainingsdatensatz 1.600.000 Datenpunkte und der Testdatensatz 497 Datenpunkte, wobei jeder die folgenden Attribute aufweist [36]:

- *Polarität*: Sentiment des Tweets: 0 für negativ, 2 für neutral und 4 für positiv. Repräsentiert die Klasse.
- *ID des Tweets*
- *Zeitstempel des Tweets*
- *Query* : Hilfsmittel zum Suchen und Auswählen von Tweets über die Twitter API [37].
- *Benutzername des Autors*
- *Inhalt des Tweets*

Die Tweets wurden mittels der Twitter-API gesammelt [26]. Mit dieser Schnittstelle ist es möglich, Tweets auf programmatischem Wege per HTTP-Request abzurufen [38].

[5] https://scikit-learn.org/stable/
[6] https://www.tensorflow.org/
[7] https://keras.io/
[8] https://github.com/tensorflow/privacy
[9] http://help.sentiment140.com/for-students

4.3.1. Trainingsdaten

Um den Trainingsdatensatz zu generieren, wurde ein Verfahren namens Distant Supervision verwendet [26]. Dabei wurden alle Tweets, die ein positives Emoticon enthalten, als positiv markiert. Alle Tweets, die ein negatives Emoticon beinhalten, wurden als negativ klassifiziert. Also wird anhand dessen, ob ein bestimmtes Emoticon enthalten ist, das Sentiment des Tweets vorhergesagt. Dies spart Zeit im Vergleich zu dem Ansatz, bei dem Trainingsdaten von Hand klassifiziert werden müssen. Somit ist es möglich, einen solch großen Datensatz zu generieren. In diesem Fall wurde darauf geachtet, nur englische Tweets abzurufen, was die Limitierung mit sich bringt, dass die Sentiment Analyse nur mit Texten dieser Sprache korrekt funktionieren wird.

Der bereitgestellte Trainingsdatensatz wurde bereits in einigen Punkten von Go et al. für die Sentiment Analyse vorbereitet. Zuerst wurden ausgewählte Emoticons aus dem Text entfernt. Das hat den Grund, dass die Klassifikatoren sonst einen zu großen Wert auf diese legen, worunter die Genauigkeit leiden würde [26]. Wenn ein Tweet das Emoticon ":P" enthält, wurde der gesamte Tweet entfernt.

Die restlichen in der verwandten Arbeit genannten Maßnahmen wurden jedoch nur unvollständig an dem zum Download bereitgestellten Datensatz durchgeführt. Dazu zählen das Löschen von Tweets mit gegensätzlichen Emoticons, sowie das Entfernen von Retweets und doppelten Tweets. 160 Retweets existieren noch im Trainingsdatensatz, welche entfernt werden. Dies hat den Zweck der Vermeidung von zusätzlicher Gewichtung eines Tweets beim Trainingsprozess [26]. Des weiteren existieren zwei Tweets, welche Datenpunkte mit unterschiedlichen Emoticons enthalten. Diese werden nach der Idee von Go et al. ebenfalls annulliert.

Die Polarität ist als numerisches Attribut vorhanden und hat entweder den Wert 0 (negativ) oder 4 (positiv). Neutrale Tweets sind im Trainingsdatensatz also nicht enthalten. Insgesamt besitzen 800.000 Datenpunkte die Polarität 0 und ebenfalls 800.000 Datenpunkte die Polarität 4. Wie Kaur et al. [39] festgestellt haben, ist es für eine optimale Genauigkeit eines Klassifikators wichtig, dass die einzelnen Klassen gleichmäßig verteilt sind, was hier der Fall ist.

Anhand der Verteilung der Tweet-ID lässt sich feststellen, ob doppelte Datenpunkte vorhanden sind. Insgesamt kommen 1.685 IDs exakt zweimal vor. Hier reicht es jedoch nicht, je ein Duplikat zu löschen, da alle Duplikatpaare jeweils unterschiedliche Polaritäten aufweisen. Dies wurde anhand des Mittelwertes der Klasse innerhalb eines Duplikatpaares geprüft. Wenn dieser ungleich 0 oder ungleich 4 ist, haben die geprüften zwei Tweets mit gleicher Tweet-ID eine unterschiedliche Polarität, was einer widersprüchlichen Information für den Klassifikator entspricht. Weil das bei allen Duplikatpaaren der Fall war, wurden alle 3.370 Tweets entfernt.

Alle Trainingsdatenpunkte wurden im Zeitraum von April bis Juni 2009 gepostet und haben beim Attribut Query den Wert "NO_QUERY", was bedeutet, dass keine Query verwendet wurde, um möglichst alle Kategorien abzudecken.

Der Inhalt jedes Tweets sollte einzigartig sein, damit der Klassifikator kein extra Gewicht auf sich wiederholende Datenpunkte legt [26]. Nach dem Löschen der Tweets mit gleicher Tweet-ID bestehen dennoch 23.592 Datenpunkte, deren Texte diese Bedingung verletzen. Dabei existieren 6.758 Gruppen, also Inhalte, die mehr als einmal vorkommen. Innerhalb von 547 Gruppen traten Tweets

mit unterschiedlicher Polarität auf, was eine widersprüchliche Information für den Klassifikator darstellen würde, denn der gleiche Text wurde hier mit verschiedener Polarität markiert. Es werden alle 23.592 Duplikate entfernt und dann ein Vertreter jeder Gruppe erneut hinzugefügt. Dabei wird darauf geachtet, dass bei Gruppen mit unterschiedlicher Polarität der Modalwert innerhalb der Gruppe als Klasse des Vertreters gewählt wird.

Nach diesen Schritten beinhaltet der Datensatz eine Menge von 1.579.634 Datenpunkten. Davon sind 791.293 Tweets positiv und 788.341 negativ, was einer gleichmäßigen Verteilung entspricht.

4.3.2. Testdaten

Die Polarität enthält hier zusätzlich zum Wertebereich der Trainingsdaten den Wert 2 (neutral). 182 Tweets sind positiv (4), 177 negativ (0) und 138 neutral (2). Da der Klassifikator nur mit Daten der Polarität 0 oder 4 trainiert wird, werden alle Tweets des Testdatensatzes mit der Polarität 2 entfernt. Diese würden beim Testen die Zugehörigkeit zu einer Klasse mit dem Wert 0 oder 4 vorhergesagt bekommen, was in jedem Fall falsch wäre, weshalb die Entfernung nötig ist.

Diese 359 Tweets des Testdatensatzes wurden von Hand markiert, was somit im Gegensatz zum Trainingsdatensatz unabhängig von Emoticons geschah. Damit limitiert die unterschiedliche Art der Generierung beider Datensätze die Genauigkeit, die das Modell auf dem Testdatensatz erreichen kann. Diese hängt nämlich unter anderem davon ab, wie genau der Trainingsdatensatz das tatsächlich korrekte Sentiment durch Distant Supervision repräsentiert.

Hier ist es außerdem zum Einsatz von Queries gekommen, mit welchen es möglich ist, Tweets von einer bestimmten Kategorie abzurufen. Die Testdatenpunkte wurden im Zeitraum von Mai bis Juni 2009 gepostet. Damit liegen sie innerhalb des Zeitraumes der Trainingsdaten. Im Falle, dass sich die Sprachgewohnheiten auf Twitter im Laufe der Zeit ändern sollten, ist es vorteilhaft, dass beide Datensätze vom gleichen Zeitraum sind.

4.4. Vorverarbeitung

4.4.1. Säuberung des Textes

Um die Dokumente des Trainingskorpus in ein Format zu bringen, welcher in einer Reduzierung des Featurerraums resultiert, durchläuft jeder Text einen Prozess, welcher an folgendem Beispiel, das alle für die Vorverarbeitung relevanten Elemente enthält, verdeutlicht wird:

```
1  @rick_stone123 I'm sooo sad!!! they killed off Kutner on House, i dont know
       whyyyyyyyy https://tinyurl.com/6r934hrf
```

Im ersten Schritt werden alle Großbuchstaben in Kleinbuchstaben umgewandelt.

```
1  @rick_stone123 i'm sooo sad!!! they killed off kutner on house, i dont know
       whyyyyyyyy https://tinyurl.com/6r934hrf
```

Im Folgenden werden Benutzernamen, die auf Twitter immer mit einem "@" beginnen sowie Hyperlinks und URLs durch ein entsprechendes Schlüsselwort ersetzt. Dafür kommen reguläre Ausdrücke zum Einsatz, mit denen es möglich ist, nach bestimmten Mustern in Texten zu suchen [40]. Desweiteren werden Sonderzeichen und Satzzeichen mittels des regulären Ausdrucks [^\w\s] entfernt. Es werden alle Zeichen ausgewählt, bis auf solche, welche in einem sinnvollen englischen Wort vorkommen können. Diese beinhalten das lateinische Alphabet und die Zahlen von 0 bis 9 sowie Unterstriche. Dabei ist es wichtig, dass dieser Schritt nach dem Ersetzen von Benutzernamen und Hyperlinks geschieht, da insbesondere das Entfernen von Schrägstrichen bei URLs und "@" bei Benutzernamen für eine Verfälschung sorgen würde und eine Erkennung nicht mehr gewährleistet werden kann.

```
1  USERNAME im sooo sad they killed off kutner on house i dont know whyyyyyyyy URL
```

Nun werden die veränderten Wörter als Tokens getrennt. Hierfür kommt das Natural Language Toolkit[10] zum Einsatz, dass als Bibliothek in Python existiert und eine Funktion zur Tokenisierung von Sätzen in einzelne Wörter bereitstellt [41]. Zurückgegeben wird eine Liste, die die einzelnen Wörter enthält. Dies ist hilfreich, um die Textvorverarbeitung mit einzelnen Wörtern fortzusetzen. Für jedes Element in dieser Liste werden nun ausgewählte Verfahren durchgeführt.

```
1  [USERNAME, im, sooo, sad, they, killed, off, kutner, on, house, i, dont, know,
       whyyyyyyyy, URL]
```

Eines dieser Verfahren ist das Auflösen der Kontraktion von Wörtern. Hierbei kommt eine Liste mit bekannten englischen Kontraktionen[11] zum Einsatz, die mit den einzelnen Elementen verglichen wird. Dabei wurde darauf geachtet, dass der Apostroph in jedem Wort dieser Liste entfernt wird, da dieses in unserem Text ebenfalls bereits annulliert wurde. Zudem wurden alternative Formulierungen in dem Verzeichnis eliminiert, so dass jedes kontrahierte Wort nur auf exakt eine ausformulierte Zeichenkette abgebildet wird. Beinhaltet diese Liste eines der Elemente, wird dieses in die entsprechende Ausformulierung umgewandelt. Somit wird vermieden, dass beispielsweise *don't* und *do not* als unterschiedliche Formulierungen betrachtet werden, da *don't* in *do not* umgewandelt wird. Da die Umwandlung in Kleinbuchstaben in unserem Dokument bereits erfolgte, wurde *I* als Großbuchstabe in der Ausformulierung in den semantisch identischen Kleinbuchstaben *i* geändert, da sonst *I* und *i* ebenfalls als unterschiedliche Wörter wahrgenommen werden.

```
1  [USERNAME, i am, sooo, sad, they, killed, off, kutner, on, house, i, do not, know,
       whyyyyyyyy, URL]
```

Ein weiteres Verfahren ist das Entfernen von sich wiederholenden Buchstaben. Dabei werden alle Buchstaben, die mindestens dreimalig in Folge innerhalb eines Wortes vorkommen, so gekürzt, dass sie nun zweimal in Folge vorkommen. Die Übertreibung, die durch Verwendung solcher Wörter ausgedrückt werden soll, soll für das Modell sichtbar sein, in dem sich diese Wörter von den Wörtern ohne hyperbolischen Charakter differenzieren. Jedoch soll keine Unterscheidung zwischen beispielsweise "whyyyy" und "whyyyyyy" erfolgen, wodurch die Anzahl der Features verringert werden soll.

[10] https://github.com/nltk/nltk
[11] https://gist.github.com/loretoparisi/db70e9b91c7f2363a8dc9ecd80d58ce6

```
[USERNAME, i, am, soo, sad, they, killed, off, kutner, on, house, i, do, not, know,
    whyy, URL]
```

Wenn Stoppwörter sowie Wörter mit weniger als 2 Zeichen entfernt werden, resultiert dies in folgender Liste:

```
[USERNAME, soo, sad, killed, kutner, house, know, whyy, URL]
```

Basierend auf den Erkenntnissen der Arbeit von Toman et al. [42] wird in dieser Bachelorarbeit auf den Einsatz der Verfahren Stemming und Lemmatisierung verzichtet, da sich diese Wortnormalisierungstechniken leicht negativ auf Textklassifizierungen auswirken sollen.

Um auswerten zu können, welchen Einfluss die einzelnen Vorverarbeitungsmethoden auf die Genauigkeit des Modells haben, werden fünf verschiedene Aneinanderreihungen von Vorverarbeitungsmethoden angewandt und ausgewertet.

Tabelle 4.1.: Darstellung der angewandten Vorverarbeitungsmethoden in den Versuchsreihen.

Reihe	Umwandeln in Kleinbuchstaben	Entfernung von		Umwandlung von				Entfernen von Stoppwörtern
		Sonder- und Satzzeichen	Wörtern mit Länge <2	Benutzernamen	URLs	sich wiederholenden Buchstaben	Wortkontraktionen	
1	x	x	x					
2	x	x	x	x	x	x		
3	x	x	x	x	x	x		x
4	x	x	x	x	x	x	x	
5	x	x	x	x	x	x	x	x

Tabelle 4.1 zeigt, dass die Reihen aufeinander aufbauen und jeweils neue Methoden inkludieren. Die einzige Ausnahme ist, dass in Reihe 3 zusätzlich zur Reihe 2 Stoppwörter entfernt werden, während bei Reihe 4 Wortkontraktionen aufgelöst werden. Der Einfluss beider Verfahren soll getrennt beobachtet werden, da die Stoppwortliste aufgelöste Wortkontraktionen enhält. Würde man also zuerst die Kontraktionen auflösen, würden einige danach entfernt werden, was zur Folge hätte, dass der Einfluss der Auflösung der Wortkontraktionen nicht in vollem Umfang beobachtet werden könnte. Erst in Reihe 5 sollen beide zusammen angewandt werden.

Jede Reihe wird auf alle Dokumente des Trainings- sowie Testdatensatzes angewandt. Der jeweils entstehende Datensatz wird in einer neuen CSV-Datei gespeichert, welche dann bei der Vektorisierung und dem Training des Modells geladen wird. Dies hat den Vorteil, dass die Vorverarbeitung nur einmalig stattfinden muss. Dabei werden Wörter mit weniger als 2 Zeichen sowie Stoppwörter später im Vektorisierungsprozess entfernt, da die Vectorizer von scikit-learn diese Optionen anbieten.

Um das Ausmaß der Vorverarbeitungsmethoden auf den Featureraum zu untersuchen, wurden die Anzahl der entstehenden Features nach jeder Reihe betrachtet. Dabei wird in Tabelle 4.2 ersichtlich,

dass die Vorverarbeitungsmethoden nach Reihe 2 eine Reduzierung des Featureraums um circa die Hälfte erreichen. Nach Reihe 3, 4 und 5 ist die Reduzierung nicht mehr signifikant.

Tabelle 4.2.: Größe des Featureraums nach Anwendung der Vorverarbeitungsreihen. Der Featureraum nach Vorverarbeitungsreihe 1 wird hier als Ausgangspunkt angesehen, da dort grundlegende notwendige Vorverarbeitungsmethoden durchgeführt werden.

Vorverarbeitungsreihe	**Anzahl Features**	**Anteil**
1	851573	100%
2	428036	50.26%
3	427725	50.23%
4	427959	50.26%
5	427652	50.22%

4.4.2. Vektorisierung des Textes

Für die Umwandlung der vorliegenden Korpora in ein Vektormodell wird der TfidfVectorizer beziehungsweise der CountVectorizer aus der Bibliothek scikit-learn verwendet. Da als Parameter die Dokumente nur als gesamte Zeichenkette akzeptiert werden, wird die vorherige Listendarstellung so umgewandelt, dass die Wörter wieder zu ganzen Sätzen zusammengefügt werden. Aufgrund von Leistungsgrenzen hinsichtlich des Speichers und vorrangig der Trainingsgeschwindigkeit bei den LR-Modellen, wurde die Menge von Features auf 5.000 begrenzt. Absteigend sortiert nach dem absoluten Häufigkeitswert über das komplette Korpus definieren die ersten 5.000 Features den gesamten Featureraum. Im Rahmen dieser Bachelorarbeit wurde sich für das 1-Gramm-Modell entschieden, so dass jedes Wort, welches in dem Korpus vorkommt, einmalig als einzelnes Feature behandelt wird.

Ein beispielhaftes Textkorpus mit zwei Dokumenten nach der Säuberung entsprechend Reihe 3 (vorläufig ohne das Entfernen von Stoppwörtern) hat den folgenden Inhalt:

```
1  this week is not going as i had hoped
2  what tragedy and disaster in the news this week
```

Für dieses resultiert unter Verwendung des TfidfVectorizers, der nachträglich die Stoppwörter annulliert, das Vektormodell, welches in Tabelle 4.3 angegeben ist. Die Features sind als Spalten angegeben und jede Zeile entspricht dem jeweiligen Dokument.

Tabelle 4.3.: BoW für das Beispielkorpus mit der TF-IDF-Metrik.

disaster	**going**	**hoped**	**news**	**tragedy**	**week**
0	0.632	0.632	0	0	0.449
0.534	0	0	0.534	0.534	0.380

Die zugehörigen Klassen der Trainingsdatenpunkte werden als eindimensionaler Vektor gespeichert, bei dem der Index der jeweiligen Klasse dem Index (Zeile) des zugehörigen Textdokumentes im

Wort-Vektormodell entspricht. Der Testdatensatz wird ebenfalls anhand des bereits festgelegten Featureraums, definiert durch den Trainingsdatensatz, in ein eigenes Vektormodell umgewandelt.

4.4.3. Bewältigung von Speicherproblemen

Für alle 1.579.634 Dokumente des Trainingskorpus entsteht also je eine Zeile in der Matrix im Vektormodell. Da die Features auf 5.000 limitiert sind, entsteht eine Matrix mit $1.579.634 \cdot 5.000 = 7.898.170.000$ Einträgen. Bei den gespeicherten Einträgen handelt es sich abhängig vom verwendeten Vectorizer um Werte vom Datentyp float64 beziehungsweise int64, was bedeutet, dass für jeden Wert 8 Byte im Speicher genutzt werden. Bei der Größe der Matrix ergibt sich somit eine Gesamtspeichernutzung von 63,18536 GB.

Der Vectorizer gibt eine Matrix, im Folgenden undicht genannt, zurück, bei welcher Einträge, die eine 0 enthalten, nicht gespeichert werden. Diese Repräsentation beansprucht vergleichsweise wenig Speicherplatz und ist vorteilhaft. Denn nur wenige der 5.000 Features kommen auch in einem Dokument vor, womit die Mehrheit der Werte, die gespeichert werden, 0 beträgt. Die undichte Matrix für das Trainingskorpus hat einen Speicherbedarf von nur 48 Byte, was einer Speichernutzung von $7,59 \cdot 10^{-10}\%$ relativ zur originalen Matrix entspricht.

Da scikit-learn in dieser Arbeit für die Verwendung von MNB und SVM genutzt wird und diese Algorithmen in der Implementierung von scikit-learn die Matrixrepräsentation des Vectorizers als Eingabe zulassen, müssen für diese beiden keine weiteren Vorkehrungen in Bezug auf die Eingabedaten getroffen werden.

Weil jedoch die LR mittels TensorFlow implementiert wird und diese Bibliothek die undichte Matrix nicht als Eingabe nutzen kann, muss diese in eine vollwertige Matrix umgewandelt werden. Hierbei ergibt sich das Problem, dass eine Matrix dieser Größe nicht komplett in den verwendeten Arbeitsspeicher geladen werden kann. Somit kann die gesamte Matrix nicht auf einmal umgewandelt werden, da dies den RAM überlasten würde.

Zuerst wurde versucht, die undichte Matrix in 16 Chunks aufzuteilen und diese als dichte Matrizen auf der Festplatte zu speichern. Hiermit ergibt sich ein durchschnittlicher Speicherbedarf von circa 4 GB pro Datenstück. Bei einem Arbeitsspeicher von 16 GB, abzüglich der Ressourcennutzung durch interne Rechnerprozesse, ist es möglich, jedes Datenteil nacheinander vollständig zu laden und zu verarbeiten. Folglich wurde die zurückgegebene Matrix des Vectorizers in Chunks geteilt, wovon jeder in die dichte Matrixrepräsentation umgewandelt und auf der Festplatte gespeichert wurde. Beim iterativen Trainingsprozess werden diese Chunks dann nacheinander vom Festplattenspeicher in den Arbeitsspeicher geladen. Der Gedanke dabei war, dass das Umwandeln und spätere Laden eines einzigen Chunks nicht zu Speicherfehlern führt, da dessen Größe das Limit nicht übersteigt.

Jedoch ergab sich hierbei das Problem, dass während des Trainings für jede Epoche alle Chunks von der Festplatte geladen werden müssen. Dies brachte einen immensen Geschwindigkeitsverlust beim Training der LR-Modelle mit sich.

Die nötige Lösung ist, die undichte Matrix manuell über eine Generatorfunktion als Batches iterativ an das Modell zu übergeben und nur den gerade benötigten Bereich, also der aktuelle Batch, in eine

dichte Matrix umzuwandeln. Nachdem dieser vom Modell verarbeitet wurde, wird der verwendete Arbeitsspeicher für den nächsten Batch freigegeben. Somit bleibt die Belastung des Arbeitsspeichers gering und es müssen keine zusätzlichen Dateien auf der Festplatte gespeichert werden.

Damit konnte die Trainingsdauer deutlich reduziert werden, denn alle benötigten Daten sind direkt vom Arbeitsspeicher abrufbar.

4.5. Baseline

Als Leitfaden für die Implementierung der Baseline wird sich auf die Arbeit von Go et al. [26] bezogen. Die Implementierung der Vorverarbeitungsmethoden sowie der Modelle musste jedoch selbst erfolgen, weil Go et al. ausschließlich erläutern, welche Schritte sie durchgeführt haben. Im Rahmen dieser Bachelorarbeit werden die ML-Algorithmen MNB, SVM und LR verwendet. Unter der Verwendung von drei Algorithmen, drei Metriken und fünf Vorverarbeitungsreihen wird je ein neues Modell trainiert, so dass 45 unterschiedliche Modelle entstehen. Somit werden also die, in der erwähnten Arbeit [26] genutzten, Vorverarbeitungsmethoden bei unserer Baseline so erweitert, dass eine Evaluation des Einflusses auf die Vorhersagegenauigkeit des Modells möglich ist. Als Trainings- und Testdaten für die Modelle werden die, durch den in 4.4.2 beschriebenen Prozess, entstandenen BoW-Modelle genutzt. Die Auswertung der Ergebnisse folgt in Kapitel 5.

SVM und MNB werden mit scikit-learn implementiert, wodurch es möglich ist, die undichte BoW-Matrix direkt zu übergeben. Hier wird also der Schritt der Umwandlung in eine dichte Matrix übersprungen.

Um die Vorteile von TensorFlow Privacy nutzen zu können, wurde in TensorFlow für die LR ein KNN modelliert, welches diese umsetzt. Dabei wird auf Keras zurückgegriffen. Dadurch, dass hier jedoch keine undichten Matrizen als Parameter übergeben werden können, wird die BoW-Matrix in Batches über eine Generatorfunktion an das Modell übergeben, wobei die Umwandlung in eine dichte Matrix in iterativen Schritten bezogen auf einen Batch erfolgt. Im Folgenden wird der Aufbau des KNN für die LR sowie die Anpassung der Lernrate erläutert.

Das KNN besteht aus einer Eingabeschicht, welches 5000 Neuronen enthält, eines für jedes Feature. Die Sigmoidfunktion wirkt als Aktivierungsfunktion. Die Ausgabe enthält einen Wert im Bereich von 0 bis 1, welcher den Konfidenzwert angibt, dass ein Tweet positiv ist. Bei einem Wert unter 0.5 wird er als negativ klassifiziert, sonst als positiv. Als Verlustfunktion wird die *BinaryCrossEntropy* über Keras bereitgestellt. Als Optimizer wird der SGD-Optimizer von TensorFlow verwendet, wobei die Lernrate im Folgenden ermittelt wird. Dabei ist ein Optimizer ein Algorithmus, der die Anpassung der Gewichte gemäß der Gradienten vornimmt.

Als Validierungssatz wurden zufällige 10% des Trainingsdatensatzes gewählt, um während des Trainings abschätzen zu können, wie sich das Modell auf unbekannten Daten verhält.

Es ist bekannt, dass die Wahl der passenden Lernrate für ein Modell wichtig ist. Diese gibt an, wie groß die Schritte entlang der Kostenfunktion nach jeder Iteration sind. Wählt man die Lernrate zu klein, dauert es länger, das Minimum der Verlustfunktion zu finden, womit mehr Epochen nötig

sind, um das Training zu beenden. Wählt man sie zu groß, besteht die Gefahr, das Minimum zu überspringen, womit der Kostenwert wieder zunimmt. [43]

Das Ziel dieser Analyse ist es, einen geeigneten Lernverlauf zu erreichen, welcher sich dadurch kennzeichnet, dass der Kostenwert auf dem Trainingsdatensatz kontinuierlich sinkt, bis er auf einem Plateau stagniert [44]. Der Verlustwert auf dem Validierungssatz sollte ebenfalls sinken, bevor er eventuell an einem bestimmten Punkt steigt. Falls dieser Punkt erreicht wird oder der Validierungsverlust nicht weiter sinkt, soll das Training beendet werden. Denn Ersteres ist ein Kriterium für Overfitting, bei welchem die Trainingsdaten zu genau erlernt werden [44]. Folglich sinkt die Fähigkeit des Modells, unbekannte Daten zu klassifizieren. Während der Kostenwert auf den Trainingsdaten also weiter sinkt, steigt dieser auf dem Validierungssatz. Das Erreichen eines geeigneten Lernverlaufes bei der Baseline ist vor allem für den Vergleich mit dem DP-Modell wichtig, um den Einfluss der verrauschten Gradienten während des Trainings beobachten zu können. Für die Analyse werden folgende Parameter verwendet:

- Vorverarbeitungsreihe: 2
- Metrik: absoluter Häufigkeitswert
- Batchsize: 16

Einen Ansatz schlägt die Arbeit von Leslie N. Smith [45] vor, bei dem die Lernrate mit jeder Iteration linear steigt, um zu beobachten, welchen Einfluss sie auf den Kostenwert und die Genauigkeit hat. Somit können die Lernraten ausgeschlossen werden, bei denen das Training nur sehr langsam vorangeht sowie solche, bei denen die Genauigkeit unregelmäßig verläuft oder sinkt. Folglich erhält man einen Bereich, der die geeigneten Lernraten angibt.

Bezogen auf unseren Anwendungsfall wird die Lernrate ab einem Startwert von 10^{-7} linear vergrößert, so dass sie aller 3 Epochen um den Faktor 10 erhöht wird. Der Test nimmt daher 21 Epochen in Anspruch, um eine finale Lernrate von 1 zu erreichen. Abbildung 4.1 zeigt, dass der Verlust bei einer Lernrate von circa 10^{-4} das stärkste Gefälle aufweist. Weiterhin liegt das Minimum bei circa 10^{-2}, nachdem das Gefälle des Graphen bei circa 10^{-3} abnimmt. Nach 10^{-2} divergiert der Trainingsprozess und der Kostenwert steigt. Somit ist ein möglicher geeigneter Bereich für die Lernrate l: $10^{-4} \leq l \leq 10^{-2}$.

Das Modell wird zunächst mit zwei unterschiedlichen initialen Lernraten trainiert, welche bei einem Plateau verringert werden. Dessen liegt die Erkenntnis zu Grunde, dass das Verringern der Lernrate im Laufe des Trainingsprozesses einen positiven Einfluss hat [46]. Durch die geringere Lernrate ist es möglich, die Schritte in Richtung Minimum zu verkleinern, wodurch die Optimierung im späteren Trainingsprozess, nachdem bereits ein deutlicher Abfall des Verlustwertes stattfand, verfeinert werden kann. Sobald der Kostenwert auf dem Validierungssatz für 3 Epochen nicht mehr sinkt, wird die Lernrate mit dem Faktor 0.1 multipliziert, wobei darauf geachtet wird, dass diese den Wert von 10^{-6} nicht unterschreitet. Sobald sich der aktuell minimalste Kostenwert für 7 Epochen nicht verringert, wird das Training beendet.

Als Kriterium für das Trainingsende und das Verringern der Lernrate wurde der Kostenwert statt der Genauigkeit auf den Validierungsdaten gewählt. Dies hat den Grund, dass die Genauigkeit zwar

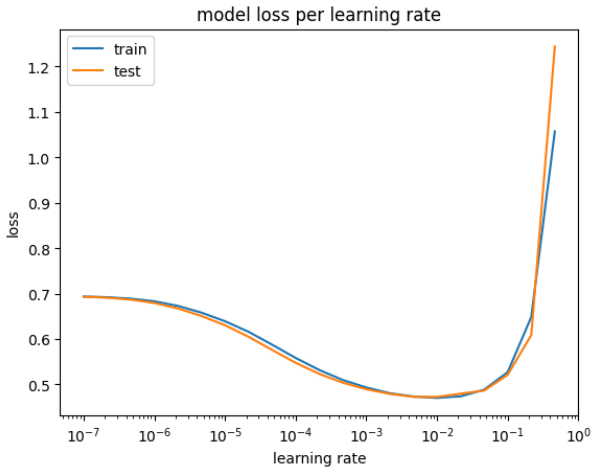


Abbildung 4.1.: „LR range test“ [45] angewandt auf unseren Datensatz: Kostenwert in Abhängigkeit von der Lernrate.

angibt, welcher Anteil korrekt klassifiziert wurde, der Kostenwert jedoch zusätzlich quantifiziert, wie sicher das Modell bei der Klassifizierung ist.

Zusätzlich wird Smiths Idee zum zyklischen Verändern der Lernrate angewandt [45]. Aller 4 Epochen wird ein Zyklus durchlaufen, bei dem die Lernrate von 10^{-4} auf 10^{-2} steigt, bevor sie wieder auf 10^{-4} sinkt. Der Vorteil liegt darin, dass Sattelpunkte der Kostenfunktion durch zeitweises Erhöhen der Lernrate übersprungen werden können [45]. Eine zu kleine Lernrate könnte nicht den nötigen Gradienten hervorbringen, um diesem Sattelpunkt zu entkommen. Hier wird das Training ebenfalls nach 7 Epochen ohne verbesserte Validierungskosten beendet.

In Abb. 4.2 wird ersichtlich, dass das Training bei einer initialen Lernrate von 10^{-2} bereits nach einer Epoche gering ist, da die Schritte in Richtung Minimum größer sind als bei 10^{-3}. Somit ist es in diesem Fall von Vorteil, zu Beginn des Trainings eine höhere Lernrate zu wählen, um erste große Schritte in Richtung Minimum zu tätigen. Jedoch ist hier der Verlauf des Kostenwertes unregelmäßig, weshalb es sich durchaus bewährt, die Lernrate im späteren Trainingsprozess zu reduzieren. Der dadurch entstehende Effekt ist ein stagnierender Verlust ab Epoche 20. Bei Smiths Ansatz erreicht der Kostenwert ebenfalls schnell das Minimum, jedoch fällt es dem Modell durch den zyklischen Charakter der Lernrate schwer, dass der Verlustwert stagniert.

Wie aus Tabelle 4.4 hervorgeht, wird mittels des Initialwertes 10^{-2} der niedrigste Verlust erreicht. Mit 10^{-3} wird zwar die exakt gleiche Validierungsgenauigkeit erreicht, jedoch mit dem Nachteil einer längeren Trainingsdauer. Smiths zyklischer Ansatz sichert unter dem Kompromiss der minimal schlechteren Genauigkeit eine signifikant schnellere Optimierung des Kostenwertes, wobei diese jedoch qualitativ schlechter ist als bei 10^{-2}. Es muss also abgewogen werden, ob die Trainingsgeschwindigkeit oder die optimale Minimierung der Verlustfunktion im Vordergrund stehen soll. Wenn in allen LR-Modellen die bestmögliche Genauigkeit erreicht werden soll, und somit zu diesem Zweck letzteres wichtiger ist, sollte eine initiale Lernrate von 10^{-2} sowie die Reduzierungstechnik verwendet werden.

Zum Zweck der Vergleichbarkeit zu dem DP-Modell bietet sich jedoch die Lernrate 10^{-3} am meisten an, da sie ein gleichmäßiges Kostengefälle hervorbringt, und dies über einen längeren Zeitraum.

Wenn also die Verlustkurven verglichen werden sollen, ist es für den Vergleich angenehmer, eine größere Epochenspanne zu betrachten, in der es nicht zu unvorhersehbaren Steigungen des Kostenwertes und divergierendem Trainingsverhalten kommt. Deshalb soll für den Zweck dieser Arbeit die Lernrate 10^{-3} für alle weiteren LR-Modelle verwendet werden. Im weiteren Verlauf der Experimente fiel jedoch auf, dass diese Lernrate bei der TF-IDF Metrik eine zu hohe Trainingsdauer forderte, weshalb dort 10^{-2} verwendet wird.

Tabelle 4.4.: Wert und Zeitpunkt des erreichten minimalen Kostenwertes sowie entsprechende Genauigkeit auf dem Validierungssatz pro Lernrate.

Lernrate	**Kostenminimum**	**Genauigkeit**	**bei Epoche**
Zyklus	0.46680	79.03%	13
10^{-2}	0.46634	79.04%	26
10^{-3}	0.46698	79.04%	47

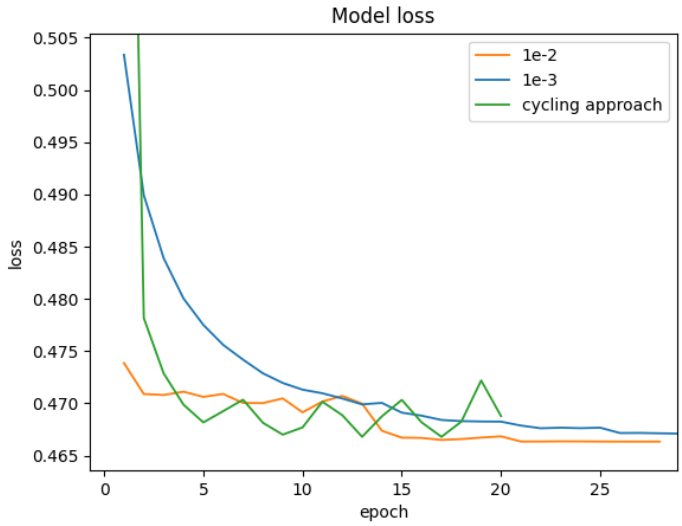


Abbildung 4.2.: Verlauf des Validierungsverlustes des Modells pro verwendeter Lernrate.

4.6. Privatsphäre-erhaltendes Modell

Um ein privatsphäre-erhaltendes Modell für die Sentiment Analyse zu schaffen und um abzuschätzen, welchen Einfluss die Privatisierung auf dieses hat, wird die Baseline durch DP erweitert.

Da die LR mit der Vorverarbeitungsreihe 3 sowie der Metrik des absoluten Häufigkeitswertes ihre höchste Genauigkeit vorwies, wird aufbauend auf dieser Grundlage das DP-Modell implementiert. Dafür wird der, in der Baseline genutzte, SGD-Optimizer der Bibliothek TensorFlow durch einen entsprechenden DP-SGD-Optimizer aus TensorFlow Privacy ersetzt. Folglich wird der *DP-KerasSGDOptimizer* eingesetzt, welcher DP implementiert.

Das Modell wird für 58 Epochen trainiert, da die Baseline dort das Kostenminimum erreichte. Die Lernrate wird, um dem nicht-privaten Modell zu entsprechen, auf 10^{-3} initialisiert und bei einem Plateau reduziert. TensorFlow Privacy erlaubt die Angabe verschiedener Parameter, aus denen unter anderem die Privatsphäremetriken berechnet werden können. Über *noise_multiplier* wird angegeben, wie viel Rauschen beim Training hinzugefügt werden soll. Um unterschiedliche ε-Werte zu erhalten, wird dieser Parameter während den Experimenten verändert.

Als Batchgröße wird wie bei der Baseline der Wert 16 gewählt. Unser Trainingsdatensatz enthält nach dem Vorverarbeiten sowie nach der Abgabe von 10% an einen Validierungsdatensatz 1.421.670 Tweets. Dabei ist eine Limitierung des TensorFlow Privacy Optimizers, dass alle Batches vollständig sein müssen, weshalb 6 Datenpunkte zusätzlich entfernt werden. So ist der Trainingsdatensatz bezüglich der Länge durch die Batchgröße teilbar und enthält 1.421.664 Datenpunkte. Damit ergibt sich δ nach $\delta \leq \frac{1}{N}$ zu $\delta = 1e-7$. *L2_norm_clip* wird auf 1.0 gesetzt und die Anzahl der Mikrobatches der Batchgröße angeglichen. L2 Norm Clipping bezieht sich auf das Begrenzen der euklidischen Norm des Gradienten, so dass der Einfluss eines individuellen Datenpunkts begrenzt wird [47].

Als Vorbild für die ε-Werte gilt die Arbeit von Lange et al. [48]. Dort wird DP auf die Diagnose von Covid-19 anhand von Röntgenbildern der Lunge angewandt, um die Patientendaten zu schützen. Demnach wird unser Modell mit ε der Menge $\{0.1, 1, 10\}$ trainiert, um den Einfluss verschiedener Privatisierungsgrade auf die Performance zu beurteilen. Zusätzlich entspricht der ε-Wert ∞ der Baseline, wo keine Privatisierung mittels DP gewährleistet wird. Da Werte von $\varepsilon \leq 1$ eine hohe Privatsphäre schaffen, werden diese angestrebt [49, 50]. Um das Ausmaß benachbarter Privatisierungsgrade auszuwerten, wird ε um eine entsprechende Zehnerpotenz verschoben [48].

5. Evaluierung der Ergebnisse

5.1. Auswertung der Baseline-Modelle

In diesem Abschnitt werden die resultierenden Ergebnisse, dargestellt in Tabelle 5.1, bezüglich der Sentiment Analyse in Abhängigkeit der verwendeten Vorverarbeitungsreihe, Metrik sowie des Algorithmus ausgewertet und verglichen. Die angegebenen Reihen wurden in Abschnitt 4.4.1 definiert.

Tabelle 5.1.: Testgenauigkeit der Baseline-Modelle.

Metrik	**Vorverarbeitungsreihe**	**Algorithmus**		
		MNB	**SVM**	**LR**
absoluter Häufigkeitswert	Reihe 1	79.11%	80.78%	80.78%
	Reihe 2	79.39%	80.78%	80.22%
	Reihe 3	**81.89%**	81.06%	**80.78%**
	Reihe 4	78.55%	79.94%	79.94%
	Reihe 5	81.06%	79.67%	79.67%
TF-IDF	Reihe 1	79.11%	79.11%	79.11%
	Reihe 2	80.22%	79.11%	79.94%
	Reihe 3	81.34%	79.94%	79.67%
	Reihe 4	79.11%	78.83%	78.83%
	Reihe 5	81.34%	80.50%	79.67%
Existenz	Reihe 1	78.55%	80.22%	79.94%
	Reihe 2	78.55%	80.78%	80.22%
	Reihe 3	81.34%	79.94%	80.50%
	Reihe 4	79.11%	80.78%	79.94%
	Reihe 5	80.50%	79.39%	78.83%

In Abbildung 5.1(a) fällt auf, dass die Vorverarbeitungsreihen ungeachtet der verwendeten Metrik einen einheitlichen, signifikanten Einfluss auf MNB haben. Das Motiv ist, dass die Genauigkeit von Reihe 1 bis Reihe 3 steigt, bei Reihe 4 abnimmt, und bei Reihe 5 wieder ansteigt. Dabei profitiert MNB am meisten von dem Entfernen von Stoppwörtern.

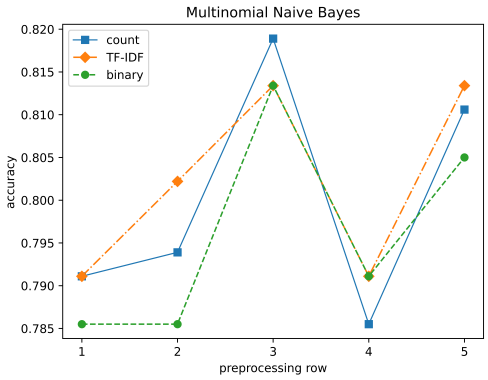
Weniger Einfluss haben die Vorverarbeitungsmethoden auf SVM, wie in Abbildung 5.1(b) dargestellt wird. Hier stellt sich die Verbesserung je nach verwendeter Metrik als unregelmäßig heraus. Bei der Metrik des Existenzwertes übt das Entfernen von Stoppwörtern einen schlechten Einfluss aus, während SVM bei den anderen beiden Metriken dem Motiv von MNB folgt, mit der Ausnahme, dass beim absoluten Häufigkeitswert die Genauigkeit bei Reihe 5 weiter sinkt, statt steigt.

Bei der LR ist in Abbildung 5.1(c) erkennbar, dass sich dort die höchste Genauigkeit je Metrik nach Reihe 2 beziehungsweise 3 finden lässt. Bei der TF-IDF Metrik steigt die Genauigkeit nach Reihe 5, während sie bei den anderen Metriken weiter abfällt.

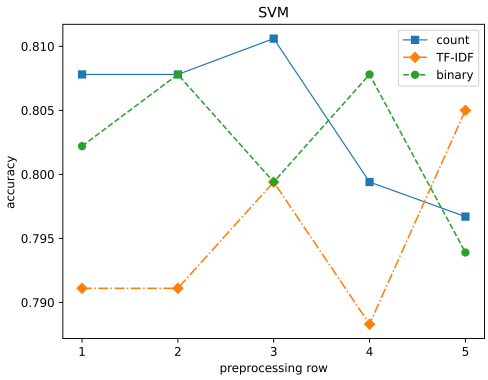
Die höchste Genauigkeit der Baseline erzielte MNB mit 81.89% mit der Vorverarbeitungsreihe 3 sowie der Metrik des absoluten Häufigkeitswertes, während MNB ebenfalls die schlechteste Genauigkeit mit 78.55% hervorbringt. Dies zeigt, wie wichtig die korrekte Wahl von Vorverarbeitungsreihe

und Metrik für MNB ist, um eine möglichst hohe Performance zu erreichen. 81.06% ist die maximal erreichte Genauigkeit von SVM unter den gleichen Parametern wie der höchste Wert von MNB. Mit 80.78% unter der Vorverarbeitungsreihe 1 und 3 sowie der Metrik des absoluten Häufigkeitswertes erreicht die LR ihren Höchstwert.

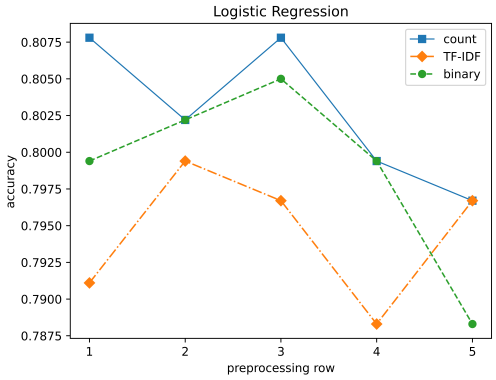
Die TF-IDF Metrik resultiert in der niedrigsten durchschnittlichen Genauigkeit bei SVM und LR, während der absolute Häufigkeitswert auch im Durchnitt über alle Reihen die besten Ergebnisse in den analysierten Algorithmen hervorbringt. Somit zeigen die Ergebnisse, dass diese Metrik den anderen überlegen ist und in Kombination mit der Vorverarbeitungsreihe 3 die bestmögliche Genauigkeit erreicht.



(a) MNB.



(b) SVM.



(c) LR.

Abbildung 5.1.: Veränderung der Genauigkeit nach verwendeter Vorverarbeitungsreihe und Metrik.

Allgemein fällt auf, dass in allen Fällen bis auf SVM mit der Existenzmetrik die Transformierung von Wortkontraktionen einen negativen Einfluss ausübt. Die Vermutung ist, dass das Umwandeln in einem Unigramm-Modell dazu führt, dass der Kontext der einzelnen Wörter, der durch die Wortverbindung verdeutlicht wird, durch das Aufteilen in mehrere Features verloren geht. Beispielsweise behandelt das Modell *don't* als einzelnes Feature, nach der Umwandlung werden *do* und *not* jedoch als getrennte Features betrachtet, womit der direkte Zusammenhang, in dem diese beiden Wörter stehen, für das Modell nicht mehr ersichtlich ist. Dies wäre ein möglicher Grund, weshalb die Genauigkeit leidet.

In der Arbeit von Go et al. [26] wurden mit dem gleichen Datensatz, der Vorverarbeitungsreihe 2 und dem Unigramm-Modell folgende Genauigkeitswerte erreicht:

- MNB mit Metrik des absoluten Häufigkeitswerts: 81.3%
- SVM mit Metrik der Existenz: 82.2%
- LR mit eigener gewichteter Metrik: 80.5%

Der größte Unterschied liegt in der schlechteren Genauigkeit von unserem MNB (79.39%), welche jedoch durch das zusätzliche Entfernen von Stoppwörtern das Ergebnis von Go et al. übertreffen konnte (81.89%). Die Differenz zu unserer SVM Performance (80.78%) lässt sich vermutlich unter anderem dadurch erklären, dass in der verwandten Arbeit eine andere SVM-Software, namentlich "SVM$^{\text{light}}$", verwendet wurde. Das Ergebnis von Gos LR ist mit den Ergebnissen der Metrik des absoluten Häufigkeitswertes in dieser Arbeit vergleichbar (80.22%).

Generell lässt sich sagen, dass marginale Unterschiede dadurch entstehen können, dass in dieser Bachelorarbeit der Featureraum auf 5.000 limitiert wurde, während er in Go's Arbeit vollends ausgeschöpft wurde. Weiterhin wurde in der Arbeit eine Feature-Anzahl von 794.876 bei keiner Vorverarbeitung angegeben, während in dieser Bachelorarbeit selbst nach der ersten Reihe[12] und dem Entfernen von Gos unentdeckten Duplikaten 851.573 Features verbleiben. Diese unterschiedliche Reduzierung des Featureraums kann die leichten Unterschiede zur verwandten Arbeit ebenfalls erklären.

5.2. Auswertung der privatsphäre-erhaltenden Modelle

Tabelle 5.2 bestätigt den in Abschnitt 2.4.2 beschriebenen Kompromiss zwischen Privatisierung und Performance des Modells. Denn je mehr Privatisierung mit sinkenden ε erreicht wird, desto weniger Genauigkeit erreicht das Modell auf dem Testdatensatz. Durch das mathematische Rauschen, dass den Gradienten zugefügt wird, ist es schwerer, das Minimum der Kostenfunktion über diese Gradienten zu finden, weshalb die Gewichte nicht wie bei der Baseline optimal angepasst werden können, was zu einer schlechteren Performance führt.

Tabelle 5.2.: DP-Modell mit Privatsphäreparameter ε und erreichter Genauigkeit auf dem Testdatensatz. Die Genauigkeit für $\varepsilon=\infty$ lässt sich aus Tabelle 5.1 ermitteln.

ε	Genauigkeit
0.1	76.04%
1	77.99%
10	78.27%
∞	80.78%

Die Frage, ob ein bestimmtes Modell für die Praxis geeignet ist, ist subjektiv und hängt davon ab, wieviel Genauigkeit benötigt wird und in welchem Maße der Entwickler bereit ist, diese zu opfern, um ein privates Modell zu schaffen. Es muss also abgewogen werden, ob in der Praxis ein Modell

[12] Umwandeln in Kleinbuchstaben, Entfernung von Sonder- und Sattzeichen sowie von Wörtern mit einer Länge < 2

genutzt werden soll, dass eine Vorhersagegenauigkeit von 80.78% auf dem Testdatensatz erreicht, bei welchem es möglich ist, Rückschlüsse auf Trainingsdaten zu schließen, oder beispielsweise ein Modell mit 77.99% Genauigkeit, wo jedoch eine entsprechende Privatisierung mit $\varepsilon = 1$ gewährleistet wird. Als Vergleich für ein beispielhaftes Privatsphärebudget in der Praxis kann Microsoft angeführt werden, wo $\varepsilon = 1.672$ für das Sammeln der Nutzerdaten bezüglich der Menge verbrachter Sekunden innerhalb einer Windows 10 Anwendung genutzt wird [51].

In Abb. 5.2 wird ersichtlich, dass der Trainingsprozess bei $\varepsilon = 1$ und $\varepsilon = 10$ ähnlich verläuft und kein Plateau nach 58 Epochen erreicht. Würde man die Trainingsdauer jedoch verlängern, erhöht sich ε, da die Epochenanzahl zunimmt, weshalb die Begrenzung auf 58 notwendig ist. Das Steigern der Performance eines DP-Modells über die Verlängerung der Trainingsdauer bietet sich also durch die Beeinflussung des Grades der Privatsphäre nicht an. Bei $\varepsilon = 0.1$ erreicht das Modell nach 26 Epochen ein Plateau, an dem sich die Genauigkeit nicht mehr verbessert.

Es fällt auf, dass der Abstand der Genauigkeitskurven zwischen kleinen Werten wie $\varepsilon = 1$ und $\varepsilon = 0.1$ größer ist als zwischen $\varepsilon = 1$ und $\varepsilon = 10$. Die Auswirkung auf das Training ist also bei Werten innerhalb $1 \leq \varepsilon \leq 10$ ohne großen Unterschied, während bei $\varepsilon \leq 1$ ein größerer Verlust in der Genauigkeitsentwicklung zu verzeichnen ist.

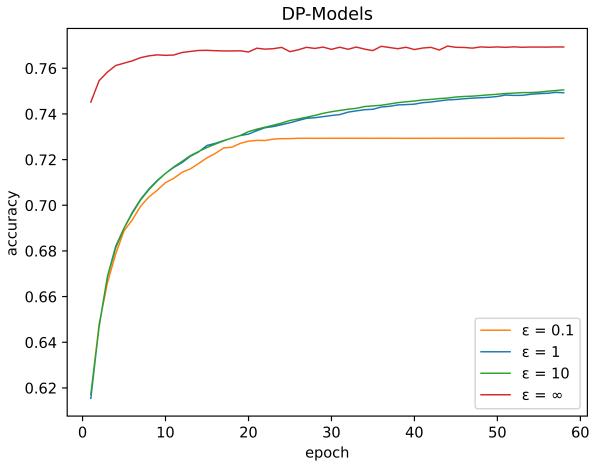


Abbildung 5.2.: Verlauf der Genauigkeit auf dem Validierungssatz der DP-Modelle mit unterschiedlichen ε-Werten.

Im Vergleich zur verwandten Arbeit [30], wo die LR unter Verwendung von Bloomfiltern keinen Performanceverlust verzeichnete, zeigt sich hier durch die Anwendung von DP eine Reduktion in der Genauigkeit. Wie jedoch bereits in Kapitel 3 erläutert, ist die DP die überlegene Methode, wenn eine sichere und zuverlässige Privatisierung das Ziel ist.

6. Fazit und Ausblick

Das Ziel dieser Arbeit war es, ein privatsphäre-erhaltendes ML-Modell für die Sentiment Analyse auf Twitter zu entwickeln und den Einfluss der Privatisierung mittels DP auf die Nutzbarkeit zu beurteilen. Dafür wurde zunächst eine nicht-private Baseline entwickelt, welche dann um DP mit den Privatsphäreparametern $\varepsilon = \{0.1, 1, 10\}$ erweitert wurde.

Hinsichtlich der Vorverarbeitung bei der Sentiment Analyse kamen wir zu dem Ergebnis, dass diese den größten Einfluss auf MNB und den geringsten auf SVM hat. Dabei zeigt sich das Entfernen von Stoppwörtern als allgemein vorteilhaft. Eine von uns neu gewonnene Erkenntnis ist, dass das Auflösen von Wortkontraktionen in einem Unigramm-Modell einen nachteiligen Effekt hervorbringt. Die höchste Genauigkeit der nicht-privaten Baseline erreicht MNB mit der Metrik des absoluten Häufigkeitswertes unter Anwendung der Vorverarbeitungsreihe 3 mit 81.89%. Die niedrigste Genauigkeit erreichte ebenfalls MNB mit 78.55% mit der gleichen Metrik unter Reihe 4.

Weiterhin haben wir festgestellt, dass die Implementierung von DP mit der LR für diesen Anwendungsfall ohne einen großen Genauigkeitsverlust möglich ist. Der niedrigste Genauigkeitswert auf dem Testdatensatz war hierbei 76.04% mit $\varepsilon = 0.1$ im Vergleich zur Baseline (80.78% mit $\varepsilon = \infty$).

Ob dieses Ergebnis eine zuverlässige Anwendbarkeit des privatsphäre-erhaltenden Modells gewährleistet, ist eine subjektive Einschätzung und abhängig davon, wieviel Genauigkeit benötigt wird und in welchem Maße der Entwickler bereit ist, diese zu opfern, um ein privates Modell zu schaffen, welches keine Extrahierung von Trainingsdaten durch beispielsweise MIAs zulässt. Rein basierend auf der verbleibenden Genauigkeit ist die zuverlässige Nutzbarkeit des Modells trotz des vorhandenen Kompromisses zwischen Privatsphäre und Performance zwischen 2.51% und 4.74% gewährleistet. Dieser Genauigkeitsverlust für unseren Anwendungsfall der Sentiment Analyse auf Twitter erscheint in Kontrast zur gewonnenen Privatisierung gering.

Für weiterführende Arbeiten bietet sich an, einen MIA auf das Modell durchzuführen, um auch aus praktischer Sicht beurteilen zu können, wie sehr die Trainingsdaten vor Extrahierung geschützt sind. Um die Genauigkeit unserer Sentiment Analyse weiterhin zu verbessern, ist es möglich, die Experimente mit einem unbegrenzten Featureraum durchzuführen und andere Vorverarbeitungsmethoden anzuwenden. Zudem wurde gezeigt, dass ein Recurrent Neural Network mittels Worteinbettung für eine genauere Sentiment Analyse sorgen könnte [52]. Allgemein scheinen Worteinbettungen, im Gegensatz zu dem hier vorgestellten BoW-Ansatz mit Unigrammen, vorteilhafter zu sein [53]. Demnach wäre es interessant, diese Ansätze mittels DP zu implementieren und zu klären, inwieweit die Nutzbarkeit der entstehenden Modelle beeinflusst wird.

Literatur

[1] Federico Neri u. a. „Sentiment analysis on social media“. In: *2012 IEEE/ACM international conference on advances in social networks analysis and mining*. IEEE. 2012, S. 919–926.

[2] Bing Liu. *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. 2. Aufl. Studies in Natural Language Processing. Cambridge University Press, 2020. DOI: 10.1017/9781108639286.

[3] Huina Mao, Xin Shuai und Apu Kapadia. „Loose tweets: an analysis of privacy leaks on twitter“. In: *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*. 2011, S. 1–12.

[4] Reza Shokri u. a. *Membership Inference Attacks against Machine Learning Models*. 2016. DOI: 10.48550/ARXIV.1610.05820. URL: https://arxiv.org/abs/1610.05820.

[5] Greg Pass, Abdur Chowdhury und Cayley Torgeson. „A picture of search“. In: *Proceedings of the 1st international conference on Scalable information systems*. 2006.

[6] Rosie Jones u. a. „'I know what you did last summer' query logs and user privacy“. In: *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*. 2007, S. 909–914.

[7] Mohammad Al-Rubaie und J Morris Chang. „Privacy-preserving machine learning: Threats and solutions“. In: *IEEE Security & Privacy* 17.2 (2019), S. 49–58.

[8] Erik Cambria u. a. „A practical guide to sentiment analysis“. In: (2017).

[9] Dan Jurafsky und James H. Martin. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Upper Saddle River, N.J.: Pearson Prentice Hall, 2009. ISBN: 9780131873216 0131873210.

[10] Giulio Angiani u. a. „A Comparison between Preprocessing Techniques for Sentiment Analysis in Twitter.“ In: *KDWeb*. 2016.

[11] Qiong Liu und Ying Wu. „Supervised Learning“. In: (Jan. 2012). DOI: 10.1007/978-1-4419-1428-6_451.

[12] Steve Shwartz. „12 Twitter Sentiment Analysis Algorithms Compared“. In: *AI Perspectives* (2021). Online; aufgerufen am 05.12.2022, https://www.aiperspectives.com/twitter-sentiment-analysis/.

[13] Niklas Lang. „Stemming vs. Lemmatization“. In: *databasecamp.de* (2022). Online; aufgerufen am 05.12.2022, https://databasecamp.de/daten/stemming-lemmatizations.

[14] Akrivi Krouska, Christos Troussas und Maria Virvou. „The effect of preprocessing techniques on Twitter sentiment analysis“. In: *2016 7th international conference on information, intelligence, systems & applications (IISA)*. IEEE. 2016, S. 1–5.

[15] Jason Brownlee. „A Gentle Introduction to the Bag-of-Words Model“. In: *machinelearningmastery.com* (2017). Online; aufgerufen am 06.12.2022, https://machinelearningmastery.com/gentle-introduction-bag-words-model/.

[16] Rui Zhao und Kezhi Mao. „Fuzzy bag-of-words model for document representation“. In: *IEEE transactions on fuzzy systems* 26.2 (2017), S. 794–804.

[17] Lukáš Havrlant und Vladik Kreinovich. „A simple probabilistic explanation of term frequency-inverse document frequency (tf-idf) heuristic (and variations motivated by this explanation)“. In: *International Journal of General Systems* 46.1 (2017), S. 27–36.

[18] Hans Christian, Mikhael Pramodana Agus und Derwin Suhartono. „Single document automatic text summarization using term frequency-inverse document frequency (TF-IDF)“. In: *ComTech: Computer, Mathematics and Engineering Applications* 7.4 (2016), S. 285–294.

[19] Benoit Liquet, Sarat Moka und Yoni Nazarathy. *The Mathematical Engineering of Deep Learning*. 2022.

[20] Sebastian Ruder. „An overview of gradient descent optimization algorithms“. In: *arXiv preprint arXiv:1609.04747* (2016).

[21] Sebastian Rachka. *What are gradient descent and stochastic gradient descent?* Online; aufgerufen am 20.12.2022, `https://sebastianraschka.com/faq/docs/gradient-optimization.html`. 2022.

[22] Hongsheng Hu u. a. „Membership inference attacks on machine learning: A survey“. In: *ACM Computing Surveys (CSUR)* 54.11s (2022), S. 1–37.

[23] William E Winkler. „Matching and record linkage“. In: *Business survey methods* 1 (1995), S. 355–384.

[24] Cynthia Dwork, Aaron Roth u. a. „The algorithmic foundations of differential privacy“. In: *Foundations and Trends® in Theoretical Computer Science* 9.3–4 (2014), S. 211–407.

[25] Roxana Danger. *Differential Privacy: What is all the noise about?* 2022. DOI: `10.48550/ARXIV.2205.09453`. URL: `https://arxiv.org/abs/2205.09453`.

[26] Alec Go, Richa Bhayani und Lei Huang. „Twitter sentiment classification using distant supervision“. In: *CS224N project report, Stanford* 1.12 (2009), S. 2009.

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

[28] Huu-Thanh Duong und Tram-Anh Nguyen-Thi. „A review: preprocessing techniques and data augmentation for sentiment analysis“. In: *Computational Social Networks* 8.1 (2021), S. 1–16.

[29] I Hemalatha, GP Saradhi Varma und A Govardhan. „Preprocessing the informal text for efficient sentiment analysis“. In: *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)* 1.2 (2012), S. 58–61.

[30] Hugo Alatrista-Salas, Hugo Cordero und Miguel Nunez-del-Prado. „PS I Love You: Privacy Aware Sentiment Classification“. In: *Computación y Sistemas* 23.4 (2019), S. 1507–1515.

[31] Peter Christen u. a. „Pattern-mining based cryptanalysis of Bloom filters for privacy-preserving record linkage“. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2018, S. 530–542.

[32] Thilina Ranbaduge, Dinusha Vatsalan und Ming Ding. „Privacy-preserving Deep Learning based Record Linkage“. In: *arXiv preprint arXiv:2211.02161* (2022).

[33] F. Pedregosa u. a. „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830.

[34] Martín Abadi u. a. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[35] H Brendan McMahan u. a. „A general approach to adding differential privacy to iterative training procedures“. In: *arXiv preprint arXiv:1812.06210* (2018).

[36] Alec Go, Richa Bhayani und Lei Huang. *Sentiment140 dataset with 1.6 million tweets*. Online; aufgerufen am 05.10.2022, https://www.kaggle.com/datasets/kazanova/sentiment140. 2009.

[37] twitter.com. *Search Tweets - How to build a query | Docs | Twitter Developer Platform*. Online; aufgerufen am 23.10.2022, https://developer.twitter.com/en/docs/twitter-api/tweets/search/introduction.

[38] twitter.com. *Make your first request to the Twitter API | Docs | Twitter Developer Platform*. Online; aufgerufen am 23.10.2022, https://developer.twitter.com/en/docs/twitter-api/getting-started/make-your-first-request.

[39] Harsurinder Kaur, Husanbir Singh Pannu und Avleen Kaur Malhi. „A systematic review on imbalanced data challenges in machine learning: Applications and solutions“. In: *ACM Computing Surveys (CSUR)* 52.4 (2019), S. 1–36.

[40] Jan Goyvaerts. *Regular-Expressions.info - Regex Tutorial, Examples and Reference - Regexp Patterns*. Online; aufgerufen am 16.11.2022, https://www.regular-expressions.info/.

[41] Steven Bird, Ewan Klein und Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. "O'Reilly Media, Inc.", 2009.

[42] Michal Toman, Roman Tesar und Karel Jezek. „Influence of word normalization on text classification“. In: *Proceedings of InSciT* 4 (2006), S. 354–358.

[43] Mateusz Kwasniak. „How to Decide on Learning Rate“. In: *towardsdatascience* (2020). Online; aufgerufen am 17.01.2022, https://towardsdatascience.com/how-to-decide-on-learning-rate-6b6996510c98.

[44] Xue Ying. „An overview of overfitting and its solutions“. In: *Journal of physics: Conference series*. Bd. 1168. IOP Publishing. 2019, S. 022022.

[45] Leslie N. Smith. *Cyclical Learning Rates for Training Neural Networks*. 2015. DOI: 10.48550/ARXIV.1506.01186. URL: https://arxiv.org/abs/1506.01186.

[46] Kaichao You u. a. „How does learning rate decay help modern neural networks?“ In: *arXiv preprint arXiv:1908.01878* (2019).

[47] Martin Abadi u. a. „Deep learning with differential privacy“. In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 2016, S. 308–318.

[48] Lucas Lange, Maja Schneider und Erhard Rahm. *Privacy in Practice: Private COVID-19 Detection in X-Ray Images*. Nov. 2022. DOI: 10.48550/arXiv.2211.11434. arXiv: 2211.11434 [cs]. URL: http://arxiv.org/abs/2211.11434 (besucht am 22. 11. 2022).

[49] Nicholas Carlini u. a. „The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks.“ In: *USENIX Security Symposium*. Bd. 267. 2019.

[50] Milad Nasr u. a. „Adversary instantiation: Lower bounds for differentially private machine learning“. In: *2021 IEEE Symposium on security and privacy (SP)*. IEEE. 2021, S. 866–882.

[51] Bolin Ding, Janardhan Kulkarni und Sergey Yekhanin. „Collecting Telemetry Data Privately“. In: *CoRR* abs/1712.01524 (2017). arXiv: 1712.01524. URL: http://arxiv.org/abs/1712.01524.

[52] Nhan Cach Dang, Maria N Moreno-Garcia und Fernando De la Prieta. „Sentiment analysis based on deep learning: A comparative study“. In: *Electronics* 9.3 (2020), S. 483.

[53] Yijun Shao u. a. „Clinical Text Classification with Word Embedding Features vs. Bag-of-Words Features“. In: *2018 IEEE International Conference on Big Data (Big Data)*. 2018, S. 2874–2878. DOI: 10.1109/BigData.2018.8622345.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit mit dem Thema:

„Privatsphäre-erhaltende Sentiment Analyse auf Twitter“

selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Leipzig, den 20.03.2023

Felix Vogel