Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Ferdinand Emanuel Trendelenburg

Verbesserung der Klassifizierung bei geringer Datenmenge unter Berücksichtigung des Wortkontexts

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Ferdinand Emanuel Trendelenburg

Verbesserung der Klassifizierung bei geringer Datenmenge unter Berücksichtigung des Wortkontexts

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Marina Tropmann-Frick
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 5. Januar 2024

Ferdinand Emanuel Trendelenburg

Thema der Arbeit
Verbesserung der Klassifizierung bei geringer Datenmenge unter Berücksichtigung des Wortkontexts

Stichworte
NLP, Text Klassifizierung, Transformer, BERT, Logistsische Regression

Kurzzusammenfassung
Diese Arbeit beschäftigt sich mit einigen Text-Klassifizierungsalgorithmen und derer Performance auf Datensätzen mit geringer Test- und Validierungsdatenmenge. Außerdem werden zwei neuartige Ansätze vorgestellt, die mittels einer bestimmten Kontextsuche eine höhere Zuverlässigkeit erbringen könnten. Dafür wurde ein bestehender, etablierter Algorithmus modifiziert und ein zweiter neuer Algortihmus konzipiert und implementiert. Die Performance dieser neuen Algortihmen wurden anschließend mit der Performance einer Auswahl von etablierten Algorithmen verglichen. In diesem Vergleich konnte keine zufriedenstellende Verbesserung der Klassifizierung durch das Hinzufügen einer Kontextvariable erzielt werden. Allerdings konnte ein Trend bei der Ergenzung um die Kontextvariable zu dem etablierten Algorithmus beobachtet werden, der sich allerdings nur auf 0.4% beläuft. Diese Verbesserung ist nur marginal, jedoch gibt diese einen Hinweis darauf, dass eine stärkere Verbesserung erreicht werden könnte, wenn an der Stelle weiter geforscht werden würde.

Title of the paper
Improved classification with a small amount of data, taking into account the word context

Keywords
NLP, Text classifyer, Transformer, BERT, Logistical regression

Abstract
This paper deals with some text classification algorithms and their performance on datasets with few test and validation data. In addition, two novel approaches are presented that could yield a higher performance by using context search. For this purpose, an existing, established algorithm was modified and a second new algorithm was designed and implemented. The performance of these new algorithms was then compared against the performance of a selection of established algorithms. In this comparison, no satisfactory improvement in classification could be achieved by adding a context variable. However, a trend can be observed in the addition of the context variable to the established algorithm, although this only amounts to

0.4%. This improvement is only marginal, but it gives an indication that a greater improvement could be achieved if further research was carried out in this area.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Problemstellung und Motivation

In der Fakultät für Geisteswissenschaften unter der Leitung von Herrn Prof. Dr. Philippe Depreux an der Universität Hamburg ist es üblich, lateinische Schenkungsurkunden (Abb. 1.1, 1.2) händisch in ihre Urkundenbestandteile zu unterteilen und zu klassifizieren (WillkommenWerkstatt). Dies bedeutet allerdings einen erheblichen Zeitaufwand und setzt Fachwissen voraus. Ein Algorithmus, der eine verlässliche Unterteilung und Klassifizierung ermöglicht, würde daher eine erhebliche Arbeitserleichterung und Einsparung von Ressourcen bedeuten. Im Zuge einer Werkstudenten-Anstellung bestand die Aufgabenstellung darin, ein NLP- (Natural Language Processing) Modell zu entwickeln, welches diese Urkunden in Urkundenbestandteile (wie z.B. Anrede, Gegenstand der Schenkung, etc.) unterteilt und klassifiziert. Problematisch dabei ist, dass einige Klassen nur selten vorkommen und dadurch unterrepräsentiert im Training der künstlich intelligenten Algorithmen sind. Wenn konventionelle Algorithmen (SVM, RF, NN, LR) mit den vorhandenen Daten (lateinische, unterteilte und kategorisierte Schenkungsurkunden) getestet werden, erreichten diese keine zufriedenstellenden Ergebnisse. Tests wurden im Zuge der Vorbereitung auf das Projekt durchgeführt und Algorithmen ermittelt, die gute, jedoch nicht zufriedenstellende Ergebnisse ermitteln konnten. Von Anwenderseite wird eine accuracy von über 0.95 angestrebt.

Abbildung 1.1: Urkunde: Praefatio Paris BnF Lat. 2123 (P3) (UrkundePraefatio (2023))

Abbildung 1.2: Detailansicht Urkunde (UrkundePraefatio (2023))

1.2 Zielsetzung

Das Ziel besteht darin, Algorithmen zu entwickelt oder bestehende zu modifizieren, die trotz einer Trainingsphase mit Daten von geringer Menge, eine möglichst gute Klassifizierung der Texte erreichen können.

1.3 Abgrenzung des Themas

Diese Arbeit wird sich vorwiegend mit dem oben beschriebenen ersten Schritt (Klassifizierung), aber nur eingeschränkt mit dem zweiten Teilproblem (Unterteilung der Urkunden in ihre Urkundenbestandteile) befassen. Die Lösung des zweiten Teilproblems wird konzeptionell erläutert und die Ergebnisse werden komprimiert diskutiert.

1.4 Überblick über den Aufbau der Arbeit

Zuerst werden die Datensätze beschrieben und anschließend einige Arten der Feature Extraction erläutert. Im zweiten Schritt wird unter anderem eine Arbeit vorgestellt, die viele Klassifizierungsalgorithmen miteinander vergleicht. Im dritten Schritt wird eine Auswahl etablierter Algorithmen vorgestellt und begründet, warum diese für einen Vergleich mit dem neuen und mit den modifizierten Algorithmus ausgewählt wurden. Des Weiteren wird eine Betrachtung der neuen Lösung sowie der modifizierten Lösung erst auf konzeptioneller, dann auf detaillierter ebene erfolgen. Als Nächstes werden die ausgewählten Metriken und Methoden festgelegt, um im darauf folgenden Kapitel die Ergebnisse eines modifizierten Algorithmus sowie die eines neuen Algorithmus mit den Ergebnissen etablierter Algorithmen vergleichen zu können. Anschließend wird das Konzept des Algorithmus erläutert, um das Problem zu lösen, Urkundenbestandteile automatisiert in Urkunden ermitteln zu können, gefolgt von den Ergebnissen und deren Diskussion. Abschließend wird ein Fazit und ein Ausblick auf Verbesserungen oder Modifikationen erwogen.

2 Auswahl von Datensätzen und Preprocessing der Texte

Der bisherige Arbeitsablauf verläuft folgendermaßen: Lateinischen Urkunden wurden eingescannt, mittels Optical Character Recognition (OCR) in Text umgewandelt und auf Fehler überprüft. Anschließend hat eine Doktorandin diese Urkunden in Urkundenbestandteile unterteilt und klassifiziert (WieHamburg). Der Gegenstand meiner Arbeit bestand darin, ein System zu etablieren, das diesen letzten Arbeitsschritt übernimmt und Urkunden in Urkundenbestandteile unterteilt und klassifiziert. Dies soll anhand der von der Doktorandin vorgenommenen Unterteilung erlernt und getestet werden.

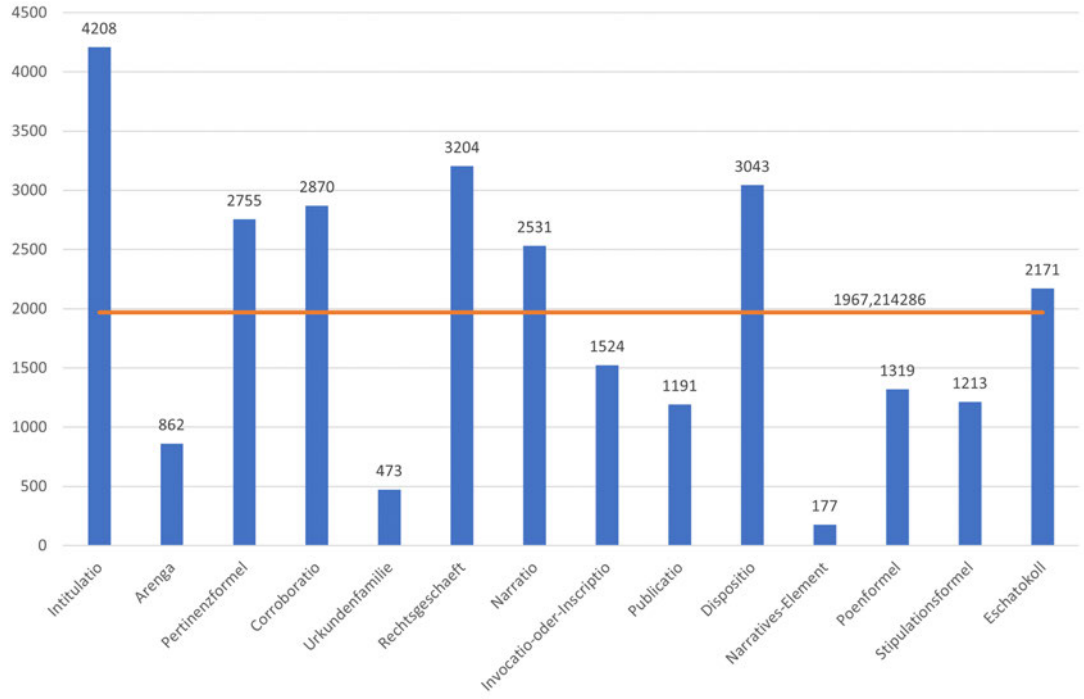

Abbildung 2.1: Anzahl der Texte je Urkundenklasse. Orange Linie zeigt die durchschnittliche Anzahl von Texten je Klasse. (UrkundePraefatio (2023))

Der verwendete lateinische Datensatz, der von der Doktorandin zur Verfügung gestellt wurde, umfasst 4900 Urkunden, die in 22.471 Texte unterteilt und 14 möglichen Urkundenbestandteilen zugeordnet wurden. Die Aufteilung der unterteilten Urkunden über die möglichen Urkundenbestandteile ist keineswegs gleichmäßig. So umfasst beispielsweise der Urkundenbestandteil 'Intitulatio' 4.208 Texte, 'Narratives-Element' aber nur 177 Texte. In orange ist der Wert gekennzeichnet, den jede Klasse haben müsste, um eine gleichmäßige Verteilung zu erreichen (Abb. 2.1).

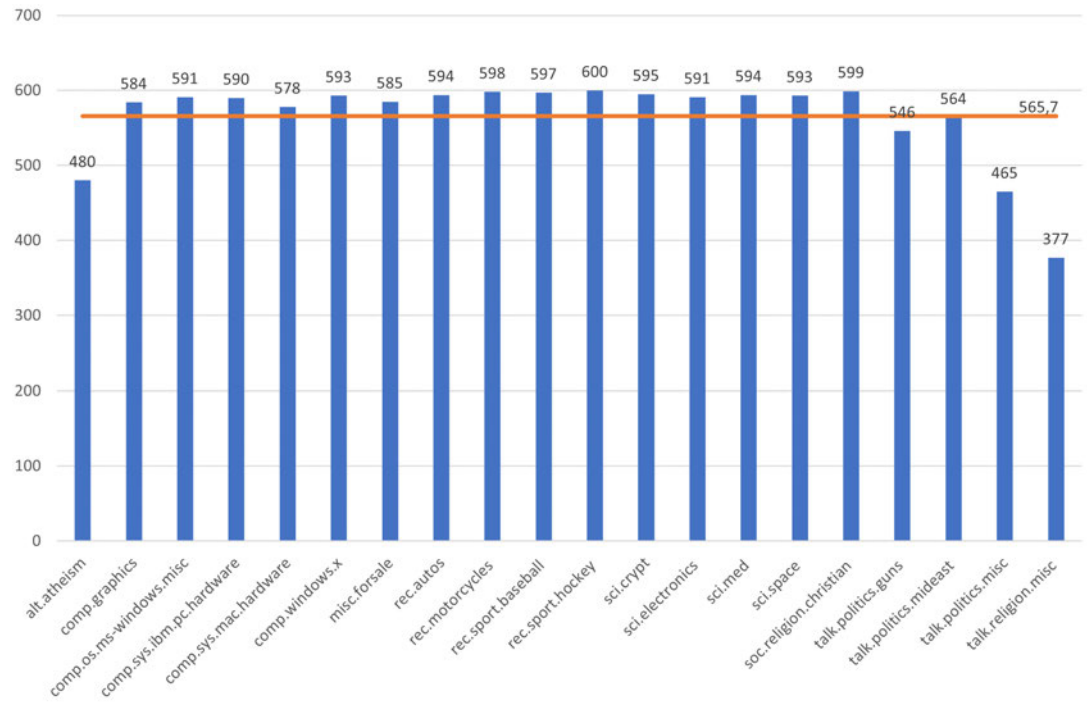

Abbildung 2.2: Anzahl der Texte je Newsgroup. Orange Linie zeigt die durchschnittliche Anzahl von Texten je Klasse. (HomeSet)

Als weiterer Datensatz wurde 'the 20 newsgroups dataset' (HomeSet; Abb. 2.2) gewählt, welcher über eine Anzahl von 11.314 Texten verfügt, die relativ gleichmäßig auf 20 Klassen verteilt sind. Der Datensatz besteht aus Texten verschiedener Nachrichtenressorts in englischer Sprache. Dieser zweite Datensatz wurde gewählt, um die Güte des neu entwickelten Algorithmus auf einem Datensatz zu testen, der über eine homogene Verteilung der Texte über dessen Klassen verfügt. In orange ist wieder der Wert eingezeichnet, den jede Klasse haben müsste, um eine komplette Gleichverteilung der Texte zu erreichen. Außerdem ist dieser Datensatz in der Wissenschaft häufig verwendet worden und erhöht somit die Vergleichbarkeit der hier vorgestellten Ergebnisse.

2.1 Wordprocessing

Damit die Worte aus einem Text von den Systemen verarbeitet werden können, wird als erstes 'Wordprocessing' (Datenaufbereitung) durchgeführt. Diese Verfahren besteht in diesem Fall

aus dem Erarbeiten von Tokens aus Texten. Dabei wird ein Text in seine Worte und Satzzeichen unterteilt (tokenizing). Als Nächstes werden die Wortstämme der Tokens gesucht. Dieses Verfahren wird 'Stemming' oder auch 'Lemmatizing' genannt. Zuletzt werden die 'Stopwords' entfernt. Dabei werden Worte wie Pronomen, Füll-Wörter etc., welche unwichtig für die Analyse sind, entfernt. Das Generieren der Tokens aus dem Text wird mit dem Python-Paket 'nltk' (NLTKPackage) durchgeführt. Die englischen und lateinischen 'Stopwords' werden vom Paket 'stopwords' (stopwords) bereitgestellt, und das 'stemming' erfolgt über das Paket 'simplemma' (simplemma), um einen Lemmatizer zu nutzen, der multilingual ist, und dadurch sowohl für lateinische als auch für englische Texte anwendbar ist. Stopwords im Lateinischen sind jedoch sehr beschränkt. In dieser Bibliothek existieren gerade einmal 49 'Stopwords' (StopwordsLatin), in der englischen Bibliothek sind es dagegen 174 Wörter (StopwordsEnglish).

2.2 Feature-Extraction

Für einige der konventionellen Algorithmen reichen diese preprocessing Maßnahmen jedoch nicht aus. Im ersten Schritt müssen in diesen Fällen die Texte in Vektoren oder Tokens unterteilt werden. Dies wird anhand verschiedener Algorithmen (Term Frequency and Inverse Document Frequency, Bag Of Words, Tokenizierung in Sequenzen) durchgeführt, welche als Älgorithmen der Feature-Extractionßusammengefasst werden können. Die ermittelten Vektoren oder Tokens werden als Features bezeichnet.

2.2.1 TF-IDF

TF-IDF steht für Term Frequency (TF) and Inverse Document Frequency (IDF) und ist ein Verfahren, Features aus Texten zu extrahieren. Dieser Algorithmus berücksichtigt keinen Wortkontext. TF-IDF legt dabei einen besonderen Wert darauf, zu ermitteln wie wichtig ein bestimmtes Wort in einem Dokument für die richtige Klassifizierung ist. Für diese Feature-Extraction werden die Menge der Dokumente D, das zu betrachtende Wort w und das aktuelle Dokument d, welches aus der Menge der Dokumente D stammen muss, benötigt.

$$w_d = f_{w,d} * log(|D|/f_{w,D}) \tag{2.1}$$

$f_{w,d}$ beschreibt, wie häufig das Wort w im Dokument d vorkommt. $f_{w,D}$ beschreibt, in wie vielen Dokumenten aus D das Wort w vorkommt (Form. 2.1) Ramos (2003).

2.2.2 Bag Of Words & CountVectorizer

Das Bag Of Words Modell ist ein sehr simples und intuitives Verfahren, um aus Daten wie Texten Merkmale für eine Klassifizierung zu extrahieren. Das Verfahren betrachtet dabei die Häufigkeit eines Wortes in einer bestimmten Klasse (Term Frequency (TF)).

Dafür wird ein Wörterbuch anhand der Trainingsdaten aufgebaut, in dem alle Wörter enthalten ist. Anschließend wird anhand dieses Wörterbuchs eine Liste erstellt, die dieselbe Länge wie das Wörterbuch hat und in der für jedes Wort die TF gespeichert wird. (George und Joseph (2014)).

2.2.3 Tokenizieren in Sequenzen

Das Tokenizieren in Sequenzen ist das einzige der vorgestellten Verfahren, bei dem der Kontext der Wörter nicht verloren geht. Bei diesem Verfahren werden Wörter der Texte in ein Wörterbuch geschrieben und Listen erstellt. Die Wörter in Texten werden durch ihre Indices ersetzt. Die Merkmale, die daraus entstehen, beinhalten Zusammenhänge von Wörtern in Texten. Ein Tokenizer, der diese Art von Tokenizieren in Sequenzen bereitstellt, ist der Tokenizer aus dem Paket Tf.keras.preprocessing.text.Tokenizer.

3 Grundlagen und Auswahl Algorithmen

3.1 Verwandte, wissenschaftliche Arbeiten

Wie oben beschrieben, wurden zu Beginn des Projektes auf etablierten Algorithmen Tests mit den Datensätzen angefertigt. Diese Algorithmen wurden aufgrund ihrer Verbreitung im Anwendungsgebiet Textklassifizierung ausgewählt.

A Survey Report on Text Classification with Different Term Weighing Methods and Comparison between Classification Algorithms - Singh und Patra (2013): Diese Arbeit untersucht die Abhängigkeit zwischen Performance und Modell. Die Performance wird in dieser Arbeit in 'precision' und 'recall' gemessen. Ein besonderer Fokus der Arbeit ist die Untersuchung der Kombinationen aus 'Termgewichtungsmethoden' und der Klassifizierungsalgorithmen. Untersucht wurden: 'K-nearest neighbor', 'Naïve bayes', 'SVM', 'NN' und 'DT' mit sowohl überwachten als auch nicht-überwachten Termgewichtungsmethoden. Zu den nicht-überwachten Termgewichtungsmethoden zählen die auch in dieser Arbeit häufig angewandten TF und IDF (siehe dazu Abschnitt 2.2.1). Die Autoren dieser Arbeit kommen zu dem Ergebnis, dass keine Datenrepräsentierungsmethode und kein Algorithmus als generelles Modell für alle Anwendungsfälle verwendet werden kann, sondern die Auswahl dieser Bausteine immer anhand des zu lösenden Problems und der zugrunde liegenden Daten getroffen werden muss.

Using logistic regression method to classify tweets into the selected topics - St u. a. (2016): In der Arbeit 'Using logistic regression method to classify tweets into the selected topics' wird mittels logistsischer Regression eine Klassifizierung von Tweets auf dem Sozialen Netzwerk Twitter (heute 'X') durchgeführt. Als Metrik wird die 'accuracy' gewählt, welche bei einer Menge von 9000 Tweets 1800 Tweets mit einer 'accuracy' von 93% korrekt zuordnete. Diese Arbeit begründet die Auswahl der Logistischen Regression als Basis-Algorithmus für die Erweiterung eines kontextlosen Algorithmus um eine Kontextvariable.

Text Classification Algorithms: A Survey - Kowsari u. a. (2019): In der Arbeit 'Text Classification Algorithms: A Survey' wurden verschiedene Klassifizierungsalgorithmen miteinander vergleichen. Um eine möglichst präzise Vorhersagen zu erreichen, wurde in dieser Arbeit ein starker Fokus auf das Preprocessing der Daten gelegt. Dieses Preprocessing beinhaltetet unter anderem das Bereinigen der Daten, wie zum Beispiel das Löschen von Stopwörtern, das Ersetzten von Großbuchstaben durch kleine Buchstaben und das Suchen des Wortstamms (Lemmatisierung). Die Ergebnisse dieser Arbeit sind in einem öffentlichen GitHub Repository einzusehen. (Kowsari K (Tab. 3.1))

Algorithmus	accuracy	support
Convolutional Neural Networks (CNN)	0.76	7532
Neural Networks (NN)	0.82	7532
Support Vector Machine (SVM)	0.85	7532
Decision Tree (DT)	0.55	7532
Random Forest (RF)	0.77	7532

Tabelle 3.1: Auszug der Ergebnisse des GitHub Repositorys Kowsari K

Bei diesen Ergebnissen handelt es sich um Tests auf dem Datensatz '20 Newsgroups', bei denen der gesamte Datensatz mit allen möglichen Klassen ausgewählt wurden. Die besten Ergebnisse erreichten in dieser Arbeit die SVM und die neuronalen Netzwerke. Diese SVM benutzt als Vektorisierer ausschließlich den CountVectorizer aus dem Paket 'scikit-learn', welcher keinen Wortkontext betrachtet (SklearnCountVectorizer). Algorithmen, die den Wortkontext berücksichtigen, erreichten in dieser Arbeit schlechtere Ergebnisse als kontextlose Algorithmen. So kann bei den Ergebnissen des NN eine bessere Präzision erreicht werden, als bei dem CNN, welches eine Modifikation um eine Kontextbetrachtung des NN ist.

A Survey on Text Classification Algorithms: From Text to Predictions - Gasparetto u. a. (2022): Eine der aktuellsten und umfangreichsten Arbeiten ist die Arbeit 'A Survey on Text Classification Algorithms: From Text to Predictions'. Sie beschäftigt sich mit allen relevanten Text Klassifizierungs-Algorithmen. Diese Arbeit bildet ein sehr viel breiteres und detaillierteres Bild der Klassifizierungslandschaft ab und untersucht die Modelle auf viele verschiedene Aufgabenbereiche wie Stimmungsanalyse, Themenbeschriftung, Named Entity Recognition, News Klassifikation, uvm. Hierfür wurden viele konventionelle Algorithmen verwendet, wie SVM, DT, LR, RF, K-NNN und Naïve Bayes, zusätzlich State-Of-The-Art Transformer Model-

le wie BERT. Auch mit vielen verschiedenen Datensätze wurde gearbeitet: exemplarisch zu erwähnen wären die 20 News, IMDb und AG News.

Test set Macro-F_1 and Subset Accuracy (%) on synthesized datasets (best results in bold).

	F_1-Score		Accuracy	
Model	**EnWiki-100**	**RCV1-57**	**EnWiki-100**	**RCV1-57**
Naïve Bayes (OVA)	63.64	56.30	39.24	47.27
Linear SVM (OVA)	80.29	71.73	66.93	67.67
FastText Classifier	74.45	69.65	68.23	67.02
BiLSTM (GloVe)	81.22	76.62	68.05	72.48
XML-CNN (FastText)	78.19	73.02	66.64	70.98
BERT (base)	**85.52**	**78.07**	**75.28**	73.48
XLM-R (base)	84.60	77.19	74.25	**74.01**

Tabelle 3.2: Macro-F1 und Accuracy Scores beim Ermitteln von Themenbeschriftung auf den Datensets 'EnWiki-100' und RCV1-57 (Gasparetto u. a. (2022))

In Tabelle 3.2 werden die Ergebnisse der Themenbeschriftung gezeigt. Das BERT Modell zeigt, gemessen am F1-Score, eine deutliche bessere Performance sowohl für EnWiki-100 (85,52) als auch für den RCV1-57 (78,07) Datensatz.
Anschließend wurde noch ein zweiter Test der verschiedenen Transformer und nicht Tranformer Modelle mit News Klassifizierungsaufgaben durchgeführt. Hierfür wurde unter anderem das Dataset der 20 News verwendet.
Auch in Tabelle 3.3 dominierten Transformer Modelle die Tests.

Test accuracy score (%) on NC and TL benchmark datasets (best results in bold).

	NC				TL			
Model	**AG**	**20N**	**R52**	**R8**	**Yah!A**	**TREC**	**Ohs**	**DBP**
XLNet [19]	**95.55**	-	-	-	-	-	-	**99.40**
BERT-base [100,132]	95.20	85.30	96.40	97.80	**78.14**	**98.20**	70.50	99.35
BERT-large [132]	95.34	-	-	-	-	-	-	99.39
L-MIXED [135]	95.05	-	-	-	-	-	-	99.30
DNC + CUW [136]	93.90	-	-	-	74.30	-	-	99.00
SVDCNN [74]	90.55	-	-	-	-	-	-	-
ULMFiT (small-data) [137]	93.70	-	-	-	74.30	-	-	99.20
USE_T + CNN [138]	-	-	-	-	-	97.44	-	-
MPAD [101]	-	-	-	97.57	-	95.60	-	-
STM + TSED + PT + 2L [139]	-	-	-	-	-	93.48	-	-
DELTA [155]	-	-	-	-	75.10	92.20	-	
VLAWE [140]	-	-	-	89.30	-	94.20	-	-
TM [141]	-	-	89.14	97.50	-	90.04	-	-
HAN [80]	-	-	-	-	75.80	-	-	-
SSGC [107]	-	88.60	94.50	97.40	-	-	68.50	-
SGCN [104]	-	88.50	94.00	97.20	-	-	68.50	-
TextGCN [98]	-	86.34	93.56	97.07	-	-	68.36	-
GCN [98]	-	87.90	93.80	97.00	-	-	68.20	-
NABoE-full	-	86.80	-	97.10	-	-	-	-
RMDL (15 m) [146]	-	87.91	-	-	-	-	-	-
GraphStar [147]	-	86.90	95.00	97.40	-	-	64.20	-
SCDV-MS [156]	-	86.19	-	-	-	-	-	-
STC-Q [157]	-	87.30	-	-	-	-	-	-
BertGCN [100]	-	89.30	**96.60**	98.10	-	-	**72.80**	-
RoBERTaGCN [100]	-	**89.50**	96.10	**98.20**	-	-	**72.80**	-
RepSet	-	77.02	-	96.85	-	-	66.12	-
Rel-RWMD kNN	-	74.78	-	95.61	-	-	58.74	-
DRNN [62]	94.47	-	-	-	-	-	-	99.19
ULMFiT [61]	94.99	-	-	-	-	96.40	-	99.20
DPCNN [148]	93.13	-	-	-	76.10	-	-	99.12
CCCapsNet [150]	92.39	-	-	-	73.85	-	-	98.72
CharCNN [125]	91.45	-	-	-	71.20	-	-	98.63
EFL [151]	86.10	-	-	-	-	80.90	-	-
byte mLSTM [152]	-	-	-	-	-	90.40	-	-
BLSTM-2DCNN [153]	-	-	-	-	-	96.10	-	-
oh-2LSTMp [154]	-	86.68	-	-	-	-	-	-
VDCNN [73]	91.33	-	-	-	73.43	-	-	98.71
RoBERTa [100]	-	83.80	96.20	97.80	-	-	70.70	-

Die betrachteten wissenschaftlichen Arbeiten trugen zur Auswahl der im nächsten Abschnitt betrachteten Algorithmen bei.

3.2 Ausgewählte Algorithmen und ihre Methodik

Es gibt viele Algorithmen, die Klassifizierungen durchführen können. In dieser Arbeit werden Besonderheiten einiger dieser Algorithmen erklärt und begründet, weshalb diese sowohl für einen Vergleich mit dem modifizierten (Logistische Regression mit Kontexterweiterung LRC) als auch dem neu entwickelten Algorithmus (Context-Classifyer CC) ausgewählt wurden. Einige dieser Algorithmen sind besonders zuverlässig und häufig genutzt auf dem Gebiet der Textklassifizierung (Transformer, SVM), andere haben in vorangegangenen Tests besonders gute Ergebnisse erreicht (LR, SVM, Transformer). Wieder andere haben einen starken Fokus auf Kontext/Aufmerksamkeiten und sind daher interessant um mit den neuen Algorithmen verglichen zu werden (CNN, Transfomer). Einige sind besonders schnell (SVM, DT, RF) oder die Grundlagen anderen Algorithmen (DT, NN) (Tab. 3.4).

Algorithmus	Auswahlkriterium
LR	Starker Klassifizierungsalgorithmus in vorangegangenen Tests
SVM	Bensonders gut für geringe Datenmenge (Sivakami (2018))
CNN	Starker Fokus auf Kontext durch Conv. Layers
RF	wenig rechenintesiv
Transformer	Für sequenzielle Daten geeignet & spezieller Fokus auf Kontext durch Aufmerksamkeiten

Tabelle 3.4: Übersicht über gewählte Klassifizierungsalgorithmen und deren Begründungen - LR Logistische Regression, SVM Support Vektor Mashine, CNN Conolutional Neuronal Network, RF Random Forest

3.2.1 Logistische Regression

Logistische Regression ist ein Verfahren, Texte anhand von Merkmalen Klassen zuzuordnen. Multinomiale logistische Regression wird für diesen Fall genutzt, da es sich um eine nicht binäre Klassifikation handelt (Kwak und Clayton-Matthews (2002)). Eine Regression (Form. 3.1) ist das Verrechnen von ermittelten Intensitäten (β_x) von Features (f_x) mit einem Faktor, der die Wichtigkeit (α) von diesem bestimmten Feature repräsentiert. Diese (β_x) Werte werden auch Regressionskoeffizienten genannt (Rawlings u. a. (1998)).

$$reg(\alpha, \beta, f) = \alpha + \beta_1 * f_1 + \beta_2 * f_2 + \cdots + \beta_k * f_k \tag{3.1}$$

Diese Wichtigkeit von einem Feature wird anhand des Maximum-Likelyhood Verfahrens in der Trainingsphase ermittelt. Eine logistische Regression entsteht, wenn diese Regression mit einer logistischen Funktion (Form. 3.2) verbunden wird.

$$log(z) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x} \tag{3.2}$$

Eine logistische Regression (Form. 3.3) beschreibt die Wahrscheinlichkeit, mit der bestimmte Merkmalskombinationen in einer Klasse liegen, da sich durch die logistische Funktion Werte in den Extremen 1 und 0 annähern.

$$log(\beta, f) = \frac{exp(\beta_0 * f_0 + \beta_1 * f_1 + \cdots + \beta_k * f_k)}{1 + exp(\beta_0 * f_0 + \beta_1 * f_1 + \cdots + \beta_k * f_k)} \tag{3.3}$$

Für eine multinomiale logistische Regression, die in diesem Beispiel benötigt wird, wird die Regression der zu ermittelnden Klasse r exponenziert (exp) und durch 1 + die Summe aller exponenzierten Regressionen aller Klassen geteilt (El-Habil (2012)) (Form. 3.4).

$$log(z) = \frac{exp(\beta_0 r * f_0 + \beta_1 r * f_1 + \cdots + \beta_k r * f_k)}{1 + \sum_{s=1}^{c} exp(\beta_0 s * f_0 + \beta_1 s * f_1 + \cdots + \beta_k s * f_k)} \tag{3.4}$$

So wird für jede Klasse die Wahrscheinlichkeit errechnet, dass das betrachtete Element dieser spezifischen Klasse angehört. Alle Wahrscheinlichkeiten der Klassen ergeben addiert 1.

3.2.2 SVM

Support Vector Machines (SVM) sind ein Algorithmus zum Klassifizieren von Daten, die häufig zur Textklassifizierung verwendet werden (Tong und Koller (2001), Uchenna Oghenekaro und Benson (2022), Sivakami (2018)). Diese eignen sich besonders gut für Daten geringer Menge (Sivakami (2018)). Eine SVM funktioniert, indem Daten erst mittels 'Feature Extraktion' vektori-

siert werden und dann in eine mehrdimensionale Matrix geladen werden, in der die Anzahl der Dimensionen der Anzahl der extrahierten Features entspricht. Diese Vektorisierung kann auf mehrere Arten durchgeführt werden. Es existieren Algorithmen wie der Bag Of Words Algorithmus, die diese Daten losgelöst von ihrem Kontext vektorisieren (Nitsche und Tropmann-Frick (2020)). Außerdem gibt es auch Modelle wie 'Continuous Bag-Of-Words' (CBOW) (Nitsche und Tropmann-Frick (2020)), welche diese Vektorisierung vornehmem und den Kontext der Wörter berücksichtigen. Anschließend errechnet der Algorithmus Hyperplanes in den Raum, welche die Klassen voneinander trennen sollen. Als Supportvektoren werden die nächsten Datenpunkte an der Hyperplane bezeichnet (Abb. 3.1). Von diesen Datenpunkten ist die Position und die Ausrichtung der Hyperplane abhängig. Eine Idee ist, den Abstand zwischen den Datenpunkten und der Hyperplane zu maximieren. Der Abstand der Suportvektoren zu der Hyperplane wird als 'Margin' bezeichnet (Tong und Koller (2001)). Beispielsweise erhält man für eine binäre Klassifikation durch eine SVM einen Wert zwischen -1 und $+1$.

Multi-Klassen-Klassifikation: Gibt es mehrere Klassen, die vorhergesagt werden können, gibt es Verfahren, welche Multi-Klassen-Klassifikationen angehen. Diese Verfahren sind in folgende Konzepte unterteilbar (Galar u. a. (2011)):

1. One-vs-One
2. One-vs-Many/One-vs-All

Bei der **One-vs-One** Klassifizierung wird für jede Klassenpaarkombination eine SVM erlernt, die zwischen diesen beiden unterscheiden kann. Anschließend werden diese Ergebnisse kombiniert und als resultierende Klasse zurückgegeben.
Das **One-vs-Many/One-vs-All** Modell erlernt für jede mögliche Klasse eine 'Klasse X gegen alle andere Klassen' SVM. Diese Ergebnisse werden wieder kombiniert und als resultierende Klasse zurückgegeben (Galar u. a. (2011)).
SVM's gibt es in linearer und in nicht linearer Form.

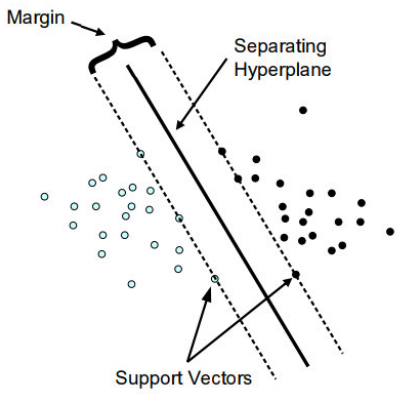

Abbildung 3.1: Support Vectors (Meyer (2012))

Lineare SVM: Bei einer linearen SVM werden lineare Funktionen zur Definition der Hyperplanes genutzt. Eine Klassifizierung mit einer solchen Funktion kann nicht so flexibel wie die nicht lineare SVM an die Trainingsdaten angepasst werden, da die Klassifizierung durch die Linearität der Funktion eingeschränkt wird (Mammone u. a. (2009)). Die Darstellung einer zwei dimensionalen, linearen SVM ist in Abbildung 3.1 zu erkennen. Linare SVM können besonders gut verwendet werden, wenn eine lineare Abhängigkeit zwischen Features angenommen wird. Somit beeinflussen Ausreißerwerte nur geringfügig die Ausrichtung der Hyperplane und ein Overfitting dieser Werte kann vermieden werden.

Nicht lineare SVM: Eine nicht lineare SVM bedient sich nicht linearer Funktionen zur Bestimmung der Hyperplanes (Abb. 3.2), und damit auch zum Separieren der Klassen. Durch eine nicht lineare Funktion ist eine deutlich flexiblere Anpassung der Hyperplanes an die Daten möglich (Mammone u. a. (2009)).

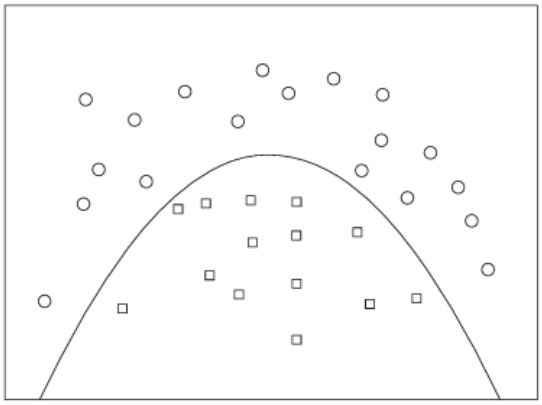

Abbildung 3.2: Nicht lineare SVM (Mammone u. a. (2009))

3.2.3 CNN

Ein Convolutional Neural Network (CNN) ist ein künstliches neuronales Netzwerk, das über zusätzliche, vorgeschaltete Schichten (Layer) verfügt. Diese Schichten sind abwechselnd Convolutional Layer und Pooling Layer. Im Preprocessing der Daten ist es wichtig, dass nicht nur eine Vektorisierung auf TF-IDF oder nach dem BOW Model erfolgt, sondern der Wortkontext erhalten bleibt. Dies wird anhand der Tokenizierung in Sequenzen realisiert. Anschließend erfolgt eine Schicht, die für das 'flatten' zuständig ist. Dieser Vorgang passt die Ergebnisse der vorherigen Schicht auf die Anzahl und Form der Eingabeneuronen der nächsten Schicht an. Die letzte Schicht ist ein vollständig verbundenes neuronales Netz, welches die möglichen Klassen als Ausgabe-Neuronen hat (Abb. 3.3) (Kowsari u. a. (2019)).

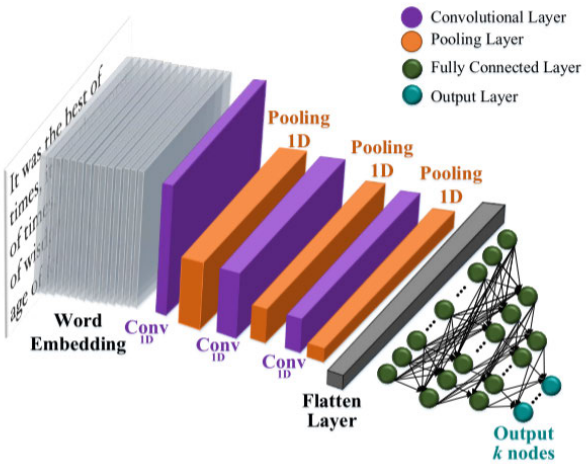

Abbildung 3.3: CNN für Text Klassifizierung (Kowsari u. a. (2019))

Convolutional Layer: Als Convolutional Layer bezeichnet man eine bestimmte Art von Schicht in einem künstlichen neuronalen Netz, welche sich auf das Zusammenfassen benachbarter Eingangsneuronen konzentriert (falten). Dafür wird von bestimmten Filtern Gebrauch gemacht, die in der Trainingsphase trainiert werden. Diese Filter laufen beispielsweise bei zweidimensionalen Eingabematrizen wie einfache Schwarzweißbilder als $N \times N$ Matrix über die Eingangsneuronen und finden Muster. Diese Filtermatrizen weisen eine ungerade Länge und Breite auf.

Die Idee ist, dass die Filter Pixel für Pixel über die Eingabeneuronen gelegt werden und die Werte aus der Filtermatrix mit den darunterliegenden Eingabeneuronen multipliziert

werden (Form. 3.6). Die errechneten Produkte werden anschließend aufsummiert und in einer 'featuremap' festgehalten (Form. 3.5).

$filter$ = Filtermatrix
$input$ = Eingabenneuronen-Matrix
$output$ = featuremap
$l(X)$ = Länge einer Matrix X
$b(X)$ = Breite einer Matrix X
σ = sigmoid-Funktion
β = bias

$$\sum_{x=0}^{l(input)} \sum_{y=0}^{b(input)} output_{x,y} = auspraegung(x,y) \tag{3.5}$$

$$auspraegung(x,y) = \sigma\left(\beta + \sum_{i=0}^{l(filter)} \sum_{j=0}^{b(filter)} input_{x-(l(filter)/2)+i,y-(b(filter)/2)+j} * filter_{j,j}\right) \tag{3.6}$$

In der Abbildung 3.4 wird die Berechnung eines Featurewerts dargestellt. Hierfür wird ein Auszug einer Inputmatrix (links) mit einem Filter (mitte) betrachtet, der eine vertikale Linie in einem zweidimensionalen Bild erkennen kann. Rechts steht der berechnete Wert für die Ausprägung dieses Features.

Dieses Verfahren kann beliebig oft für jeden erlernten Filter wiederholt werden. Daraus resultieren mehrere featuremaps, die auch in unserem Beispiel als $M \times M \times F$ Matrix betrachtet werden können, in der die F-Dimension die Anzahl der featuremaps der Maße $M \times M$ sind. Die Länge und Breite der errechneten featuremap kann von der Eingabelänge der Matrix abweichen. Dies geschieht allerdings nur, wenn keine Paddingmethode verwendet wird. Mit dem 'padding' soll erreicht werden, dass die Länge und Breite der Matrix gleich bleibt. Das bedeutet, dass eine 64×64 Neuronenmatrix als Eingabe auch eine 64×64 Matrix als featuremap ausgibt. Dies wird erreicht, indem weitere imaginäre Eingabeneuronen um die Inputmatrix erzeugt wird. Dieses padding hat eine Breite von der Hälfte der Länge und Breite der Filtermatrix. Dies hat zum Zweck, dass die gesamte Berechnung durchgeführt werden kann, wenn die Filtermatrix über die Eingabeneuronen gelegt wird. Beim Zeropadding werden die padding Werte auf 0 gesetzt. Das Samepadding spiegelt die äußeren Werte in der Eingabematrix.

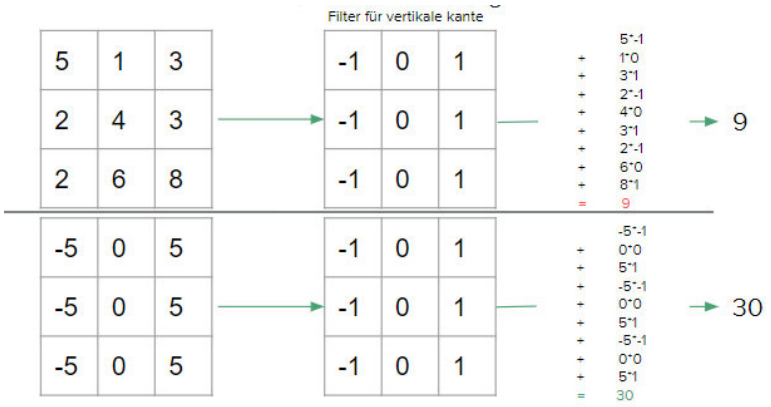

Abbildung 3.4: Der Faltprozess

Pooling: Pooling beschreibt den Prozess, Eingabeneuronen nach dem Falten zusammenzufassen, mit dem Ziel, so viele Informationen wie möglich zu erhalten. Dies kann auf verschiedene Arten erfolgen. So gibt es beispielsweise das 'Average-Pooling' oder auch das 'Max-Pooling'. Pooling definiert die Dimensionen der Eingabeneuronen N der nächsten Schicht im CNN. Wenn ein $N \times N$ Pooling durchgeführt wird, wird aus den Outputneuronen der Convolutional Layer ein $N \times N$ Feld ausgewählt und zusammengefasst. Bei einem 'Average-Pooling' wird der Durchschnitt aus allen Werten gebildet und zurückgegeben (Abb. 3.6), beim 'Max-Pooling' wird der größte Wert zurückgegeben (Abb. 3.5). Der gängigste und meistverbreitete Algorithmus zum Pooling ist das 'Max-Pooling' (Kowsari u. a. (2019)).

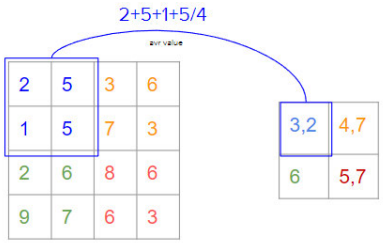

Abbildung 3.5: Average-Pooling

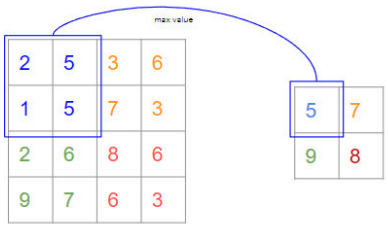

Abbildung 3.6: Max-Pooling

3.2.4 Random Forest

Um den Random Forest (RF) Algorithmus zu verstehen, muss erst das zugrundeliegende Konzept des Decision Tree Models (DT) verstanden werden.

Grundlagen des Decision Tree Model: Dieser Algorithmus löst eine Klassifizierung anhand eines binären Entscheidungsbaumes. Der Baum wird mittels seiner Trainingsdaten gebildet und orientiert sich dabei daran, Unterteilungen in jedem Knoten zu machen. Damit soll eine Minimierung der Entropie erreicht werden. Entropie ist der Zustand, wenn ein Datensatz eine stark heterogene Menge ist, also eine Menge, die viele Daten verschiedener Klassen beinhaltet. Die Entropie kann folgendermaßen errechnet werden:

$$-\sum_{i=1}^{m} p_i * log(p_i) \tag{3.7}$$

p zeigt die Wahrscheinlichkeit der Klasse i im aktuell betrachteten Datensatz und m ist die Anzahl der Klassen (Form. 3.7) (Kingsford und Salzberg (2008)). Dieser Lernprozess ist ein 'greedy' Algorithmus und sucht sich immer das aktuell beste Ergebnis. Das bedeutet Entscheidungen, die im Wurzelknoten oder in der 1. Ebene gemacht wurden, können nicht mittels backtracking in der 3. oder 4. Ebene des Baumes geändert werden, da dies einen komplett neuen DT ergeben würde. Außerdem würde der Algorithmus in seiner Form diese Entscheidung niemals treffen, da sie zu einer temporär höheren Entropie führen würde.

Random Forest: Der Random Forest Algorithmus beschreibt eine Modifikation des DT. Die Besonderheit ist, dass der Algorithmus mehrere DTs generiert. Dafür erstellt der Algorithmus mehrere Teilmengen aus dem eigentlichen Trainingsdatenset, welche 'Bootstrap-Datensets' genannt werden und trainiert damit eigenständige DT. Der Grund hierfür liegt darin, dass DT oft dazu neigen nicht generalisiert zu sein. Das bedeutet, dass sie nicht gut anwendbar auf neue Datensätze sind und ausschließlich einige wenige Merkmale lernt. Außerdem lernen jede DTs im Random Forrest auch ein zufällig selektiertes 'featureset'.
Jeder DT errechnet anhand des neuen Datensatzes eine eigenständige Vorhersage und anschließend wird mittels gleich gewichtetem Voting die Klasse ausgewählt, welche am häufigsten bestimmt wurde. Diesen Vorgang nennt man Aggregation (Cutler u. a. (2012)).

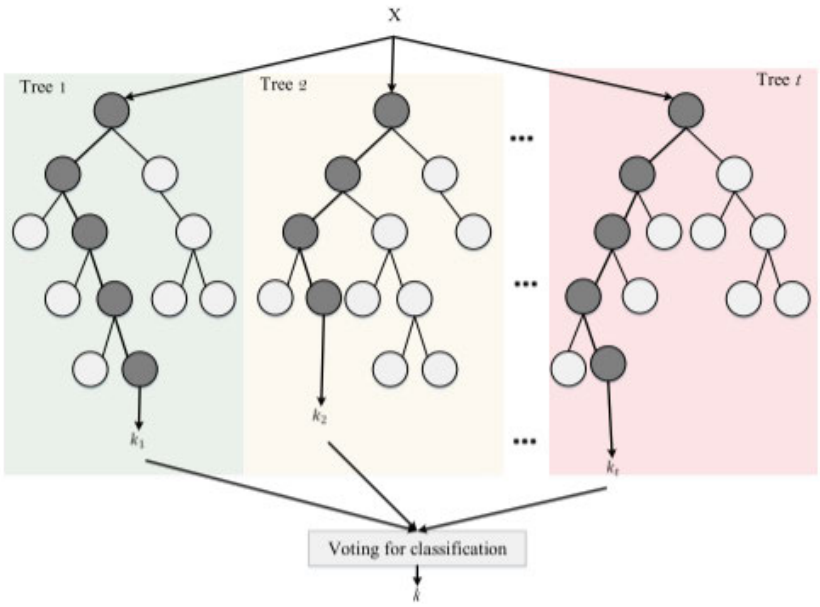

Abbildung 3.7: Random Forest Model (Kowsari u. a. (2019))

3.2.5 Transformer

Das Transformer Modell (Abb. 3.8) ist das modernste der betrachteten NLP-Modelle, das für das Klassifizieren von Texten genutzt werden kann. Dieses Modell legt einen großen Wert auf den Kontext, da es eine 'attention layer' aufweist, in dem Aufmerksamkeiten trainiert und berücksichtigt werden.

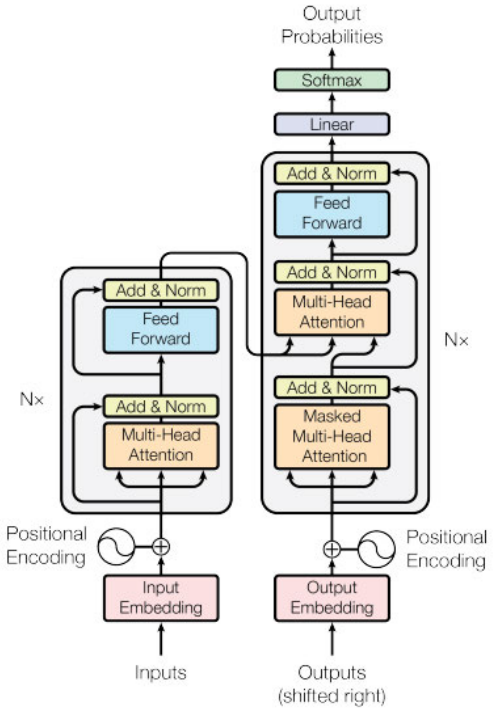

Abbildung 3.8: Transformer Model (Vaswani u. a. (2017))

Das Modell besteht aus einem Decoder und einem Encoder. Die linke Hälfte der Abbildung 3.8 zeigt den Encoder, die rechte Hälfte den Decoder. Der Encoder ist zusammengesetzt aus einer 'multihead attention layer' und einem voll verbundenen 'feed foreward' Netzwerk. Eine Aufmerksamkeitsschicht (attention layer) erfüllt den Zweck, die Wörter der Eingabesequenz unterschiedlich zu gewichten. Wörter mit einem hohen Aufmerksamkeit-Wert werden besonders stark gewichtet und Wörter mit einem niedrigen Wert eher schwach. In Abbildung 3.9 wurde eine Aufmerksamkeit erlernt, die den korrekten Zusammenhang zwischen 'the' und 'animal' erkennen kann.

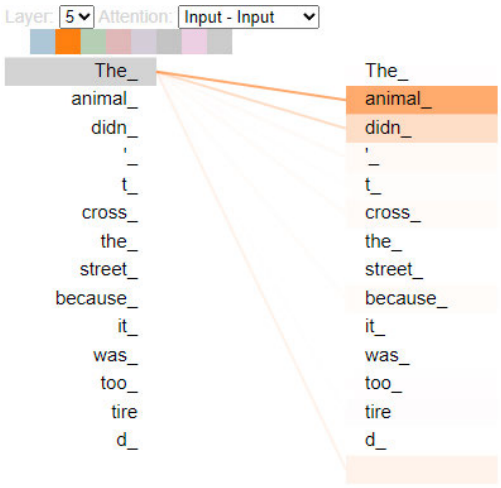

Abbildung 3.9: Attention (Tensor2TensorColaboratory)

Jeder Encoder kann dabei auf alle Positionen (Repräsentation von Wörtern in Vektoren) des vorherigen Encoders zugreifen, sodass ein Kontext erkannt werden kann. 'Multihead attention layer' betrachten dabei mehrere Aufmerksamkeiten und kombinieren diese Ergebnisse anschließend miteinander. Nachdem die Eingabe durch diesen Prozess verarbeitet wurde, wird ein Vektor errechnet, der die Eingabesequenz repräsentiert. Dieser Vektor ist zusammen mit dem vorherigen Output des Decoders die Eingabe des Decoder-Blocks. Dieser errechnet mit dem komprimierten Vektor des Encoders (Verständnis über den Satz) und den vorherigen Ausgaben (Kontext des Ausgabewertes bezeihungsweise des Satzes) die Ergebnisse (Vaswani u. a. (2017)). Sobald in den Analysen der Transformer eingesetzt wird, wurde der 'Bidirectional Encoder Representations from Transformers', kurz BERT, verwendet.

3.2.6 BERT

Der 'Bidirectional Encoder Representations from Transformers, kurz BERT, ist ein Encoder-Only Transformermodell und eignet sich daher besonders für Klassifikationsprobleme (Devlin u. a. (2018)). Aus diesem Grund wurde es auch für diesen Vergleich ausgewählt.
Für den Prozess, die Algorithmen zu vergleichen, wurden zwei vortrainierte BERT Modelle verwendet. Einmal der 'distilbert-base-uncased' (Distilbert-base-uncased) für Texte in der englischen Sprache (20 Newsgroups) und der 'simple-latin-bert-uncased' (Simple-latin-bert-uncased) für lateinische Texte (mittelalterliche Urkunden).

Pre-Training: Das Ziel vom Pretraining ist, ein generalisiertes Modell zu erlernen, das anschließend mittels finetuning auf bestimmte Daten angepasst werden kann. Dies spart Zeit und Ressourcen. Dieses Ziel wird in BERT-Modell durch zwei Methoden erreicht, welche im Folgenden beschrieben werden:

Masked Language Model: Beide dieser Modelle ('distilbert-base-uncased', 'simple-latin-bert-uncased') sind 'uncased' BERT Algorithmen. 'Uncased' bedeutet, dass keinen Unterschied zwischen kleinen und großen Buchstaben gemacht wird. BERT ist ein von Google entwickeltes Transformermodell, das ein Modell des 'Masked Language Modeling' (MLM) ist. Dieses Modell wurde mit maskierten Trainingsdaten erlernt. Das bedeutet, dass in jedem Trainingsdatensatz 15% der Wörter durch das [MASK]-Token ersetzt werden. Für diese Masken werden vom BERT Modell anschließend Wörter errechnet, die diese Lücken füllen sollen. Anhand des richtigen oder fehlerhaften Ersetzen der Masken, trainiert das Modell seine Präzision. Außerdem wird durch dieses 'masked lerarning model' eine bidirektionale Aufmerksamkeit erlernt. Durch das Maskieren eines Wortes in der Mitte des Textes, kann der Algorithmus auch nachfolgende Worte in seinen Entscheidungsprozess einfließen lassen und erlernt damit diese bidirektionale Aufmerksamkeit (Devlin u. a. (2018)).

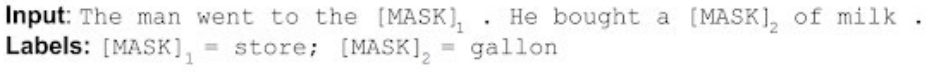

Abbildung 3.10: Beispiel: ermittelte Worte für [MASK]-Tokens (OpenBlog)

Ein nicht bidirektionales Modell kann den ersten Satz sehr gut ergänzen, wird jedoch Probleme im zweiten Satz haben, da für die Ermittlung der zweiten Maske der Kontext der nachfolgenden Worte wichtig ist. Ein nicht bidirektionales Modell könnte nach 'He bought a' beispielsweise 'house' oder andere Wörter einfügen. Erst mit der bidirektionalen Aufmerksamkeit kann das Modell treffende Aussagen über die Maske machen, da das Modell nun eine Aufmerksamkeit auf 'milk' erlernen kann (Abb. 3.10) (Devlin u. a. (2018)).

Next Sentence Prediction: Das BERT Modell verfügt weiterhin über erlernte Beziehungen zwischen Sätzen. So kann ein nachfolgender Satz als solcher erkannt und auch als solcher gebildet werden. Um diese Beziehung zu erlernen, wird das Modell mit vielen Satzpaaren trainiert. Dabei besteht die Hälfte aus tatsächlichen Satzpaaren und die andere Hälfte aus zufällig zusammengesetzten Satzpaaren. Das Training hat anschließend das Ziel, den zweiten

Sätzen der richtigen Satzpaare das Label 'IsNext' zu geben, und den zufälligen das Label 'NotNext' (Abb. 3.11). (Devlin u. a. (2018))

Abbildung 3.11: Beispiel: Next Sentence Prediction (OpenBlog)

Finetuning: Um das Modell anschließend auf bestimmte Daten anwendbar zu machen, ist es notwendig, die Berechnungen dieses Modells auf diese Daten zu optimieren. Dafür ist ein Trainings- und Validierungsdatensatz notwendig. Dort wird der Output mit den gelabelten Daten verglichen und die Parameter der Models angepasst.

4 Lösungskonzeption

Um sich dem Problem der Klassifizierung von Texten geringer Trainingsdatenmenge zu widmen, wurden zwei Lösungsansätze verfolgt. Im ersten Ansatz wurde ein starker Algorithmus um eine Kontextvariable erweitert (LCR) und im zweiten Ansatz ein komplett neuer Algorithmus entworfen (CC).

4.1 LRC

Der erste Ansatz zur Optimierung einer Klassifikation besteht darin, einen bestehenden Algorithmus, der keinen Kontext berücksichtigt, um eine Kontextvariable zu erweitern. Der Algorithmus, der für die Erweiterung ausgesucht wurde, ist die logistische Regression, da sie in den vorhergehenden Tests bessere Ergebnisse als die meisten anderen kontextfreien Klassifizierungsalgorithmen erreichen konnten.

4.1.1 LRC im Detail

Die logistische Regression ermittelt für den zu analysierenden Text eine Wahrscheinlichkeit für jede mögliche Klasse. Das Ziel der Erweiterung ist es, diesen errechneten Wert mit einer 'streak' zu multiplizieren. Diese 'streak' errechnet sich, indem der Text aus der Perspektive jeder einzelnen Klasse betrachtet wird. Im ersten Schritt wird der Text nach den Verfahren aus Kapitel 2 (tokenizierung, stopwords, stemming) aufgearbeitet. Als Nächstes wird für jedes Wort die Wortwahrscheinlichkeit und der Wortkontext errechnet. Im Prozess der Ermittlung des Wortkontextes wird überprüft, ob das Wort in der Klasse vorkommt, die aktuell betrachtet wird. Anschließend wird überprüft, ob das darauf folgende Wort als 'successor' des aktuellen Wortes abgespeichert wurde. Wenn dies der Fall ist, wird die temporäre 'streak' um eins erhöht. Sollte der Nachfolger des aktuellen Wortes nicht als 'successor' des aktuellen Wortes auftauchen, wird die temporäre 'streak' auf 1 gesetzt. Der Wert der temporären 'streak' wird in jeder Iteration mit dem aktuellen 'streak' verglichen und der größere der beiden bildet den neuen 'streak'-Wert. So wird die maximale Wortkette ermittelt, die das System mit der Klasse verbindet. Dieser Wert ist eine Ganzzahl größer gleich 1 und wird mit dem Ergebnis

der logistischen Regression der Klasse multipliziert. Anschließend wird die Klasse mit dem höchsten Wert als vorhergesagte Klasse zurückgegeben.

4.2 Context-Classifyer

Die ersten Ergebnisse der Erweiterung der logistischen Regression erschienen vielversprechend, da eine Verbesserung der accuracy festzustellen war. Daher war der nächste Schritt, einen Algorithmus zu entwerfen, der einen noch stärkeren Fokus auf die Wortkontextsuche legt und außerdem eine 'Wort für Wort Klassifikation' der Texte ermöglicht, um diese als Grundlage einer späteren Unterteilung der Urkunden in Urkundenbestandteile zu nutzen. Der neuartige Klassifizierungsalgorithmus baut auf einer Datenbank auf, die an ein 'Bag Of Words' Model erinnert.
Über solch einen 'Bag Of Words' Algorithmus wird eine Wahrscheinlichkeit errechnet, dass ein bestimmtes Wort in einer Klasse (Urkundenbestandteil) vorkommt. Die erweiterte Idee besteht darin, diese Wahrscheinlichkeit mit einem berechneten Kontextwert zu multiplizieren und somit einen konzeptionellen Kompromiss aus einem convolutional neural network (CNN) und einem 'Bag Of Words' Modell zu kreieren.

4.2.1 Context-Classifyer im Detail

Datenorganisation: Alle Wörter der Trainingsdaten werden in eine 'In-Memory-Datenbank', repräsentiert durch ein Python Dictionary, aufgenommen. Die Daten werden auf der ersten Ebene nach Wörtern organisiert. Über das Wort - im folgenden 'Index-Wort' - ist ein weiteres Dictonary adressierbar, in dem die Frequenz gespeichert ist, wie oft das Index-Wort in den Trainingsdaten auftaucht, sowie ein Dictonary, indem die Urkundenbestandteile, in dem das Index-Wort vorkommen kann, als weiteres Dictonary organisiert ist. In dem Dictonary steht die Frequenz, in der das Wort in dem Urkundenbestandteil vorgekommen ist und eine Liste an Wörtern, die vor ('pre') und nach ('succ') dem Index-Wort in dem Urkundenbestandteil vorkam. Sollte das Index-Wort am Ende eines Urkundenbestandteils stehen, steht als 'succ' Wert ein 'endOfFile' und sollte es am Anfang eines Urkundenbestandteils stehen, steht als 'pre' ein 'startOfFile'. Das Dictonary ist als JSON serialisiert folgendermaßen strukturiert 4.1:

```
{
    "<wort>": {
        "frequency": 2095,
        "list": {
            "<Urkundenbestandteil>": {
                "frequency": 2095,
                "pre": {
                    "<wort1>": 200,
                    "startOfFile": 1895
                },
                "succ": {
                    "<wort200>": 95,
                    "endOfFile": 2000
                }
            }
        }
    },
        ...
}
```

Abbildung 4.1: Beispielhafte Datenrepräsentation eines Wortes

Diese Organisation der Daten hat den Effekt, dass unter anderem Filgendes abgefragt werden kann:

- Wie häufig kommt in einem bestimmten Urkundenbestandteil das Indexwort vor?
- Welches sind die nachfolgenden und vorausgehenden Wörter dieses Indexwortes in diesem Urkundenbestandteil?
- Wie häufig kommt ein Wort in dem gesamten Trainingssatz vor?

Mathematischer Hintergrund: Um zu bestimmen, für welchen Urkundenbestandteil ein bestimmtes Index-Wort charakteristisch ist, werden zwei Faktoren berücksichtigt: Die **Wort-**

wahrscheinlichkeit und der **Wortkontext**, in dem das Index-Wort in dem jeweiligen Urkundenbestandteil auftaucht.

Wortwahrscheinlichkeit: Aus den Daten lässt sich nun eine Wahrscheinlichkeit errechnen, die besagt, wie häufig das Index-Wort in einem bestimmten Urkundenbestandteil vorkommt. So sind Wörter, die in jedem Urkundenbestandteil vorkommen (beispielsweise Füllwörter, etc.) ein schlechter Indikator für eine Kategorie. Die Wahrscheinlichkeit wird wie folgt berechnet (Form. 4.1):

$$WW(w,u) = h_u(w,u)/h(w) \tag{4.1}$$

w = Index-Wort
u = zu untersuchende Klasse (Urkundenbestandteil)
$h_u()$ = Häufigkeit des Index-Wortes in der Klasse (Urkundenbestandteil)
$h()$ = Häufigkeit des Index-Wortes

Für Wörter, die in vielen Urkundenbestandteilen gleichmäßig verteilt vorkommen, ergibt sich eine geringe Wortwahrscheinlichkeit. Für Index-Wörter, die besonders häufig in einem Urkundenbestandteil vorkommen, errechnet sich eine hohe Wortwahrscheinlichkeit. Diese Wortwahrscheinlichkeiten werden Wort für Wort zu einer akkumulierten Wortwahrscheinlichkeit aufsummiert, die wir im Folgenden Wortwarscheinlichkeit nennen werden (Form. 4.2).

$$Wortwahrscheinlichkeit(u) = \sum_{w=1}^{W} WW(w,u) \tag{4.2}$$

Wortkontext: Außerdem soll der Kontext des Index-Wortes berücksichtigt werden, also mit welchen Wörtern das Index-Wort in welchem Urkundenbestandteil häufig in Zusammenhang steht. Darin besteht die Innovation dieses Algorithmus. Dafür werden die Wörter in den Texten betrachtet, die nach dem Index-Wort in dem zu klassifizierenden Text stehen. Sollte das Wort im Datensatz als nachfolgendes Wort in dem zu untersuchenden Urkundenbestandteil vorkommen ('succ'), wird die temporäre 'streak' erhöht. Sollte es nicht vorkommen, wird die temporäre 'streak' auf den Wert 1 gesetzt. Zusätzlich zu dem temporären 'streak' wird eine weitere Variable 'streak' geführt. Die temporäre 'streak' wird in jedem Iterationsschritt mit der 'streak' verglichen. Sollte die temporäre 'streak' größer sein als die 'streak', wird der Wert der 'streak' auf den Wert der temporären 'streak' gesetzt. Nun wird das Index-Wort der Nachfolger

des Index-Wortes und der Prozess wird wiederholt. Dieser Prozess ermöglicht das Ermitteln der maximalen Wortkette aus dem Text, die der Algorithmus mit der Klasse in Verbindung bringen kann (Form. 4.3).

$$\text{streak(i, W, k, ts, s)} = \left\{ \begin{array}{ll} s & i >= len(W_k) \\ streak(i+1, W, k, 1, s) & W_{k,i+1} \notin W_{k,i}['succ'] \\ streak(i+1, W, k, ts+1, max(st, s)) & sonst \end{array} \right\} \quad (4.3)$$

i = aktueller Index
W = Wörter
k = zu untersuchender Urkundenbestandteil
ts = temporäre streak
s = streak
$max(a, b)$ = maximaler Wert von a und b
$len(x)$ = Länge vom Element x

Die Einstiegsfunktion lautet (Form. 4.4):

$$Wortkontext(woe, k) = streak(0, W, k, 0, 0) \quad (4.4)$$

Hierbei sind W die Wörter in einem Text und k eine zu untersuchende Klasse.
Auf diese Art wird in einem Text für jeden möglichen Urkundenbestandteil ein Kontext ermittelt.

Berechnung: Nachdem jede Wortwahrscheinlichkeit der Wörter im Text und der Wort-Kontext für jeden möglichen Urkundenbestandteil ermittelt worden sind, werden beide Werte miteinander multipliziert. Somit wird für jeden möglichen Urkundenbestandteil eine Wahrscheinlichkeit errechnet, dass der Text in diesem Urkundenbestandteil vorkommt. Diese Multiplikation stellt einen ersten Ansatz einer Berechnung dar und kann später optimiert werden, um eine Präzisionssteigerung zu erreichen.

Diese Berechnung der Wahrscheinlichkeit für jedes Urkundenbestandteils sieht wie folgt aus (Form. 4.5):

$$p(t,k) = \left(\sum_{i=0}^{l(t)} Wortwarscheinlichkeit(W(t)_i, k) \right) * Wortkontext(W(t), k) \tag{4.5}$$

t = zu untersuchender Text
k = zu untersuchender Urkundenbestandteil
$p(t,k)$ = Wahrscheinlichkeit für Text t in Urkundenbestandteil k
$l(t)$ = Länge von t
$W(t)$ = Wörter in t

5 Metriken zum Vergleich

Um Ergebnisse zu analysieren, müssen zuerst Metriken festgelegt werden, nach denen ein Algorithmus als 'gut' und als 'schlecht' eingeschätzt werden kann. Diese Metriken sind:

1. Korrektheit (accuracy, precision and recall)
2. Zeit

Die Korrektheit ist die weitaus wichtigere Metrik, da es sich in diesem Fall um eine Anwendung handelt, bei der nicht häufige Analysen gemacht werden müssen und diese auch nicht schnell sein müssen, wie beispielsweise bei einer Objekterkennung in Bildern für ein selbstfahrendes System. Die Analyse eines Corpus (hier eine Sammlung von Texten) kann hier Stunden, Tage oder Wochen dauern. Wichtiger ist die Präzision, die einen sehr hohen Wert erreichen soll, damit die fehlerhaften Klassifizierungen so gering wie möglich ausfallen. Ein Algorithmus, der beispielsweise 90% Genauigkeit hat, ist ein gutes Ergebnis, aber für diesen Anwendungsfall nicht genügend, da jeder zehnte Text falsch eingeschätzt würde und damit jeder Text noch einmal kontrolliert werden müsste.

5.1 Korrektheit

Die gängigsten Metriken für Korrektheit sind: $accuracy$ (Form: 5.1), $precission$ (Form: 5.2), $recall$ (Form: 5.5) und der $f1$ (Form: 5.6) Wert. Diesen Metriken liegen 'true positives' (TP = korrekt als der Klasse zugehörig klassifiziert), 'false positives' (FP = fälschlich als der Klasse zugehörig klassifiziert) und 'false negatives' (FN = fälschlich als der Klasse nicht zugehörig klassifiziert) zugrunde. Die Werte lassen sich durch eine 'Error-Matrix' (Abb. 5.1) visualisieren.

$$accuracy() = \frac{\sum TP}{\sum} \tag{5.1}$$

$$precission(class) = \frac{TP_{class}}{TP_{class} + FP_{class}} \tag{5.2}$$

$$recall(class) = \frac{TP_{class}}{TP_{class} + FN_{class}} \tag{5.3}$$

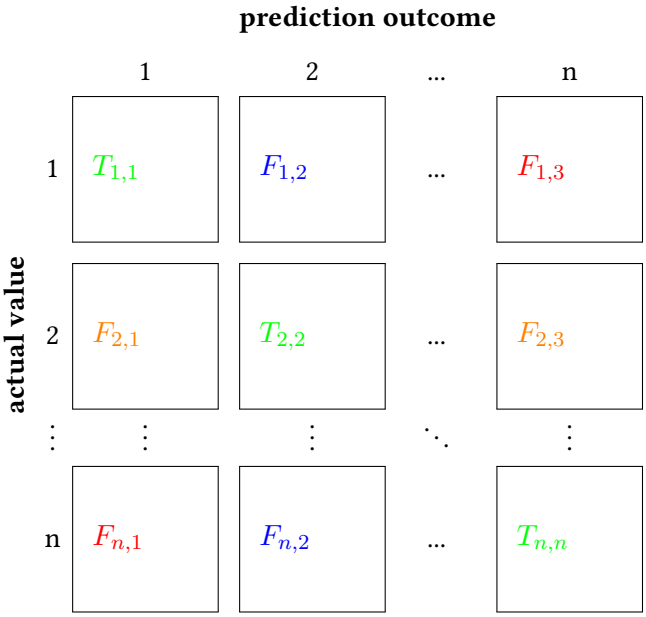

Abbildung 5.1: Error Matrix

Der $f1$-Wert (Form: 5.6) kombiniert sowohl recall und precission in einem Harmonischen Mittel miteinander. Ein Harmonisches Mittel zeichnet sich dadurch aus, dass abweichende recall- und precissions-Werte sich negativ auf den $f1$ Wert ausüben. Dadurch werden sowohl recall als auch precission gleich gewichtet in die Metrik einbezogen (Takahashi u. a. (2022)). Im 'Micro-Average' Verfahren, das in dieser Arbeit für die Ermittlung der Ergebnisse verwendet wird, werden die Ergebnisse der Klassen vor den Divisionen addiert. Dadurch kann eine mögliche Disbalance der Klassen ausgeglichen werden.

$$precission_{mic} = \frac{\sum TP}{\sum TP + \sum FP} \tag{5.4}$$

$$recall_{mic} = \frac{\sum TP}{\sum TP + \sum FN} \tag{5.5}$$

$$f_{1mic} = 2 * \frac{precission_{mic} * recall_{mic}}{precission_{mic} + recall_{mic}} \tag{5.6}$$

5.2 Zeit

Die Zeit ist wie oben beschrieben kein wichtiger Faktor in diesem Anwendungsfall, jedoch einer, der nicht zu vernachlässigen ist, wenn es um andere Anwendungsfälle geht. Sollte sich

der Algorithmus als präzise genug erweisen, ist es denkbar, diesen Algorithmus auf andere Probleme anzuwenden. Es wird sowohl die Zeit für das Trainieren als auch für die Vorhersage gemessen. Da ein Training nur einmalig gemacht werden muss. Eine Vorhersage muss immer wieder ausgeführt werden, sodass eine lange Trainingszeit verzeihbarer ist, als eine lange Dauer für eine Vorhersage.

6 Vergleich der Ergebnisse

6.1 Der Prozess des Testens

Um Ergebnisse zu ermitteln, welche aussagekräftig sind, wurden alle Algorithmen auf den gleichen Datensätzen trainiert und validiert. Hierfür wurden beide Datensätze nach Trainingsdaten (70%) und Validierungsdaten (30%) aufgeteilt. Diese Tests wurden alle auf einem System mit einem AMD Ryzen 9 5900X 12x 3.70GHz Prozessor, 32GB (2x 16GB) RAM und einer GeForce RTX 3060 12GB Grafikkarte ausgeführt. Der Vorgang wurde bis auf eine Ausnahme zehnmal durchgeführt, der Mittelwert der accuracy, precision, recall und $f1$ über die zehn Durchläufe ermittelt und die verstrichene Zeit aufsummiert. Das Modell, welches nicht zehnmal durchlaufen wurde, ist das Transformer Model, da ein einfacher Durchlauf dieses Algorithmus über 400 Stunden auf beiden Datensätzen dauert. Die Zeit des Transformers wurde, um eine Vergleichbarkeit der Laufzeiten zu gewährleisten, verzehnfacht. In jedem Durchlauf werden die Daten, die schon durch den Datenaufbereitungs-Schritt (Pre-Processing) bereinigt wurden, in ihre Wörter zerteilt. Daraufhin wurde anhand der Trainingsdaten das Modell trainiert und anschließend die Vorhersage der Validierungsdaten durchgeführt. Als letzter Schritt wurden die vorhergesagten Ergebnisse mit den tatsächlichen Ergebnissen verglichen. Sowohl für das Trainieren des Modells, als auch für das Errechnen der Vorhersage, wurde die Zeit gestoppt. In diese Zeit fließen weder das Aufbereiten der Daten (Pre-Processing) noch das Errechnen der Metriken ein. Bei Algorithmen mit neuronalen Netzen wurden über zehn Epochen trainiert. Auch hier ist das Transformer Modell eine Ausnahme mit nur drei Epochen, da ein Training über zehn Epochen hochgerechnet über 100 Stunden dauern würde.

6.2 Die Ergebnisse:

Um eine aussagekräftige Einschätzung der Güte der Algorithmen zu ermitteln, werden die errechneten Werte der Metriken (Zeit, accuracy, $f1$) sowohl des LRC als auch des CC mit herkömmlichen, State-Of-The-Art Algorithmen verglichen.

Algorithmus	acc	preci	recall	$f1_{micro}$	Zeit $\sum$	Zeit Training	Zeit Vorhersage
NN	0.81	0.80	0.80	0.80	49.8m	48.6m	1.1m
CNN	0.70	0.70	0.70	0.69	15.5h	15.3h	13.3m
DT	0.55	0.55	0.55	0.54	3.1m	2.9m	9s
RF	0.77	0.77	0.77	0.75	10.3m	10m	13.2s
LR	0.849	0.85	0.85	0.84	6m	5.8m	9.5s
LRC	0.853	0.85	0.85	0.85	12m	7.6m	4.4m
CC	0.72	0.72	0.72	0.73	8.2m	2.1m	6m
SVM	0.854	0.854	0.854	0.85	35s	26s	9.3s
Transformer	0.84	0.84	0.84	0.83	4.5t(10.9h)	101.1(10)h	7,5h(45m)

Tabelle 6.1: Ergebnisse nach zehn Durchgängen (Transformer nur 1 Durchlauf); Datensatz: 20 Newsgroups; rot: besonders lange Laufzeiten; fett gedruckt: neue Algorithmen; Transformer in Klammern Zeit für 1 Durchlauf - ohne Klammern Hochrechnung

Algorithmus	acc	preci	recall	$f1_{micro}$	Zeit $\sum$	Zeit Training	Zeit Vorhersage
NN	0.89	0.89	0.89	0.84	25.2m	24.8m	26.5s
CNN	0.87	0.87	0.87	0.80	12.2h	12.1h	7.2m
DT	0.83	0.83	0.83	0.78	23.5s	22.8s	0.7s
RF	0.88	0.88	0.88	0.83	9m	8.9m	3.3s
LR	0.891	0.891	0.891	0.84	2.8m	2.8m	0.7s
LRC	0.893	0.893	0.893	0.84	4.9m	4.7m	10.3s
CC	0.79	0.79	0.79	0.61	2.2m	1.9m	15s
SVM	0.897	0.897	0.897	0.85	5s	4.6s	0.68s
Transformer	0.90	0.90	0.90	0.87	13.4t(32h)	310(31)h	12(1.2)h

Tabelle 6.2: Ergebnisse nach 10 Durchgängen (Transformer nur 1 Durchlauf); Datensatz: mittelalterliche Urkunden; rot: besonders lange Laufzeiten; fett gedruckt: neue Algorithmen; Transformer in Klammern Zeit für 1 Durchlauf - ohne Klammern Hochrechnung

6.2.1 Zeit

Der Algorithmus mit der längsten Laufzeit ist das Transformer Modell, mit akkumuliert über 456 Stunden Laufzeit für hochgerechnet zehn Trainings- und Testdurchläufen auf beiden Datensätzen. Dieser Algorithmus ist der einzige, der getesteten Algorithmen, der nur über eine Iteration trainiert und anschließend hochgerechnet wurde, da sich dieser eine Durchlauf schon auf 45.6 Stunden belief. Dieser erhebliche Zeitaufwand ist primär dem Training geschuldet, welches 41 Stunden ausmachte. Die Testphase dauerte hingegen auch lange, aber mit addierten 19.9 Stunden ist diese mehr als doppelt so schnell wie die Trainingsphase. Die Laufzeit auf den Daten der 20 Newsgroups fiel dabei geringer aus als bei den lateinischen, mittelalterlichen Urkunden. Dies kann mit der unterschiedlichen Corpusgröße, als auch mit den unterschiedlichen Modellen, die gewählt wurden, in Zusammenhang stehen. Denn der Korpus der Newsgroups enthält 11314 Trainingsdaten, wobei der mittelalterliche Korpus 19278 Trainingsdatensätze besitzt.
Der Algorithmus mit der zweitlängsten Laufzeit ist das CNN, welches aufsummiert 24.4 Stunden für dieselbe Anzahl an Trainings und Testdaten in Anspruch genommen hatte. Beide Algorithmen trainieren über Epochen ihre Trainingsdaten. Der Transformer benötigt für die gesamte Berechnung von drei Epochen über 17-mal länger als der CNN für zehn Epochen benötigt. Beim CNN kann eine höhere Laufzeit bei den 20 Newsgroups beobachtet werden. Dies kann an der höheren Menge an Klassen dieses Datensatzes liegen. Das neuronale Netz benötigt für diese Aufgabe nur 1.3 Stunden. Er trainiert ebenfalls zehn Epochen, spart sich allerdings das Trainieren der 'Convolutional Layer', was zu erheblich geringerem Zeitaufwand führt. Auch hier werden nahezu die gesamten 1.2 Stunden für das Training verwendet. Nur ein Bruchteil der Zeit wird für das Testen benötigt. Die logistische Regression benötigt für alle zehn Iterationen nur 8.7 Minuten. Auch hier wird der größte Teil für das Trainieren benötigt (8.6 Minuten). Wird der Kontext zur Regression hinzugerechnet, benötigt der Algorithmus das Doppelte an Zeit und kommt auf 16.9 Minuten, wovon 12.3 Minuten das Trainieren benötigen und 4.6 Minuten das Testen benötigt. Der CC liegt mit 10.4 Minuten in einem ähnlichen Bereich, ist jedoch der einzige Algorithmus, der mehr Zeit in der Testphase als in der Trainingsphase benötigt. Dies ist erklärbar durch die simple Struktur der Daten, die der Algorithmus aufbaut und die aufwendige Suche, die der Algorithmus durchführen muss. Der schnellste Algorithmus ist die SVM mit 40 Sekunden für beide Datensätze (Tab. 6.1, 6.2).

6.2.2 Genauigkeit

Im Vergleich zu dem Zeitaufwand ist für den Anwender die Genauigkeit von erheblich größerer Bedeutung. Wir analysieren accuracy, precision, recall und $f1$. Im Folgenden werden vor allem die Metriken accuracy und $f1$ betrachtet, da die accuracy ein gutes Gesamtbild der Zuverlässigkeit des Algorithmus widerspiegelt und der $f1$-Score ein Harmonisches Mittel der precission und des recalls darstellt. Die Datensätze müssen unabhängig voneinander betrachtet werden, da sie verschiedene Ergebnisse ergaben.

20 Newsgroups: Der stärkste Algorithmus in Bezug auf die accuracy ist die SVM und die logistische Regression mit und ohne Kontexterweiterung, mit einer Genauigkeit von 85%. Die logistische Regression ist dabei von den ersten drei Algorithmen die schlechteste, mit einem Abstand von 0.4% zum zweitbesten Algorithmus - der logistischen Regression mit Kontexterweiterung. Diese erreicht einen exakten Wert von 85.3%. Nur die SVM konnte mit $85,4\%$ einen um $0,1\%$ besseren Wert erzielen. Das Transformer Modell erreicht mit 84% den viertbesten accuray-Wert. Dieses Ergebnis ist jedoch mit den anderen Ergebnissen schwer vergleichbar, da es sich um ein Training mit nur drei Epochen handelt und kein Mittelwert über zehn Iterationen gebildet wurde. Der CC konnte eine Genauigkeit von 72% erreichen und ist damit vor dem CNN (70%) und dem DT (55%) auf dem drittletzten Platz. Das NN konnte mit 81% einen besseren Wert erreichen als seine Kontext-erweiterte Version CNN mit 70% (Tab. 6.1).

Mittelalterliche Urkunden: In der Verarbeitung der Urkunden ist der Transformer der stärkste Algorithmus. Er erreicht eine accuracy von 90%. Die übrige Reihenfolge der besten Algorithmen verändern sich dabei jedoch kaum: den zweiten bis vierten Platz teilen sich SVM (89.7%), LRC (89.3%) und LR ($89,1\%$). Darauf folgt das neuronale Netz (89%), dicht gefolgt von dem RF Algorithmus (88%). Der CNN erreicht den drittletzten Platz mit 87%, gefolgt vom DT (83%). Den letzten Platz nimmt der CC ein mit einer Genauigkeit von 79%. Auffallend ist, dass der Abstand zwischen dem CNN und der NN deutlich geringer ausfällt (2%). Der Abstand betrug bei den Newsgroups 10% (Tab. 6.2).

Genauigkeit bei Klassen mit geringer Menge: Besonders anfällig sind bei Klassifizierungsalgorithmen Datensätzen mit einer geringen Menge, da diese oft nicht die nötige Menge an Trainingsdaten aufweisen können, um ausreichend diese Klasse zu erlernen. In dem Datensatz der Urkunden ist ein Beispiel für eine solche unterrepräsentierte Klasse die Klasse 'Narratives-Element'. Um eine Aussagekraft darüber zu erlangen, wie zuverlässig die Klassifi-

kation bei Daten geringer Menge ist, wird diese Klasse gesondert betrachtet. Sie besitzt 177 Datensätze, aus denen 124 Trainingsdatensätze und 53 Testdatensätze generiert wurden.

Algorithmus	$\mathbf{f1}_{micro}$	prec	recall
NN	0.335	0.38	0.34
CNN	0.095	0.31	0.06
DT	0.18	0.27	0.13
RF	0.2	0.78	0.11
LR	0.35	0.74	0.23
LRC	0.32	0.61	0.21
CC	0.08	0.185	0.05
SVM	0.36	0.68	0.25
Transformer	0.61	0.75	0.52

Tabelle 6.3: Mittelalterliche Urkunden - Kategorie 'Narratives-Element'

Für diesen Anwendungsfall eignet sich besonders gut die $f1$ Metrik, da sie am besten ausdrückt, wie akkurat Daten der Klasse zugeteilt werden und Bezieht außerdem ein, wie häufig ein Text dieser Klasse zugeordnet wurde, obwohl er dieser nicht angehört. Es wird also ein sehr detailliertes Bild über eine spezifische Klasse erzeugt. Auch für diese Ergebnisse wurden - ausgenommen des Transformer-Modells - zehn Iterationen ausgeführt und der Durchschnitt der Ergebnisse ermittelt. Es war festzustellen, dass einige Ergebnisse über die zehn Läufe sehr konstant waren (DT, RF, LR, LRC, CC, SVM) während andere stark fluktuierten (DNN, CNN). Die Klassifikation der SVM wird in der Literatur als gute Klassifikation angesehen, wenn es sich um Daten mit geringer Menge handelt (Sivakami (2018)). Die SVM erreicht auf den Daten des 'Narratives-Elements' einen $f1$-Wert von 0.36. Dieser Wert wird von dem Transformer Modell deutlich übertroffen (0.61). Dies kann an den Aufmerksamkeiten liegen, die das Modell erlernt. Ein starker Konkurrent der SVM ist die LR mit 0.35. Die Kontextvariable kann leider die Werte nicht verbessern, und hat sogar einen negativen Einfluss auf die $f1$ Metrik (0.32). Auch in diesem Fall setzt sich das Bild durch, dass das NN (0.335) nicht durch die Modifikation der CNN (0.095) um Conv. Layer verbessert werden konnte. Der CC schnitt mit durchschnittlichen 0.08 am schlechtesten ab (Tab. 6.3).

7 Algorithmus zum Zerteilen und Erkennen von Urkundenbestandteilen

Um Texte mit dem neuartigen Algorithmus unterteilen und klassifizieren zu können, sind folgende Schritte notwendig:

7.1 Windowing

Der zu zerteilende und zu klassifizierende Text wird in kleinere Teiltexte der Länge 'windowsize' unterteilt. Die 'windowsize' ist ein Parameter, der einen geraden Wert haben muss, der vom Nutzer bestimmt werden kann. Er hat Einfluss auf die Laufzeit, aber auch auf die Präzision des Algorithmus. Diese Unterteilung erfolgt, indem über den zu analysierenden Text iteriert wird und '$windowsize/2$ Wörter vor' und '$windowsize/2$ Wörter nach' dem Index-Wort zu einem Teiltext zusammengebunden werden. So werden bei einem zu analysierenden Text der Wörterlänge N, N viele Teilsätze der maximalen Wortlänge 'windowsize' und der minimalen Wortlänge $(windowsize/2) + 1$ generiert.

Beispiel: Die fett gedruckten Wörter in den Teiltexten sind die Index-Wörter und es wird eine 'windowsize' von vier gewählt.

Text: 'Hallo ich schreibe meine Bachelorarbeit über Künstliche Intelligenz'
Teiltexte: ['**Hallo** ich schreibe', 'Hallo **ich** schreibe meine', 'Hallo ich **schreibe** meine Bachelorarbeit', 'ich schreibe **meine** Bachelorarbeit über', …, 'über Künstliche **Intelligenz**']

Abbildung 7.1: Windowing

7.2 Klassifikation

Im Schritt der Klassifikation werden auf jeden der im Schritt 'Windowing' generierten Teiltexte die in den vorherigen Kapiteln beschriebene Klassifikation mit dem CC durchgeführt. Die Klassen in den eckigen Klammern sind fiktiv.

Beispiel:

Teiltexte: ['**Hallo** ich schreibe', 'Hallo **ich** schreibe meine', 'Hallo ich **schreibe** meine Bachelorarbeit', 'ich schreibe **meine** Bachelorarbeit über', ..., 'über Künstliche **Intelligenz**']
Ergebnis: [1, 1, 4, 1, 1, 2, 2, 2]

Abbildung 7.2: Klassifikation

7.3 Collapsing

Beim Collapsing werden alle aufeinander folgenden Urkundenbestandteil der selben Klasse zusammengefasst, sodass ein Tupel generiert wird, das an erster Stelle den Urkundenbestandteil enthält und an zweiter Stelle die Anzahl, wie häufig dieser Urkundenbestandteil hintereinander in dem Dokument an dieser Stelle vorkommt.

Beispiel:

Klassifikation: [1, 1, 4, 1, 1, 2, 2, 2]
Ergebnis: [(1, 2), (4, 1), (1, 2), (2, 3)]

Abbildung 7.3: Collapsing

7.4 Tilt Remove

Im 'tilt remove' Schritt werden alle Elemente aus der im vorherigen Schritt generierten Liste entfernt, deren Anzahl geringer als der Parameter 'tilt' ist. Das Wort, das keinem Urkundenbestandteil mehr zugeordnet ist und daher als Waise bezeichnet wird, bekommt eine neue Zuordnung in den darauf folgenden Urkundenbestandteil. Dieser Schritt bewirkt, dass kleine Fehleinschätzungen kompensiert werden, da es sehr wenig Klassen gibt, die ausschließlich aus einem Wort bestehen, und diese kleinen vermuteten Fehleinschätzungen durch den 'tilt' eliminiert werden. Der 'tilt' wurde für dieses Beispiel auf den Wert eins gesetzt.

Beispiel:

Collapsing: [(1, 2), (4, 1), (1, 2), (2, 3)]
Ergebnis: [(1, 2), (1, 3), (2, 3)]

Abbildung 7.4: Tilt Remove

7.5 Collapsing:

Nun erfolgt ein weiterer 'Collapsing' Schritt, der dasselbe bewirkt wie der erste 'Collapsing' Schritt.

Beispiel:

Trigger Remove: [(1, 2), (1, 3), (2, 3)]
Ergebnis: [(1, 5), (2, 3)]

Abbildung 7.5: Collapsing

7.6 Ergebnis

Somit haben wir unseren Text nun in zwei Teile unterteilt und den Teiltexten zusätzlich eine Klasse zugewiesen.

Beispiel:

Text: 'Hallo ich schreibe meine Bachelorarbeit' & 'über Künstliche Intelligenz'
Kategorien: 1, 2

Abbildung 7.6: Ergebnis

8 Ergebnisse des Algorithmus zum Zerteilen und Erkennen von Urkundenbestandteilen

Im Gegensatz zu den Klassifizierungsalgorithmen ist die Ermittlung der richtigen Teilung der Urkunden in Urkundenbestandteile eine kompliziertere Aufgabe. Hierzu wurden nach der Datenaufbereitung (pre-processing) die Testdaten der Urkundenbestandteile gleichmäßig in zufällig kombinierten Texten zusammengefasst. Diese willkürlich kombinierten Urkunden werden anschließend mit dem Algorithmus zerteilt und der Indexwert der Mitte jedes ermittelten Urkundenbestandteils wird mit dem Indexwert der Mitte des ursprünglichen Urkundenbestandteils im Text abgeglichen. Sollten diese Werte weniger als X Stellen voneinander abweichen, wird der Urkundenbestandteil als gefunden angesehen. In den folgenden Tests wurde für X der Wert elf festgelegt.

Dieses Verfahren wirft Probleme auf, wie zum Beispiel, dass vorherige falsche Einschätzungen des Algorithmus die späteren korrekten Zuweisungen beeinträchtigen, da sich Urkundenbestandteile hintereinander in Texten befinden.

Run	Accuracy
1	6.88
2	6.65
3	6.97
4	7.08
5	6.77
6	7.18
7	7.58
8	7.31
9	7.46
10	7.55
Gesammt	**7.14**

Tabelle 8.1: Korrektheit über zehn Durchläufen

Dieses Testverfahren erreicht über zehn Durchläufen eine durchschnittliche Wahrscheinlichkeit der korrekten Ermittlung der Mitten der Urkundenbestandteile in zufällig kombinierten Urkunden von 7.1%.

9 Fazit

9.1 Zusammenfassung

Im Zuge meiner Anstellung als Werkstudent an der Universität Hamburg in der geschichtlichen Fakultät bestand meine Aufgabe darin, mittelalterliche Urkunden in lateinischer Sprache zu unterteilen und zu klassifizieren. Eine besondere Herausforderung liegt darin, auch bei Klassen mit geringer Datenmenge zuverlässige Ergebnisse zu erreichen. Für eine möglichst effektive Klassifizierung wurde in einem ersten Schritt mehrere bereits vorhandene Klassifizierungsalgorithmen getestet. Die Ergebnisse waren allerdings nicht zufriedenstellend. Daraufhin wurden zwei Konzepte für eine Verbesserung der Ergebnisse verfolgt:

Das erste Konzept (LRC): Bestand darin, die logistische Regression um eine Kontextvariable zu erweitern. Ausschlaggebend für die Auswahl der logistischen Regression waren gute Ergebnisse von Tests, die im Vorfeld angefertigt wurden.

Das zweite Konzept (CC): Orientiert sich nicht an einem bereits existierenden Algorithmus, sondern verfolgt die Idee, mittels Multiplikation der Wortwahrscheinlichkeit mit dem Wortkontext eine Klassifizierung erreichen zu können.

Als Metriken wurden für die Algorithmen die Zeit, der $f1$-score und die accuracy gewählt. Um die Klassifikation eines Algorithmus zu beurteilen, wurde die accuracy verwendet bei der Betrachtung der Klassen mit geringer Datenmenge wurde der $f1$-Score genutzt. Außerdem wurde die benötigte Zeit, die ein Algorithmus für eine Klassifikation benötigt, analysiert, da bei den unterschiedlichen Algorithmen hierbei erhebliche Unterschiede zu verzeichnen waren. Diese Algorithmen wurden auf zwei verschiedenen Datensätze angewandt. Der erste Datensatz besteht aus mittelalterliche Urkunden auf lateinischer Sprache, die eine stark heterogene Aufteilung der Datensätze über die Klassen aufweist. Der zweite Datensatz ist der 20 Newsgroups Datensatz. Dieser Datensatz hat eine sehr homogene Aufteilung der Datensätze über die Klassen und ist ein stark verbreiteter Datensatz.
Die Ergebnisse aller getesteten Algorithmen erreichten auf dem Datensatz der mittelalterlichen

Urkunden ein deutlich höheres Niveau als auf den Datensätzen der 20 Newsgroups. Dies lässt nicht ausschließlich auf ein Merkmalsunterscheid der Daten rückzuschließen, sondern wird eine Wechselwirkung verschiedener Faktoren sein. Beispiele für einen solchen Faktor könnte der Unterschied der Anzahl der Worte der englischen Sprache im Vergleich zu derer der klassischen lateinischen Sprache sein oder der Unterschied der Anzahl der Klassen der einzelnen Datensätze (20 NG: 20, Urkunden 14). Auch eine bessere Unterscheidung der Klassen und ihrer Merkmale kann ein großer Faktor sein. Des Weiteren ist zu erwähnen, dass die ermittelten Ergebnisse der Tests auf den Datensatz der 20 Newsgroups mit denen der in Kapitel 3.1 betrachteten wissenschaftlichen Arbeit übereinstimmen und daher eine korrekte Anwendung der Algorithmen sichergestellt ist. Ein Grund dafür ist, dass viele der in der wissenschaftlichen Arbeit verwendeten Algorithmen in ihrer Implementierung übernommen und, wenn nötig, angepasst wurden. Die LRC errechnete auf den Daten der mittelalterlichen Urkunden eine vielversprechende accuracy. Er erreichte, im Gegensatz zu dem zugrundeliegenden Algorithmus (LR) über zehn Iterationen leicht bessere Ergebnisse, sodass ein Trend zu erkennen ist. Die accuracy des CC liegt um zehn Prozent unter denen der bereits etablierten Algorithmen. Bedenkt man jedoch, dass es sich bei dem CC um einen sehr simplen und bisher nicht optimierten Algorithmus handelt, ist dessen Abschneiden von besonderem Interesse. Es ist möglich, dass eine Verbesserung des CC (zum Beispiel durch eine Modifikation des mathematischen Modells) erreicht werden könnte. Die besten Ergebnisse erreichte das Transformer Model auf dem Datensatz der mittelalterlichen Urkunden mit 90% aber auch NN, SVM, die LR und die LRC erreichen knapp diese accuracy-Werte. Überraschend ist, dass der um eine Kontexterweiterung modifizierte CNN im Vergleich zu dem Algorithmus ohne Kontexterweiterung (NN) eine geringe Verschlechterung der accuracy aufweist. Allerdings macht dies lediglich zwei Prozent aus.

Außerdem wurde der benötigte Zeitaufwand analysiert, den die unterschiedlichen Algorithmen benötigen. Hierbei muss die Zeit der Trainingsphase unterschieden werden von der Zeit, die ein bereits trainierter Algorithmus für eine Klassifizierung eines Textes benötigt. Die Trainingszeit fällt dabei immer deutlich höher aus, ist allerdings auch nur einmalig notwendig. Daher kann hier auch ein größerer Zeitaufwand akzeptiert werden. Die meisten Algorithmen benötigen nur sehr wenig Zeit, allerdings erscheint die benötigte Zeit des CNN grenzwertig. Die des Transformers ist allerdings für den meisten Anwendungen inakzeptabel hoch. Da die Klassifikation der Urkunden von vielen Algorithmen mit einer ähnlich guten accuracy durchgeführt werden kann, ist nur in Ausnahmefällen der erheblich höhere Zeitaufwand, der bei dem Einsatz des Transformers entsteht, gerechtfertigt. Eine solche Ausnahme besteht immer dann, wenn ein besonderer Wert auf die korrekte Zuordnung von Klassen mit geringer

Datenmenge gelegt wird. Hier erreicht der Transformer nahezu doppelt so gute $f1$ Werte wie alle übrigen Algorithmen mit einem f1-Score von 0.61. Die SVM erreichte den zweithöchsten Wert von 0.36. Eine erhoffte Verbesserung der Ergebnisse der LR(0.35) um eine Kontextvariable in LCR(0.32) konnte leider nicht beobachtet werden. Bei einer Klassifikation von Texten, in denen ein großer Wert auf die korrekte Zuweisung von Klassen mit geringer Datenmenge gelegt wird, wird eine starke Empfehlung für das Transformer Modell ausgesprochen. Bei der Auswahl dieses Modells besteht jedoch der Kompromiss darin, dass die Laufzeit enorm hoch ist.
Eine zuverlässige Unterteilung von Urkunden in ihre Urkundenbestandteile konnte nicht erreicht werden. Jedoch besteht bei dieser Aufgabe die Schwierigkeit darin, eine gute Metrik zu finden, um Güte zu messen. Mit der Methodik, mit der die Ergebnisse gemessen wurden, wurden mit 7% keine zufriedenstellenden Ergebnisse ermittelt.

Limitation der Arbeit: Die Unterteilung der Urkunden in Urkundenbestandteile wurde ausschließlich auf Grundlage des CC durchgeführt. Da dieser sich später als der Algorithmus mit den schlechtesten Ergebnissen auf den mittelalterlichen Urkunden herausstellte, ist dies eine erhebliche Limitation. Außerdem wurden nicht alle Textklassifizierungsalgorithmen berücksichtigt, da dies den Rahmen dieser Arbeit gesprengt hätte. Ein weiterer Nachteil ist der, dass die Vergleichbarkeit des Transformer Modells mit allen anderen Modellen mit Vorsicht zu betrachten ist, da dieser im Gegensatz zu allen anderen Algorithmen ausschließlich über eine Iteration getestet und nur über drei Epochen trainiert wurde. Alle anderen Algorithmen, in denen neuronale Netze verwendet werden (CNN, NN), wurden über zehn Epochen trainiert. Diese Verringerung in den Iterationen und den Epochen sind der langen Laufzeit des Algorithmus geschuldet.

Vorteile der Arbeit: Durch diese Arbeit wurde das Verständnis der erwähnten, etablierten Klassifizierungsalgorithmen vertieft und anschließend ausführliche Tests mit diesen Algorithmen durchsgeführt. Es wurden zwei neue Konzepte von Algorithmen zur Klassifikation von sequenziellen Daten vorgestellt, getestet und anschließend mit den Ergebnissen der etablierten Klassifizierungsalgorithmen verglichen. Diese Tests wurden auf mehreren Datensätzen auf zwei unterschiedlichen Sprachen und mit sowohl stark homogener, als auch heterogener Aufteilung der Datensätze über die Klassen, durchgeführt. Anschließend wurden die Ergebnisse der Klassen mit geringer Menge aus dem Datensatz der heterogenen Aufteilung genauer betrachtet und somit eine zuverlässige Klassifizierung dieser Klassen diskutiert.

9.2 Ausblick

Diese Arbeit kann als Grundlage weiterer Forschungen auf diesem Gebiet gesehen werden, jedoch sind weitere Forschungen und Optimierungen der Algorithmen notwendig. So ist es in der Kontextsuche denkbar, nicht den besten streak zu errechnen, sondern einen akkumulierten streak. Diese könnte für den gesamten Text alle streaks erreichen und anschließend akkumuliert werden. Dies könnte einen höherer streak-Wert, und damit ein stärkerer Fokus auf den Kontext, bewirken. Gegenstand weiterer Forschungen ist eine Analyse, ob durch eine Anpassung der Parametrisierung von Variablen des CC eine Verbesserung erreicht werden kann. Auch das zugrundeliegende mathematische Verfahren ist mit einer Multiplikation der Wortwahrscheinlichkeit mit dem Wortkontext sehr simples Verfahren. Hier können Tests durchgeführt werden, welche beispielsweise das Exponieren der Wortwahrscheinlichkeit über den Wortkontext untersuchen. Bei der Unterteilung der Urkunden in Urkundenbestandteile kann an einigen Parametern experimentiert werden. So kann beispielsweise die 'windowsize', oder der 'tilt' verändert werden. Auch ist bei dem Algorithmus die Ermittlung der Güte modifizierbar. Eine Änderung des Wertes der Toleranz, welche beschriebt, wie weit die Mitte eines tatsächlichen Urkundenbestandteils von seiner vorhergesagten Mitte entfernt sein darf, kann angepasst werden und zu Verbesserungen der Analyse führen. Die vielversprechendste Veränderung des Algorithmus wäre allerdings das Ersetzen des CC durch einen besseren Algorithmus wie beispielsweise die SVM, LR, LRC oder bestenfalls ein Transformer Model.

9.3 Fazit

Eine Verbesserung der Textklassifizierung von Daten mit geringer Menge durch das Hinzufügen des Kontexts konnte in dieser Arbeit nicht erreicht werden. Jedoch konnte eine leichte Verbesserung der allgemeinen accuracy der logistischen Regression durch das Multiplizieren mit einer Kontextvariable beobachtet werden. Diese Verbesserung ist marginal, jedoch als Trend zu bewerten. Der CC erreicht mit seinem simplen und nicht optimierten Verfahren nicht das Niveau von Modellen wie dem Transformer, jedoch trotz seiner simplen Natur Ergebnisse, die sich erstaunlich nahe an denen der etablierten Algorithmen befinden. Diese Ergebnisse machen Hoffnung auf weitere Verbesserungen, es sind jedoch weitere Forschungen notwendig.

10 Glossar

Begriff	Erklärung
Window-Size	Parameter, der die Größe der Teiltexte bestimmt, die zum Unterteilen und Klassifizieren von Texten benötigt wird
Tilt	Parameter, der festlegt ab welcher 'Anzahl' im 'Trigger Remove'-Schritt ein Urkundenbestandteil als Waisen behandelt werden soll
mittelalterliche Urkunde	lateinischen Urkunden oft Schenkungsurkunden, welche in Urkundenbestandteile unterteilt werden soll
Urkundenbestandteile	Klassen in einer Urkunde, darunter Zählen z.B. 'Intitulatio' oder 'Narratives Element'
Support Vector Mashine (SVM)	intelligenter Algorithmus mit dem Ziel, das Klassifikationsproblem mit Hyperplanes in n-dimensionalen Räumen zu lösen
Random Forest (RF)	intelligenter Algorithmus mit dem Ziel, das Klassifikationsproblem mit Suchbäumen zu lösen
Convolutional Neural Network (CNN)	Ein intelligenter Algorithmus, der auf künstlichen neuronalen Netzen aufbaut, jedoch weitere zusätzliche Layers besitzt, die einen Fokus auf Kontext bzw. Nachbarn der Inputparametern legt
Regression	Verfahren in dem Regressionskoeffizienten (Gewicht von Features) mit Feature-Werten verrechnet werden, um Abhängigkeiten zwischen Feature und Klasse herstellen zu können
Logistsiche Regression (LR)	Klassifizierungsalgorithmus, der mittels Regression und einer logistischen Funktion Ergebnisse vorhersagt

Begriff	Erklärung
Treansformer Model	intelligenter Algorithmus, der besonders gut für sequenzielle Daten als Input geeignet ist
Intitulatio	Anrede in mittelalterlichen Urkunden
Narratives-Element	Erzählung des Tatbestandes der Rechtsgrundlage für die beurkundeten Vorgänge

Literaturverzeichnis

[Distilbert-base-uncased] : *distilbert-base-uncased*. – URL https://huggingface.co/distilbert-base-uncased. – Zugriffsdatum: 12.7.2023

[HomeSet] : *Home Page for 20 Newsgroups Data Set*. – URL http://qwone.com/~jason/20Newsgroups/. – Zugriffsdatum: 8.8.2023

[NLTKPackage] : *NLTK :: nltk.tokenize package*. – URL https://www.nltk.org/api/nltk.tokenize.html. – Zugriffsdatum: 15.8.2023

[OpenBlog] : *Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing – Google Research Blog*. – URL https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html. – Zugriffsdatum: 15.8.2023

[Simple-latin-bert-uncased] : *simple-latin-bert-uncased*. – URL https://huggingface.co/LuisAVasquez/simple-latin-bert-uncased. – Zugriffsdatum: 12.8.2023

[simplemma] : *simplemma*. – URL https://pypi.org/project/simplemma/. – Zugriffsdatum: 16.8.2023

[SklearnCountVectorizer] : *sklearn CountVectorizer*. – URL https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html. – Zugriffsdatum: 16.8.2023

[stopwords] : *stopwords*. – URL https://stopwords.readthedocs.io/en/latest/usage.html. – Zugriffsdatum: 16.8.2023

[StopwordsEnglish] : *stopwords english*. – URL https://github.com/astuanax/stopwords/blob/master/build/lib/stopwords/languages/english/default.txt. – Zugriffsdatum: 26.8.2023

[StopwordsLatin] : *stopwords latin*. – URL https://github.com/astuanax/stopwords/blob/master/build/lib/stopwords/languages/latin/default.txt. – Zugriffsdatum: 12.8.2023

[Tensor2TensorColaboratory] : *Tensor2Tensor Intro - Colaboratory*. – URL https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb

[Tf.keras.preprocessing.text.Tokenizer] : *tf.keras.preprocessing.text.Tokenizer*. – URL https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer. – Zugriffsdatum: 20.8.2023

[WieHamburg] : *Wie ensteht eine Edition? : Formulae - Litterae - Chartae : Universität Hamburg*. – URL https://www.formulae.uni-hamburg.de/edition-und-datenbank/wie-entsteht-eine-edition.html. – Zugriffsdatum: 9.8.2023

[WillkommenWerkstatt] : *Willkommen in der Formulae - Litterae - Chartae Werkstatt!*. – URL https://werkstatt.formulae.uni-hamburg.de/. – Zugriffsdatum: 9.8.2023

[UrkundePraefatio 2023] : *Urkunde Praefatio*. 8 2023. – URL https://werkstatt.formulae.uni-hamburg.de/texts/manifest:urn:cts:formulae:p3.105va106rb.lat001/passage/1. – Zugriffsdatum: 26.8.2023

[Cutler u. a. 2012] Cutler, Adele ; Cutler, D. R. ; Stevens, John R.: Random Forests. In: *Ensemble Machine Learning* (2012), S. 157–175. – URL https://link.springer.com/chapter/10.1007/978-1-4419-9326-7_5

[Devlin u. a. 2018] Devlin, Jacob ; Chang, Ming W. ; Lee, Kenton ; Toutanova, Kristina: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference* 1 (2018), 10, S. 4171–4186. – URL https://arxiv.org/abs/1810.04805v2. ISBN 9781950737130

[El-Habil 2012] El-Habil, Abdalla M.: An Application on Multinomial Logistic Regression Model. In: *Pakistan Journal of Statistics and Operation Research* 8 (2012), 3, Nr. 2, S. 271–291. – URL https://pjsor.com/pjsor/article/view/234. – ISSN 1816-2711

[Galar u. a. 2011] Galar, Mikel ; Fernández, Alberto ; Barrenechea, Edurne ; Bustince, Humberto ; Herrera, Francisco: An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes. In: *Pattern Recognition* 44 (2011), 8, Nr. 8, S. 1761–1776. – URL https://sci2s.ugr.es/sites/default/files/ficherosPublicaciones/1371_2011-pr-galar-ovo-ova.pdf. – ISSN 0031-3203

[Gasparetto u. a. 2022] Gasparetto, Andrea ; Marcuzzo, Matteo ; Zangari, Alessandro ; Albarelli, Andrea: A Survey on Text Classification Algorithms: From Text to Predictions. In: *Information 2022, Vol. 13, Page 83* 13 (2022), 2, Nr. 2, S. 83. – URL https://www.mdpi.com/2078-2489/13/2/83/htm. – ISSN 2078-2489

[George und Joseph 2014] George, Soumya ; Joseph, Shibily: Text Classification by Augmenting Bag of Words (BOW) Representation with Co-occurrence Feature. 1 (2014), S. 34–38. – URL https://d1wqtxts1xzle7.cloudfront.net/71843299/cbc0e35e6560fc657ad6b490aa07ad901575-libre.pdf?1633678296=&response-content-disposition=inline%3B+filename%3DText_classification_by_augmenting_the_ba.pdf&Expires=1703615145&Signature=DxagtnEWunRB~EkY-vfvPQgwDClxGaOopC5qGGQH-UNNATI4FZ-Wu57HVBLvP1bVSStFI_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

[Kingsford und Salzberg 2008] Kingsford, Carl ; Salzberg, Steven L.: What are decision trees? In: *Nature biotechnology* 26 (2008), 9, Nr. 9, S. 1011. – URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2701298/. – ISSN 10870156

[Kowsari u. a. 2019] Kowsari, Kamran ; Meimandi, Kiana J. ; Heidarysafa, Mojtaba ; Mendu, Sanjana ; Barnes, Laura ; Brown, Donald: Text Classification Algorithms: A Survey. In: *Information 2019, Vol. 10, Page 150* 10 (2019), 4, Nr. 4, S. 150. – URL https://www.mdpi.com/2078-2489/10/4/150/htm. – ISSN 2078-2489

[Kowsari K] Kowsari K: *GitHub - Text_Classification: Text Classification Algorithms: A Survey*. – URL https://github.com/kk7nc/Text_Classification. – Zugriffsdatum: 16.7.2023

[Kwak und Clayton-Matthews 2002] Kwak, Chanyeong ; Clayton-Matthews, Alan: Multinomial logistic regression. In: *Nursing Research* 51 (2002), 11, Nr. 6, S. 404–410. – URL https://europepmc.org/article/MED/12464761. – ISSN 0029-6562

[Mammone u. a. 2009] Mammone, Alessia ; Turchi, Marco ; Cristianini, Nello: Advanced Review Support vector machines. (2009). – URL https://www.researchgate.net/publication/252094616_Support_Vector_Machines_Review_and_Applications_in_Civil

[Meyer 2012] Meyer, David: Support Vector Machines * The Interface to libsvm in package e1071. (2012). – URL https://www.researchgate.net/publication/242323440_Support_Vector_Machines_The_Interface_to_libsvm_in_package_e1071

[Nitsche und Tropmann-Frick 2020] Nitsche, Matthias ; Tropmann-Frick, Marina: Context and Embeddings in Language Modelling-an Exploration. (2020). – URL https://ceur-ws.org/Vol-2277/paper24.pdf

[Ramos 2003] Ramos, Juan: Using TF-IDF to Determine Word Relevance in Document Queries. (2003). – URL https://www.researchgate.net/publication/228818851_Using_TF-IDF_to_determine_word_relevance_in_document_queries

[Rawlings u. a. 1998] Rawlings, John O. ; Pantula, Sastry G. ; Springer, David A D.: Applied Regression Analysis: A Research Tool, Second Edition. (1998). – URL https://link.springer.com/book/10.1007/b98890

[Singh und Patra 2013] Singh, Divakar ; Patra, Anuradha: A Survey Report on Text Classification with Different Term Weighing Methods and Comparison between Classification Algorithms A Survey Report on Text Classification with Different Term Weighing Methods and Comparison between Classification Algorithms General Terms Data mining, text mining. In: *Article in International Journal of Computer Applications* 75 (2013), Nr. 7, S. 975–8887. – URL https://www.researchgate.net/publication/260632345

[Sivakami 2018] Sivakami, M: Text classification techniques: A literature review. In: *Interdisciplinary Journal of Information, Knowledge, and Management* 13 (2018), S. 117–135. – URL https://www.ijikm.org/Volume13/IJIKMv13p117-135Thangaraj3803.pdf

[St u. a. 2016] St, Indra ; Wikarsa, Liza ; Indra, S T. ; Turang, Rinaldo: Using logistic regression method to classify tweets into the selected topics. (2016). – URL https://www.researchgate.net/publication/314667612

[Takahashi u. a. 2022] Takahashi, Kanae ; Yamamoto, Kouji ; Kuchiba, Aya ; Koyama, Tatsuki: Confidence interval for micro-averaged F 1 and macro-averaged F 1 scores. In: *Applied Intelligence* 52 (2022), 3, Nr. 5, S. 4961–4972. – URL https://link.springer.com/article/10.1007/s10489-021-02635-5. – ISSN 15737497

[Tong und Koller 2001] Tong, Simon ; Koller, Daphne: Support Vector Machine Active Learning with Applications to Text Classification. In: *Journal of Machine Learning Research* (2001), S. 45–66. – URL https://www.jmlr.org/papers/volume2/tong01a/tong01a.pdf

[Uchenna Oghenekaro und Benson 2022] Uchenna Oghenekaro, Linda ; Benson, Tammy: Text Categorization Model Based on Linear Support Vector Machine. In: *American Academic Scientific Research Journal for Engineering* (2022), S. 2313–4402. – URL https://asrjetsjournal.org/index.php/American_Scientific_Journal/article/view/7218

[Vaswani u. a. 2017] Vaswani, Ashish ; Shazeer, Noam ; Parmar, Niki ; Uszkoreit, Jakob ; Jones, Llion ; Gomez, Aidan N. ; Kaiser, Łukasz ; Polosukhin, Illia: Attention Is All You Need. In: *Advances in Neural Information Processing Systems* 2017-December (2017), 6, S. 5999–6009. – URL https://arxiv.org/abs/1706.03762v7. – ISBN 1706.03762v7

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 5. Januar 2024 Ferdinand Emanuel Trendelenburg