

Hochschule Darmstadt

Fachbereiche Mathematik und Naturwissenschaften & Informatik

Stimmungsquantifizierung für den Preis von Bitcoin
mit Deep Learning

Abschlussarbeit zur Erlangung des akademischen Grades
Master of Science (M. Sc.)
im Studiengang Data Science

vorgelegt von
Herr Bernhard Preisler

Referent: Herr Prof. Dr. Christoph Becker
Korreferentin: Frau Prof. Dr. Margot Mieskes

Ausgabedatum: 16.05.2018
Abgabedatum: 16.11.2018

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht. Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellenachweis versehen. Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, den 16.11.2018

Bernhard Preisler

Kurzfassung

Stimmungsquantifizierung für den Preis von Bitcoin mit Deep Learning

Die Vorhersage der Aktienkurse spielt in der Geschichte eine wichtige Rolle. Wer den Kurs am genauesten vorhersagen kann, gewinnt. In der jüngeren Vergangenheit erfreuen sich externe Quellen, unter anderem Stimmungen in Zeitungen oder Blogs, großer Beliebtheit. Weiterhin schwächen kürzliche Veröffentlichungen die vollständige ökonomische Rationalität und weisen signifikant in die Richtung des Herdentriebes, engl. Animal Spirits. Inspiriert von dieser Kombination übertragen wir das Prinzip auf die neuartigen Systeme Bitcoin und den Microbloggingdienst Twitter.

Um dies zu bewerkstelligen laden wir 10,31 Millionen Tweets vom 10.01. bis 01.08.2018 herunter und speichern sie. Danach erstellen wir mittels Amazon Mechanical Turk unseren Trainingsdatensatz, den wir für das Training der Modelle verwenden. Im nächsten Schritt trainieren wir etablierte maschinelle Lernverfahren, wie Random Forest und SVM, und ziehen lexikalische Verfahren wie vaderSentiment und sentimentr heran. Der Hauptteil unserer Arbeit konzentriert sich auf die Erstellung und Verbesserung von bestehenden Deep Learning-Modellen. Das erste Modell HDLTex ist inspiriert von K. Kowsari und besteht ausschließlich aus KNN-Layern. Das zweite Modell stammt von Y. Kim und besteht aus einer Mischung aus KNN- und CNN-Layern. Beide Modelle dienen als Ausgangslage, die wir im Laufe der Thesis signifikant auf unser Szenario weiter anpassen.

Neben einer enormen Anzahl an Tuning-Parametern und zeitintensiven Trainingsphasen, können wir nachweisen, dass Deep Learning-Netze gegenüber den üblichen lexikalischen Verfahren und maschinellen Lernverfahren ebenbürtig sind. Außerdem können wir die Modelle auf unserem Fall signifikant verbessern, sodass unsere Deep Learning-Modelle mit unseren Top-Verfahren mithalten können. Entgegen der allgemeinen Meinung stellen wir fest, dass Deep Learning-Modelle auf wenig Daten durchaus gut mit einer Genauigkeit bis zu 85 % abschneiden können. Ebenfalls können wir behaupten, dass die lexikalischen Verfahren mit einem geringen Trainingsdatensatz als sehr gute Klassifikatoren mit einer Genauigkeit bis zu 85 % arbeiten.

Nach unserem Modellbau beschäftigen wir uns mit dem Nachweis einer Verbindung zwischen dem Bitcoin-Niveau und unserem Sentiment, indem wir mit unseren Modellen Sentiments erstellen. Die tageweise erstellten Sentiments verbinden wir mit dem zugehörigen Bitcoin-Niveau. Aus unserer Zeitreihe untersuchen wir den Sentimentindex und können damit einen hoch signifikanten Bezug zu Bitcoin bis zwei Tage in die Vergangenheit nachweisen. Daraus folgt, ist die Stimmung positiv, kann das Bitcoin-Niveau steigen und umgedreht.

Bernhard Preisler

Schlagwörter: *Deep Learning, CNN, KNN, Sentimentindex, Text Mining, Twitter, Bitcoin, Animal Spirit*

Abstract

Quantifying Sentiment for the Price of Bitcoin using Deep Learning

The prediction of share prices plays an important role in history. The one who can predict the price most accurately wins. In the recent past external sources, such as moods in newspapers or articles, have enjoyed great popularity. Furthermore, recent publications weaken complete economic rationality and point significantly in the direction of “animal spirits”. Inspired by this combination we transfer this principle to the novel systems Bitcoin and the microblogging service Twitter.

To accomplish this, we download and save 10.31 million tweets from the 10th of January till the 31st of August 2018. Afterwards we use Amazon Mechanical Turk to create our training dataset, which we use for training the models. In the next step, we train established machine learning methods, such as Random Forest and SVM, and use lexical methods such as vaderSentiment and sentimentr. The main part of our work focuses on the creation and improvement of existing Deep Learning models. The first model HDLTex is inspired by K. Kowsari and is built exclusively by KNN layers. We take the second model from Y. Kim which consists of a mixture of KNN and CNN layers. Both models serve as a starting point, which we improve significantly our scenario in the course of the thesis.

In addition to an enormous number of tuning parameters and time-consuming training phases, we can demonstrate that Deep Learning networks are superior to the usual lexical and machine learning methods. Furthermore, in our case we can improve our models significantly, that the Deep Learning models can compete with our top procedures. Contrary to the general opinion we place that Deep Learning models can perform well with an accuracy till 85 % on a small amount of data. We can also say that the lexical procedures work with a small training data set as very good classifiers with an accuracy till 85 %.

After our model building we are concerned with proving a connection between the Bitcoin level and our sentiment index created by our models. We combine the sentiment created on a daily basis with the corresponding Bitcoin level. We take a sentiment index from our time series and can prove a highly significant reference to Bitcoin.

Bernhard Preisler

Key words: *Deep Learning, CNN, DNN, Sentimentindex, Text Mining, Twitter, Bitcoin, Animal Spirit*

Oft stolpert man über die anderen, öfter über sich selbst.

Indisches Sprichwort

Danksagung

Zuerst danke ich meinen Eltern, die einige schwierige Situationen mit mir mit Bravour gelöst haben und mich bis heute tatkräftig unterstützen trotz der Rechthabereien meinerseits.

Einen weiteren großen Dank richte ich an Katrin H., meiner Lebensgefährtin, die mir die letzten Jahre während des Studiums immer zur Seite stand und mich wirklich in jeder Situation unterstützt.

Ich danke Herrn Prof. Dr. Becker und Frau Prof. Dr. Mieskes, die mir die Möglichkeit gaben, die Masterthesis an der Hochschule Darmstadt zu schreiben und mir jederzeit mit ihrem professionellen Rat bei Seite standen.

Viel zu oft als selbstverständlich hingenommen, bedanke ich mich bei meinem Arbeitgeber, der mir ebenfalls das Studium ermöglichte, obwohl ich manchmal mehrere Wochen am Stück wegen dem Studium nicht abrufbar war.

Besonders bedanke ich mich bei meinen Freunden, die trotz der mittelstarken Abschottung immer noch meine Freunde sind.

Außerdem bedanke ich mich beim Daniel B., Peter B., Timo S. und Jan V., die mich beim Schreiben unterstützt haben.

Bernhard Preisler

Breisler.Pernhard@gmail.com

Alleine erreichen wir unsere Ziele nicht.

Inhaltsverzeichnis

1	Ergebnisse	1
2	Einführung	3
3	Beschreibung der Daten	5
3.1	Bitcoin	5
3.2	Twitter	6
3.2.1	Annotierte Tweets	7
3.2.2	Datenbereinigung & Feature Engineering	9
3.2.3	Zeitreihe	14
4	Deep Learning	15
4.1	Übersicht & Definitionen	15
4.1.1	Sentimentanalyse	16
4.1.2	Trainings- und Evaluationsdatensatz	17
4.1.3	Graphentheorie	18
4.1.4	Bias-Varianz-Tradeoff	19
4.2	Künstliche Neuronale Netze	20
4.2.1	Sprung zu Deep Learning	23
4.2.2	Aktivierungsfunktion	26
4.2.3	Parameterinitialisierung	27
4.2.4	Kostenfunktion	28
4.3	Regularisierungen	29
4.3.1	Frobiniusnorm	29
4.3.2	Dropout	29
4.4	Optimierungen	30
4.4.1	Mini-Batch	30
4.4.2	Weitere Optimierungsverfahren	30
4.5	Convolutional Neural Network	31
4.6	Recurrent Neural Network	33
4.7	Ergebnisse	35
4.7.1	HDLTex	35
4.7.2	CNNSC	38
4.7.3	Vergleich aller Modelle	41

5 Bitcoin und Sentiment	43
6 Ausblick	49
Literatur	53
Anhang	59
A Technische Rahmenbedingung	59
A.1 Downloadskript	59
A.2 Python	59
B Beschreibung der Daten	61
B.1 Twitter-Download-System	62
B.2 Annotierte Tweets	64
B.3 Datenbereinigung & Feature Engineering	65
B.3.1 Datenbereinigung	65
B.3.2 Feature Importance	66
B.4 Zeitreihe	69
B.5 Trainings- und Validierungsdatensatz	70
C Vergleichsmethoden	71
C.1 Sentiment Pakete	71
C.1.1 vaderSentiment	71
C.1.2 sentimentr	72
C.1.3 Ergebnisse	72
C.2 Maschinelle Lernverfahren	73
C.2.1 Random Forest	73
C.2.2 Support Vector Machines	75
C.2.3 Ergebnisse	76
C.2.4 Code	77
D Deep Learning	79
D.1 HDLTex	79
D.2 CNNSC	84
D.3 Ergebnisse	88
D.3.1 HDLTex	88
D.3.2 CNNSC	90
E Bitcoin und Sentiment	93
F Beschreibung des Anhangs	97
Index	99

Akronyme

Bezeichnung	Beschreibung
AMT	Amazon Mechanical Turk
ANN	Artificial Neural Network
AUC	Area Under Curve
CNN	Conventional Neural Network
CNNSC	Convolutional Neural Networks for Sentiment Classification
DL	Deep Learning
F1	F1-Score
FI	Feature Importance
FPR	False Positiv Rate
HDLTex	Hierarchical Deep Learning for Text Classification
KA	Krippendorf Alpha
KNN	Künstliches Neuronales Netz
LSTM	Long Short-Term Memory
LSVRC	Large Scale Visual Recognition Challenge
NLP	Natural Language Processing
Prec	Prediction
Rec	Recall
RF	Random Forest
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SVM	Support Vector Machines
TPR	True Positiv Rate
VADER	Valence Aware Dictionary for sEntiment Reasoning
WVR-Test	Wilcoxon-Vorzeichen-Rang-Test

Symbolverzeichnis

Bezeichnung	Beschreibung
σ	Aktivierungsfunktion
b	Bias
\mathbf{b}	Biasvektor
B	Bitcoin-Niveau
S_t	CNNSC-Sentiment
\mathbf{w}	Gewichtsmatrix
\mathbf{X}	Inputmatrix
$\mathbf{l}^{[0]}$	Inputvektor
α	Lernrate
L	Anzahl der Hidden-Layer
n	Anzahl der Neuronen
\mathbf{Y}	Outputmatrix
\hat{y}	Prognostizierter Wert eines Datenpunktes
y	Wahrer Wert eines Datenpunktes
Δ	Rendite von heute auf gestern

Abbildungsverzeichnis

1.1	BTC-Kurs und CNNSC-Modell	2
3.1	Sicht eines Turkers auf Amazon Mechanical Turk	8
3.2	Feature Importance mit Datenvorbereitung	11
4.1	Klassifizierungsübersicht von Deep Learning-Algorithmen	16
4.2	Darstellung eines Graphs	18
4.3	Bias-Varianz-Tradeoff	19
4.4	Perzepron eines Neuronalen Netzes	20
4.5	Schematischer Ablauf eines Lern-Algorithmuses	22
4.6	Auswirkung des Parameters <i>Lernrate</i> auf das Lernverhalten	23
4.7	Neuronales Netz	24
4.8	Zwei Hidden Layer Neuronales Netz	25
4.9	Einfaches Convolutional Neural Network Beispiel (Pooling)	31
4.10	Long Short-Term Memory Motivation	33
4.11	Long Short-Term Memory Zelle	34
4.12	Künstliches Neuronales Netz: Verlauf der Genauigkeit	36
4.13	Künstliches Neuronales Netz: Güte	37
4.14	Künstliches Neuronales Netz: ROC-Chart	38
4.15	Convolutional Neural Network: Referenzmodell	39
4.16	Convolutional Neural Network: Verlauf der Genauigkeit	40
4.17	Convolutional Neural Network: Güte	40
4.18	Convolutional Neural Network: ROC-Chart	41
5.1	Komponenten der Zeitreihen	44
5.2	BTC-Kurs und CNNSC-Modell	45
B.1	Verteilung der Hashtags - Top 20	61
C.1	ROC-Charts der lexikalischen Methoden	73
C.2	Beispiel für einen Random Forest	74
C.3	Beispiel für Support Vector Machine Klassifikation	75
C.4	ROC-Charts der maschinellen Lernmethoden	77
D.1	Modellvisualisierung vom HDLTex	82
D.2	Modellvisualisierung vom angepassten HDLTex	83
D.3	Modellvisualisierung vom CNNSC	86

D.4	Modellvisualisierung vom angepassten CNNSC	87
D.5	HDLTex: Genauigkeitsverlauf	88
D.6	Anangepasstes HDLTex: Genauigkeitsverlauf	89
D.7	Boxplot: Genauigkeit (HDLTex)	89
D.8	CNNSC: Genauigkeitsverlauf	90
D.9	Anangepasstes CNNSC: Genauigkeitsverlauf	91
D.10	Boxplot: Genauigkeit (CNNSC)	91
E.1	Korrelationsplot der Sentiments	94
E.2	Korrelationsplot der Sentiments mit log-Rendite	95

Tabellenverzeichnis

3.1 Explorative Kennzahlen der heruntergeladenen Tweets	7
3.2 Ausschnitt des Trainingsdatensatzes	9
3.3 Übersicht der Bereinigungsschritte	10
3.4 Explorative Kennzahlen der heruntergeladenen Tweets nach der Bereinigung	11
3.5 Vortrainierte Word2Vec-Modelle	13
3.6 Ausschnitt der Zeitreihe	14
4.1 Übersicht gängiger Aktivierungsfunktionen	26
4.2 Prognoseergebnisse aller Verfahren	42
5.1 Drei Tage Sentiment-Prognose für die Bitcoin-Rendite	47
B.1 Vollständige Beschreibung der Trainingsdaten	65
B.2 Bereinigung der Daten mittels der Feature Importance	68
C.1 Messergebnisse der lexikalischen Verfahren	73
C.2 Messergebnisse der maschinellen Verfahren	76
E.1 Sieben Tage Sentiment-Prognose für die Bitcoin-Rendite	96
F.1 Übersicht des physikalischen Anhangs	97

KAPITEL 1

Ergebnisse

Unsere ursprüngliche Hypothese ist herauszufinden, ob Meinungen auf Twitter sich auf das Bitcoin-Niveau auswirken, demnach der Nachweis der Nicht-Rationalität der Ökonomie. Konkret heißt das, dass wir weitere Einflussfaktoren neben dem Angebot- und Nachfrage-System für das Bitcoin-Niveau suchen. Ein weiterer Fokus der Arbeit liegt auf der Bildung von Sentimentindices errechnet aus unseren Deep Learning-Modellen. Alle Ergebnisse präsentieren wir in diesem Kapitel.

Zu Beginn laden wir mehrere Millionen Tweets, die einen Bezug zu Bitcoin aufweisen, über einem Zeitraum von 10 Monaten (10.01. bis 01.08.2018) herunter. Aus diesem Twitter-Datensatz generieren wir unseren Trainingsdatensatz, der aus 1.857 positiv oder negativ annotierten Tweets besteht, siehe Abschnitt 3.2. Diesen verwenden wir, um unsere Modelle zu trainieren.

Etablierte Modelle aus dem Anhang C, die wir zum Vergleich heranziehen, sind *vaderSentiment*, *sentimentr*, *Random Forest* und *SVM*. Unsere Deep Learning-Modelle entnehmen wir aus wissenschaftlichen Arbeiten, die wir nachbauen und für unseren speziellen Fall verbessern und anwenden. Zuerst bedienen wir uns an *Hierarchical Deep Learning for Text Classification (HDLTex)*, das ausschließlich KNN-Layer verwendet und trainieren dieses auf unserem Trainingsdatensatz, siehe Abschnitt 4.7.1. Wir können durch Reduzierung und veränderten Aufbau des Netzes unsere Ergebnisse signifikant verbessern. Als zweites Modell verwenden wir *Convolutional Neural Networks for Sentiment Classification (CNNSC)*, das aus einem CNN-Layer besteht, siehe Abschnitt 4.7.2. Dieses verbessern wir ebenfalls durch eine Erhöhung der Anzahl der Kernels, Verminderung der Kernelgröße und der Strafterme. Die stärkste Verbesserung stellen wir beim Austausch der Wort-Vektoren von Google News zu Twitter fest. Mit einer Genauigkeit von 85 % prognostiziert unser angepasstes CNNSC-Modell und das *vaderSentiment* die stärksten Ergebnisse. Alle anderen Deep Learning-Verfahren liegen entweder im ähnlichen Ergebnisbereich oder sind schlechter als unsere Vergleichsmethoden. Die Messergebnisse sind in der Tabelle 4.2 zu finden.

Nach der Erstellung und dem Vergleich erzeugen wir eine Zeitreihe, die aus dem Bitcoin-Kurs und unterschiedlichen Sentimentindizes besteht. Die Indizes generieren wir durch die Modelle, indem wir diese auf unseren Twitter-Datensatz anwenden. Bevor die Indizes gebildet werden können, führen wir eine Bereinigung mit Hilfe von Feature Importance (FI) durch, siehe Abschnitt B.3.2. Die Zeitreihe besteht aus 233 Tagen und 81 Spalten,

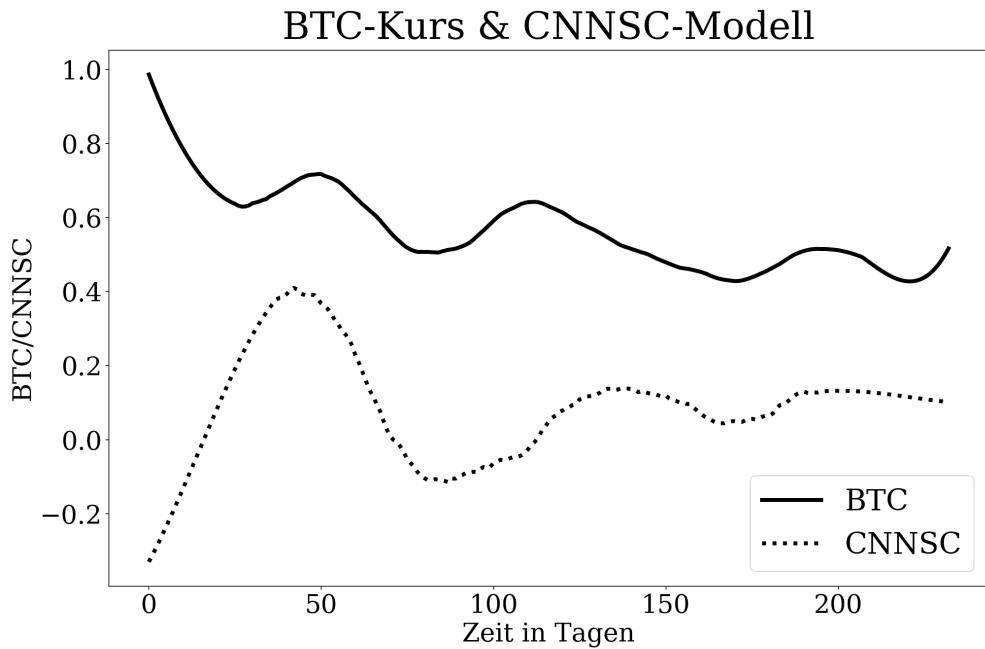


Abbildung 1.1: BTC-Kurs und CNNSC-Modell. Die x-Achse repräsentiert die Tage in einem Zeitraum vom 10.01. - 31.08.2018 und die y-Achse das Bitcoin-Niveau und dem errechneten Sentiment vom CNNSC-Modell.

siehe Abschnitt 3.2.3. Die Visualisierung 1.1 ist während der explorativen Analyse der Zeitreihe entstanden. Die x-Achse repräsentiert die Tage und die y-Achse das Bitcoin-Niveau, bzw. den errechneten Sentiment vom CNNSC-Modell. Die durchgezogene Linie zeigt den gemittelten Bitcoin-Kurs pro Tag und die gestrichelte Linie das CNNSC-Modell, das die positiven und negativen Tweets kumuliert widerspiegelt. Beide Reihen wurden mit dem Savitzky-Golay-Filter [Sav64] geglättet. Auffällig an dieser Visualisierung sind die charakteristisch ähnliche Schwankung beider Kurven. Daraufhin führen wir eine multiple lineare Regression mit und ohne ökonomische Variablen durch und stellen in jeden Fall eine hohe Signifikanz fest. Dies bestätigt unsere Hypothese, womit wir einen Bezug zwischen der Stimmung in Twitterkurznachrichten und dem Bitcoin-Niveau nachweisen konnten.

KAPITEL 2

Einführung

In einer Zeit mit einer quasi unbegrenzten Möglichkeit an frei zugänglichen Daten spielt die Textanalyse eine immer größer werdende Rolle [Män16]. Bereits ohne Computer legten erfolgreiche Unternehmen Wert auf die Meinung anderer und versuchten diese aus Zeitungen oder Meinungsumfragen zu gewinnen und mit ihrer persönlichen Meinung abzugleichen. Unsere initiale Hypothese wurde durch Cindy K. [Soo17] inspiriert. Als übergeordnetes Thema nimmt sie sich das Nachweisen von *Animal Spirits*, dt. animalische Instinkte oder das Herdentreiben, heraus, das Keynes J. [Key18] im Jahr 1936 prägt. Ursprünglich geht die Wirtschaft von einer rationalen Ökonomie aus. Kindleberger [Cha78] zeigt im Jahr 1978 im Zuge einer platzen Finanzblase, dass die Überzeugungen oder Meinungen von Investoren mit Zeitschriften und Nachrichten in Beziehung stehen. Als weitere Motivation führt Cindy K. [Soo17] unterschiedliche Studien von 1990 bis 2010 auf, die sich mit der Meinungsbildung beschäftigen. Dabei stellt sich heraus, dass typischerweise die Leser Nachrichten lesen, die mit ihren Vorstellungen Überschneidungen aufweisen. Trotzdem können die Medien durch ihre Argumentationen und Tonfall den Nutzer überzeugen. In Cindy's Veröffentlichung untersucht sie das Sentiment von 34 unterschiedlichen Lokalzeitschriften in Bezug zu deren Städten verteilt über den USA von 2000 bis 2013. Dabei entdeckt sie, dass ihr Sentiment eine um zwei Jahre versetzte hohe Korrelation in Bezug zum Immobilienindex aufweist.

Viele Werke, die das Thema *Animal Spirits* tangieren, sind durch sogenannte *Spekulationsblasen* [Gar01] motiviert. Solch eine Blase über ein Objekt bedeutet, dass der aktueller Preis wesentlich höher ist als der reale Wert des Objektes. Als wir mit Cindy's Werk [Soo17] in Kontakt treten, waren wir nach unserem Gefühl kurz vor einer Bitcoin-Blase. Außerdem arbeiteten wir vor dieser Arbeit an einer Sentimentanalyse auf Twitter. Diese Kombination ermöglicht uns den Transfer des Prinzips von Cindy auf die Systeme Twitter und Bitcoin und damit einen Bezug zwischen Twitter und Bitcoin herzustellen. Wir erwarten, dass Twitter schneller als die lokalen Zeitschriften ist, da im Gegensatz zu den Zeitungen Twitter ein Mikrobloggingdienst ist und dadurch eine kürzere Distanz zwischen der Stimmung von Twitter und dem Bitcoin-Niveau zu sehen sein könnte.

Zu Beginn beschreiben wir in Kapitel 3 unsere unterschiedlichen Datensätze, die während der Thesis anfallen. Um die Stimmung auf Twitter zu gewinnen, laden wir als erstes Tweets mit Hashtags über Kryptowährungen herunter, siehe Kapitel 3.2. Die Tweets bereiten wir so auf, dass die maschinellen Lernverfahren aus dem Anhang C und dem Abschnitt 4.7 diese verarbeiten können. Im nächsten Schritt generieren wir uns einen Trainingsdatensatz

mit Amazon Mechanical Turk (AMT), siehe Abschnitt 3.2.1. Dieser Datensatz dient als Grundlage unserer Modellbildung. Zum Vergleich ziehen wir erprobte Verfahren aus dem Bereich des maschinellen Lernens heran, siehe Anhang C. Ein großer Teil der Thesis beschäftigt sich mit der Modellbildung mittels Deep Learning, die wir vergleichen und verbessern, siehe Kapitel 4. Gegen Ende der Thesis erstellen wir mit unseren Modellen eine Zeitreihe in Abschnitt 3.2.3, die wir im Kapitel 5 diskutieren. Weiterhin versuchen wir mit dieser Zeitreihe ein Verhältnis zwischen der Stimmung auf Twitter und dem Bitcoin-Niveau herzustellen. Der Weg beginnend mit der Datenbeschaffung bis zur Analyse enthält viele Stellschrauben, die die Ergebnisse beeinflussen können. Diese besprechen wir im Ausblick, siehe Kapitel 6.

Pollyanna, Matheus, Fabrício und Meeyoung beschäftigen sich in ihrer Arbeit *Comparing and Combining Sentiment Analysis Methods* [Gon14] aus dem Jahr 2013 mit dem Vergleich unterschiedlicher Methoden, wie Emoticons [Rea05], LIWC [Tau10], SentiStrength [The13] u. v. m. Sie vergleichen ihre Ergebnisse mit fünf verschiedenen gelabelten Datensätzen. Sie finden heraus, dass maschinelle Learn-Methoden auf Twitternachrichten besser als lexikalische Methoden abschneiden [Gon14, S. 2]. Nach dem Vergleich kombinieren die Autoren alle Methoden miteinander und erlangen dadurch ihr bestes Modell und Ergebnis.

2017 wurde das Paper *When Bitcoin encounters information in an online forum: Using text mining to analyse user opinions and predict value fluctuation* [Kim18] veröffentlicht. Die Autoren beschäftigen sich ebenfalls mit der Preisvorhersage des Bitcoins mittels Kommentaren aus dem Bitcoin-Forum (<https://bitcointalk.org>) und drei weiteren Quellen [Kim18, S. 3]. Sie betrachten über einem Zeitraum von 3 Jahren die Kommentare der Nutzer und bilden ein Sentiment. Danach trainieren sie unterschiedliche Neuronale Netze mit 90 % der Daten und versuchen die restlichen 10 % vorherzusagen. Das beste Modell erreicht eine Genauigkeit von 80 % [Kim18, S. 9].

KAPITEL 3

Beschreibung der Daten

Unser Ziel ist das Bilden von Sentimentindizes aus Tweets von Twitter, die sich auf das Thema Bitcoin und allgemein auf Kryptowährungen beziehen. Um dies zu bewerkstelligen, laden wir Tweets von Twitter herunter und speichern diese, sieh Abschnitt 3.2. Danach bereiten wir die Tweets für die Lernverfahren auf, sodass wir diese Daten verwenden können, siehe Absatz 3.2.2. Da wir mit supervised Lernverfahren aus dem Anhang C und aus dem Abschnitt 4.7 arbeiten, die eine abhängige Variable (Target) benötigen, müssen wir Tweets annotieren lassen, siehe Abschnitt 3.2.1. Außerdem erzeugen wir eine Zeitreihe im Abschnitt 3.2.3 für die Zeitreihenanalyse aus dem Kapitel 5.

3.1 Bitcoin

Bevor wir tiefer in die Downloadmechaniken einsteigen betrachten wir *Bitcoin*. Satoshi Nakamoto [Nak08] sieht zwei Schwierigkeiten in der digitalen Überweisung von Geld. Normalerweise findet eine solche Überweisung immer über einem zentralen Punkt statt, beispielsweise Banken oder Online-Schnell-Überweisungsdienste. Dadurch erhalten diese Institute eine große Macht in der Finanzwelt. Dieses Problem löst Satoshi mittels einem Peer-to-Peer-Verfahren, das eine Dezentralisierung erlaubt. Ein weiteres Problem in der digitalen Welt ist Double-Spending. Wir können uns das Problem gut vorstellen, indem wir an unsere Urlaubsfotos denken und diese durch einen Messaging-Dienst teilen. Hier haben wir Double-Spending betrieben, weil wir die Fotos vervielfachen ohne das Originalbild zu löschen. Gelöst wird das Problem durch die Blockchain-Technologie. Hierbei werden alle Transaktionen wohldefiniert in einer Kette hintereinander gespeichert, sodass eine Historie entsteht. Diese Transaktionshistorie wird im Netzwerk gespeichert und an andere Server synchronisiert. Jeder Nutzer dieses Netzwerkes hat Zugriff auf diese Blockchain. Tätigt jemand eine Transaktion, wird diese durch Zurückrechnen vorheriger Transaktionen von anderen Nutzern bestätigt. Dieses Zurückrechnen wird umgangssprachlich als Mining bezeichnet. Die Sicherheit ist durch die Unveränderbarkeit der Kette gewährleistet.

Wir verwenden das Bitcoin-Niveau, Volumen und die Marktkapitalisierung für das Anreichern unserer Zeitreihe aus dem Abschnitt 3.2.3. Dabei repräsentiert das Bitcoin-Niveau das Tagesmittel des Bitcoin-Kurses.

3.2 Twitter

Twitter [Twi18] ist ein soziales Netzwerk und besteht Stand Q3 2017 aus ca. 330 Millionen Mikrobloggern [Twi17], die Inhalte konsumieren und erzeugen. Die registrierten Nutzer können Nachrichten bis zu einer Länge von 280 Zeichen auf ihrem Account veröffentlichen. Diese Nachrichten werden *Tweets*, abgeleitet von engl. tweet „zwitschern“, genannt und können Text, Fotos, Videos und Links enthalten. Wir interessieren uns in erster Linie für den Text. Veröffentlicht ein Nutzer seinen Tweet wird dieser seinen Followern, Personen die dem Nutzer folgen, angezeigt. Weiterhin versehen Nutzer die Tweets mit sogenannten *Hashtags*, die in der Textnachricht mit „#“ hervorgehoben werden und als Themenversehung dienen, z. B. „#Bitcoin“. Nutzer können gezielt nach diesen Hashtags suchen. Außerdem besteht die Möglichkeit Tweets zu teilen, auch *Retweets* genannt.

Um eine größere Menge von Tweets heranziehen zu können, müssen diese von Twitter gekauft oder fortlaufend gespeichert werden. Wir entscheiden uns für die zweite Variante und realisieren durch ein PHP-Skript und Cronjob einen kontinuierlich Download der Tweets, mehr im Anhang B.1 und über die technischen Rahmenbedingungen A.1. PHP ist eine Programmiersprache, die serverseitig ausgeführt wird. Ein Cronjob führt einen Auftrag in regelmäßig definierten Zeitabständen aus. Mittels dem PHP-Skript laden wir die Tweets über die bereitgestellte Twitter-API herunter. Die Twitter-API limitiert das Herunterladen der Tweets, indem lediglich die letzten 100 Tweets bereitgestellt werden.

Wie der Titel der Thesis verrät, entscheiden wir uns Tweets über das Themenfeld Kryptowährungen zu laden, da diese zu Beginn der Thesis ein aktuelles Thema darstellen. Dazu ist unser PHP-Skript so programmiert, dass nach definierten Hashtags gesucht wird. Die Auswahl der Hashtags basiert auf unseren subjektiven Eindrücken der Twitter-Plattform. Diesbezüglich betrachten wir die häufigsten Hashtags der beliebtesten Tweets und entnehmen deren Hashtags für unsere Suche. Beispielsweise speichern wir die Hashtags #bitcoin, #btc, #xrp, #cryptocurrency usw., mehr im Anhang B.1. Das Skript enthält die Liste der Hashtags und sucht im Minutentakt nach neuen Tweets. Die neuen Tweets speichern wir in eine temporäre Datei unterteilt nach dem jeweiligen Hashtag, die auf der Festplatte im JSON-Format abgespeichert wird. Da der Download jede Minute angestoßen wird, laden wir Tweets doppelt herunter, die wir ignorieren. Falls mehr als 15.000 Tweets erreicht wurden, wird die temporäre Datei in einen anderen Ordner kopiert und gelöscht. Der Grund für die Aufteilung der Tweets in verschiedenen Dateien ist das einfache Laden in den Arbeitsspeicher, da dieser begrenzt ist. Neben den Hashtags setzen wir den Parameter für die Sprache auf den englischsprachigen Raum.

Über die Laufzeit vom 13.09.2017 bis 01.08.2018 sammeln wir englischsprachige Tweets, siehe Tabelle 3.1. Da der Download zu Beginn fehlerhaft ist, schränken wir im Kapitel der Datenbereinigung 3.2.2 den Zeitraum auf den 10.01. bis 31.08.2018 ein. Der uneingeschränkte Twitter-Datensatz enthält 59,14 Millionen Tweets mit 1,36 Milliarden Wörtern. Der Median der Anzahl der Wörter pro Tweet beträgt 22 und pro Tag gesammelte Tweets 240.353.

Tabelle 3.1: Explorative Kennzahlen des gesamten Twitter-Datensatzes.

Beginn	13.09.2017
Ende	31.08.2018
Sprache	Englisch
Gesamtanzahl der Tweets	59,14 Millionen
Gesamtanzahl der Wörter	1,36 Milliarden
Anzahl der Tweets pro Tag (Median)	240.353
Wörter pro Tweet (Median)	22

Die Top 20 der Hashtags bestehen aus ca. 18 % allgemeinen themenbezogenen Hashtags, aus ca. 12 % anderen Kryptowährungen, aus ca. 11 % Bitcoin und aus ca. 7 % neuen Kryptowährungen, genauer in der Abbildung B.1 im Anhang.

Nach dem Download- und Speicherprozess sind 3.938 Dateien mit jeweils ca. 15.000 Tweets als JSON-Format vorhanden. Im nächsten Schritt lassen wir Tweets mittels AMT annotieren, siehe Abschnitt 3.2.1. Im Absatz 3.2.2 bereinigen und transformieren wir die Tweets so, dass der Datensatz von den Verfahren aus dem Anhang C und aus dem Abschnitt 4.7 verarbeitet werden kann. Außerdem beschreiben wir die Erstellung der Zeitreihe des Abschnitts 3.2.3 für das Zeitreihenanalysekapitel 5.

3.2.1 Annotierte Tweets

Amazon Mechanical Turk (AMT) ist ein Marktplatz von Amazon, um menschliche Intelligenz für bestimmte Aufgaben zu verwenden. Solche Aufgaben können Klassifizierung oder Labeling von Bildern sein, beispielsweise ob Bananen auf dem Bild vorhanden sind oder nicht. AMT unterteilt die Nutzer in zwei Gruppen. Die Requester speisen Aufgaben in das System ein und die Worker, auch Turks genannt, verarbeiten die Aufgaben gegen eine Gebühr. Diese Gebühr wird von den Requestern festgelegt und ist oftmals gering, siehe [For11, S. 417]. Fort et al. [For11] weisen auf die Problematik hin, dass der Verdienst der Turks oftmals unter 2 \$ die Stunde ohne Krankenversicherung oder ähnliches liegt. Wir einigen uns darauf dies bei unseren Requests zu beachten.

Neben der Unterbezahlung betrachten wir das System aus ethischer Sicht kritisch, da der Requester in der Lage ist die Bezahlung der Turks zu verweigern, wogegen der Turk keine Handlungsmacht besitzt. Diese Konstellation ist in Europa verboten [For11, S. 417].

Einen Teil der Tweets nutzen wir, um sie mittels AMT zu klassifizieren. Die Turks ordnen die Tweets einer Stimmung zwischen -4 und 4 zu. Zudem lassen wir einen Teil des Textes markieren, der relevant für die Entscheidung des Turkers ist.

Abbildung 3.1 zeigt die Sicht eines Turkers. Wählt ein Turk eine Stimmung aus, soll er als nächstes den Teil des Tweets markieren, der zu seiner Entscheidung beigetragen hat.

Instructions

- First read the tweet, assume the perspective of a cryptocurrency trader.
- Rate the tweet between -4 and 4 (-4 = extremely negative/crypto price will fall, 4 = extremely positive/ crypto price will rise). To get more information read our "Sentiment Analysis Instructions" below.
- Mark the text passage that is most important for your rating decision by left double clicking on the particular words. The words should appear comma separated in the textbox below.

Sentiment Analysis Instructions (Click to expand)

Evaluate Sentiment

Bitcoin Crash Sees Miners Fried in This Game of Chicken <https://t.co/uI9caJ2F8> Most bitcoin miners are losing money at current price (not to mention causing environmental disaster / wasting enormous electricity) #bitcoin #btc #crypto #bitcoincrash <https://t.co/vHiUtlPLHg>

Sentiment expressed by the content:

- | | |
|---|---|
| <input type="radio"/> [+1] Slightly Positive
<input type="radio"/> [+2] Moderately Positive
<input type="radio"/> [+3] Very Positive
<input type="radio"/> [+4] Extremely Positive | <input type="radio"/> [-1] Slightly Negative
<input type="radio"/> [-2] Moderately Negative
<input type="radio"/> [-3] Very Negative
<input type="radio"/> [-4] Extremely Negative |
| [0] Neutral (or N/A) | |

Marked words

<input style="width: 100%; height: 1.2em; border: 1px solid #ccc; padding: 2px; margin-bottom: 2px;" type="text"/>	<input style="border: 1px solid #ccc; padding: 2px; font-size: small; margin-bottom: 2px;" type="button" value="Clean marks"/>
(comma-separated words)	
<input style="background-color: #0070C0; color: white; border: 1px solid #0070C0; padding: 2px 10px; font-weight: bold; border-radius: 4px;" type="button" value="Submit"/>	

Abbildung 3.1: Sicht eines Turkers auf Amazon Mechanical Turk. Zu Beginn erhält der Turker eine Anleitung. In der grauen Box werden die unterschiedlichen Tweets dargestellt. Darunter wählt der Nutzer die Stimmung des Tweets aus. Durch Doppelklick auf Wörter des Tweets, kann der Turker die untere Textbox füllen. Der Turker ist dazu angehalten ausschließlich Wörter zu markieren, die seine Entscheidung beeinflussen. Als letzte Aktion verschickt er sein Ergebnis an AMT.

Da herkömmliche Klassifikatoren einen Fehler in der Klassifizierungsqualität aufweisen, entscheiden wir uns die Tweets von Personen annotieren zu lassen. Beispielsweise bedienten wir uns für die Auswahl der Tweets an vaderSentiment aus dem Abschnitt C.1.1 und stellten im Laufe der Thesis fest, dass vaderSentiment die Tweets nicht perfekt trennen kann. Ein weiterer Grund ist die spezifische Domäne der Kryptowährungen, denn in unterschiedlichen Bereichen enthalten Wörter eine andere Bedeutung.

Nach der Annotation der Tweets wandeln wir diese so um, dass die Algorithmen mit dem Datensatz arbeiten können, beschrieben im Anhang B.2. Wir lassen insgesamt 1.857 Tweets von jeweils 7 unterschiedlichen Turks annotieren. Der Median der Bearbeitungszeit eines Tweets liegt bei 30 Sekunden. Insgesamt beteiligten sich 159 Turks an den Bewertungen. Die Stimmungen teilen sich in 1.042 positive, 727 negative und 88 neutrale Tweets auf. Zu beachten gilt, dass schwach positive/negative Tweets mit einem Threshold von kleiner als 0,15 und größer als -0,15 als neutral eingestuft wurden. Uns ist an dieser Stelle bewusst, dass diese Stellschraube einen wesentlichen Einfluss auf die Resultate haben kann.

Ein weiterer interessanter Aspekt ist herauszufinden, wie qualitativ hochwertig Turks annotieren. Eine Möglichkeit dies zu überprüfen ist die Einigkeit der Turks pro Tweet für die Auswahl der Stimmung zu messen. Dazu dient der statistische Messwert Krippendorf Alpha (KA) [Kri03]. KA errechnet ein Ergebnis zwischen 0 und 1, wobei 0 für keine und 1 für vollkommene Übereinstimmung steht. Wir lassen unseren ursprünglichen Datensatz mit einer Bewertung von -4 bis 4 über KA laufen und erhalten einen KA-Wert von 0,13. Als Nächstes teilen wir den Datensatz in 1, 0 und -1 auf und erhalten 0,43. Danach identifizieren wir die Turks, die gegen die allgemeine Stimmung wählen. Damit erhalten wir einen KA von 0,54, genauer im Anhang *Code/Krippendorf Alpha.html*. Laut gängiger Meinung ist die Übereinstimmung ab 80 % gut und ab 90 % sehr gut, mit Werten unter 80 % existieren Nicht-Übereinstimmungen [Lom02, S. 8]. Da unsere Trainingsmenge klein ist, entscheiden wir uns den Krippendorf Alpha in diesem Fall zu ignorieren.

Hinweis: Die annotierten Tweets verwenden wir in der Thesis als Trainingsdatensatz.

Die Tabelle 3.2 zeigt einen Ausschnitt des Trainingsdatensatzes. Die Spalte *Durchschnitt* spiegelt die Durchschnittsbewertung der sieben Turkers wieder. *Tweet* zeigt den Text des Tweets und *Sentiment* die errechnete Stimmung aus den *Durchschnitt*, die wir als Target für die maschinellen Lernverfahren verwenden. Der vollständige Trainingsdatensatz ist im Anhang B.2 zu finden.

Tabelle 3.2: Die Tabelle repräsentiert einen Ausschnitt des Trainingsdatensatzes. Die Spalte *Durchschnitt* spiegelt die Durchschnittsbewertung der sieben Turkers wieder. *Tweet* zeigt den Text des Tweetes und *Sentiment* die errechnete Stimmung aus den *Durchschnitt*.

	Durchschnitt	Tweet	Sentiment
0	-2,285714	@SilverBulletBTC Damn, and I can not buy...	-1
1	0,428571	Gauthier-Mohammed: I will be a father of...	1
:	:	:	:
1855	-3,428571	Oh my! So many #scam these days...e	-1
1856	1,714286	New #Blockchain marketplace Repayme...	1

Um die lexikalischen und maschinellen Verfahren aus dem Anhang C sowie Deep Learning des Abschnitts 4.7 miteinander zu vergleichen, verwenden wir den vorgestellten Datensatz, indem wir eine Einteilung in eine Trainings- und Testmenge vornehmen. Zwecks der Nachvollziehbarkeit setzen wir einen Seed, genauer im Anhang B.5.

3.2.2 Datenbereinigung & Feature Engineering

In diesem Unterkapitel beschreiben wir die Erzeugung des Datensatzes für die Vergleichs- und Deep Learning (DL)-Modelle. Dazu werden die Tweets so aufbereitet, dass die Verfahren die Daten verarbeiten können.

Unterschiedliche Nutzer schreiben unterschiedliche Tweets. Die Texte weichen von der Struktur des Tweets, Rechtschreibung, Emojis u. v. m. enorm voneinander ab. Um die

Datenqualität zu erhöhen, führen wir unterschiedliche *Bereinigungsschritte* und Normalisierungen auf dem Twitter-Datensatz durch. Die wichtigsten Teile des Skripts sind im Anhang [B.3.1](#) zu finden. So wollen wir die Ergebnisse durch Anpassung der Datenqualität erhöhen. Die meisten Bereinigungsschritte entnehmen wir dem Paper [[Mar13](#), S. 403]. Die Messung der Qualität wird mit der FI aus dem Abschnitt [B.3.2](#) gemessen, die als Nebenprodukt beim Random Forest (RF) mitberechnet wird. In der folgenden Liste [3.3](#) stellen wir die Bereinigungsschritte vor, die zu einer höheren Qualität der Daten beitragen.

Tabelle 3.3: Übersicht der Bereinigungsschritte. Die Schritte begründen sich aus dem FI, genauer im Anhang [B.3.2](#)

Bereinigungsschritt	Beschreibung
<i>Doppelte Tweets</i>	Entfernen der doppelten Tweets über die Twitter-TweetID.
<i>Retweets</i>	Entfernen der Retweets. Retweets können als Zitate von anderen Nutzern angesehen werden, die auf dem eigenen Kanal veröffentlicht werden. Dadurch erhält dieser retweetete Tweet eine höhere Reichweite. Es entstehen doppelte Inhalte, die wiederum die Gewichtung der Stimmung beeinflussen. Wir sind uns bewusst, dass dies wiederum eine Stellschraube für die Qualität des Sentiments sein könnte, die in dieser Thesis nicht weiter verfolgt wird, siehe das Ausblickskapitel 6 .
<i>Uppercase zu Lowercase</i>	Uppercase zu Lowercase konvertieren, normalisiert die Texte auf Kleinbuchstaben [Mar13 , S. 403]. Sophia et al. [Cha18] sehen diesen Schritt kritisch, da die Wörter, die ausschließlich aus Großbuchstaben bestehen, durchaus einen stärkeren Ausdruck aufweisen können als kleingeschriebene Wörter.
<i>Stopwords</i>	Entfernen von Wörtern mit keinen Informationsgehalt.
<i>Hashtags</i>	Die vom Nutzer gewählten Hashtags sind nochmals im Tweet enthalten. Wir entfernen diese.
<i>Nutzer</i>	Nutzer könne andere Nutzer in den Tweets erwähnen, die mittels einem @ gekennzeichnet sind. Wir entfernen diese.
<i>HTML-Elementen</i>	In Tweets verwenden Nutzer HTML-Elemente, die wir entfernen.
<i>URLs</i>	Tweets erlauben Links/URLs mitzuliefern, die wir entfernen.

Als Ergebnis erhalten wir die 25 wichtigsten Features, siehe Abbildung 3.2. Die ersten elf Features üben einen größeren Einfluss auf die Klassifikation aus. Danach flacht die Informationsstärke ab und stagniert auf einem ähnlichen Niveau.

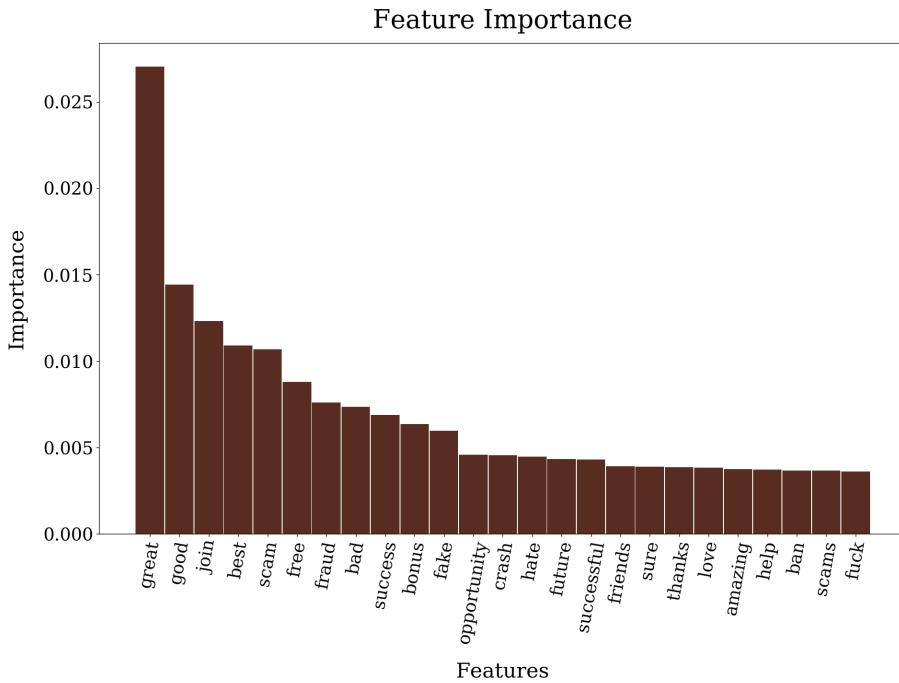


Abbildung 3.2: FI mit Datenvorbereitung (Entfernen von Kleinschreibung, Stopwörter, Hashtags, HTML-Code, URLs, Nutzer und Eigennamen).

Nach der Bereinigung erhalten wir über einen Zeitraum vom 10.01. bis 31.08.2018 einen Datensatz mit 10,31 Millionen Tweets, 156,2 Millionen Wörtern, einen Median der Anzahl der Wörter pro Tweet von 13 und einen Median der Anzahl der Tweets pro Tag von 40.081. Vergleichen wir den bereinigten Datensatz mit den ursprünglichen, verringern wir die Menge der Tweets auf 17,43 %, die Menge der Wörter auf 11,50 % und die Menge der Tweets pro Tag auf 16,70 %. Auch verringert sich der Median für die Wörter pro Tweet.

Tabelle 3.4: Explorative Kennzahlen der heruntergeladenen Tweets nach der Bereinigung.

	Voriger Wert	Neuer Wert
Beginn	13.09.2017	10.01.2018
Ende	01.08.2018	31.08.2018
Gesamtanzahl der Tweets	59,14 Millionen	10,31 Millionen
Gesamtanzahl der Wörter	1,36 Millionen	156,24 Milliarden
Anzahl der Tweets pro Tag (Median)	240.353	40.081
Wörter pro Tweet (Median)	22	13

Feature Engineering stellt einen eigenen Bereich der Datenvorbereitung dar [Zhe18]. Die Idee ist weitere Informationen aus den bereits existierenden Daten zu extrahieren oder externe Quellen, wie vortrainierte Modelle oder externe Datensätze, hinzuzuziehen. Weiterhin müssen die Tweets in ein Format umgewandelt werden, das die maschinellen Lernmethoden verwenden können, beispielsweise in Bag-of-words oder One-Hot-Kodierung. Außerdem verwenden wir im späteren Verlauf dieser Thesis vortrainierte Wort-Vektoren.

Bag-of-words

Zheng beschreibt *Bag-of-words* in “Mastering of Feature Engineering“ als einen Vektor von Zählungen [Zhe18, S. 16] von Wörtern, engl. vector of counts. Der Vektor besteht aus einer Liste von allen Wörtern, die im Vokabular vorkommen. Ein Dokument, in unserem Fall ein Tweet, wird auf diesen Vektor abgebildet, indem jedes Wort inkrementell hochgezählt wird. Enthält ein Dokument dreimal das Wort “Kryptowährung“, auch Token genannt, wird die Anzahl im vector of counts dementsprechend auf drei gesetzt.

Folglich erklärt sich auch der Begriff Bag-of-N-Grams [Zhe18, S. 21], der anstatt mit einem Wort mit N Wörtern arbeitet. Angenommen der gegebene Satz “Bitcoin is raising.“ wird als Bag-of-2-Grams gebildet, erhalten wir die Wortkombinationen “Bitcoin is“ und “is raising“. Bag-of-1-Grams ist eine weitere Bezeichnung von Bag-of-words.

One-Hot-Kodierung

In Deep Learning aus dem Kapitel 4 ist es gängige Praxis die Einteilung der Klassen One-Hot zu kodieren. D. h. jeder Klasse wird ein eindeutigen Vektor zugewiesen. Die Länge der Vektoren errechnet sich aus der Klassenanzahl. Jeder Vektor wird mit Nullen aufgefüllt und enthält eine Eins. Wird beispielsweise der Satz „*Kryptowährung ist neu.*“ betrachtet, beträgt die Klassenlänge 3. Der Vektor $[1, 0, 0]$ repräsentiert das Wort *Kryptowährung*, $[0, 1, 0]$ *ist* und $[0, 0, 1]$ *neu* [Har13, S. 129].

Word2Vec

Word Embedding repräsentiert einen Sammelbegriff für Phrasen, die in einen Vektor mit reellen Zahlen übersetzt werden. Solch ein Verfahren ist Word2Vec [Mik13]. Das Verfahren beginnt Wortpaare mittels Skip-Grams zu bilden und setzt diese Wortpaare in eine Relation zueinander. Je häufiger eine Wortkombination auftritt, desto größer ist die Wahrscheinlichkeit ihrer Zusammengehörigkeit. Wurden die Wortkombinationen generiert, werden diese mittels eines Neuronalen Netzes trainiert. Dazu dient ein linearer Hidden Layer als Look-Up-Table und die Wahrscheinlichkeit jedes Wortes wird durch den Output-Layer zugewiesen. Ein einfaches Beispiel (3.1) zeigt die Look-Up-Table.

$$[0 \ 0 \ 1 \ 0] = \begin{pmatrix} 14 & 3 & 0 \\ 2 & 5 & 18 \\ \mathbf{18} & \mathbf{22} & \mathbf{9} \\ 4 & 53 & 11 \end{pmatrix} = [18 \ 22 \ 9] \quad (3.1)$$

Als Input dient der Vektor $[0 \ 0 \ 1 \ 0]$, wobei dieser ein Wort widerspiegelt. Der Output ist ein Gewichtungsvektor. Im Rahmen der Thesis verwenden wir diese Technik, um eingebettete Vektoren im Deep Learning zu verwenden, siehe Abschnitt 4.7.2.

Eine Liste der vortrainierten Wortvektoren, die wir zwecks dem Training der Modelle im Kapitel 4.7.2 verwenden, stellen wir durch die Tabelle 3.5 bereit.

Tabelle 3.5: Vortrainierte Word2Vec-Modelle [Pen14] und [Kim14]. In der Tabelle sind die vortrainierten Word2Vec-Modelle zu finden, die wir in der Arbeit testen.

Quelle	#Vokabeln	Größe der Wordvektoren	Größe
Wikipedia 2014 + Gigaword 5	0,4 Millionen	50, 100, 200, 300, 400	0,82 GB
Common Crawl	1,9 Millionen	300	1,75 GB
Common Crawl	2,2 Millionen	300	2,03 GB
Twitter	1,2 Millionen	25, 50, 100, 200	1,42 GB
Google News	3,0 Millionen	300	1,50 GB

Stopwords

In einem Text stehen oft Wörter, die keine Information vermitteln und somit als Füllwörter aufgefasst werden können. Z. B. gehören die Wörter ‘der’, ‘man’, ‘im’ u. v. m. zu diesen sogenannten Stoppwörtern, engl. stopwords [Zhe18, S. 27]. Daher werden sie häufig aus den Texten entfernt. Ein weiterer Vorteil ist die Reduktion der Dimension am Input von Lernverfahren, die wir im Verlauf der Arbeit kennenlernen.

Weitere Datenbereinigungs- & Feature Engineering Schritte

Die folgende Liste enthält weitere Verfahren, die wir nicht in dieser Thesis betrachten, jedoch einen wesentlichen Einfluss auf die Features haben können.

- Das *Frequency-based filtering* [Zhe18, S. 27] bedient sich der Häufigkeiten an Wörtern. Beispielsweise ist ein Wort, dass in Millionen Dokumenten nur zweimal auftaucht mehr ein Rauschen anstatt einer nützlichen Information.
- Mit *Stemming* [Zhe18, S. 30] werden Wortstämme gebildet, die zu einer Reduktion der Anzahl von Wörtern führt, da die hohe Vielfalt an unterschiedlichen Wörtern mittels eines Wortstamms zusammengefasst wird.
- Die *Rechtschreibkorrektur* kann gerade bei Twitter zu signifikanten Verbesserungen führen. Dazu zählen Edit Distance [Man09, S. 59], das die Distanz zwischen zwei Wörtern berechnet, und k-gram Indizes [Man09, S. 60], das die Wörter oder Sätze in n-große Teile zerlegt und diese mittels Permutationen vergleicht.

3.2.3 Zeitreihe

In diesem Unterkapitel beschreiben wir die Generierung unserer Zeitreihe aus unseren Vergleichs-, DL-Modellen und Bitcoin-Niveau aus dem Abschnitt 3.1. Eine vollständige Beschreibung der Zeitreihe stellen wir im Anhang B.4 bereit.

Nachdem wir im Anhang C und Abschnitt 4.7 die Modelle trainieren und verbessern, verwenden wir diese, um mit dem heruntergeladenen Twitter-Datensatz aus dem Abschnitt 3.2.2 mehrere Sentiments über den gesamten Zeitraum 10.01. - 31.08.2018 zu bilden. Bevor wir loslegen wenden wir unsere Bereinigungsschritte an, siehe Abschnitt 3.2.2. Wir iterieren jeden Tweet und berechnen zu erst dessen Sentiment. Als Ergebnis erhalten wir für jeden Tweet eine Stimmung. Danach kumulieren wir jede Stimmung für jedes Modell tageweise, siehe Formel (3.2).

$$S_{mt} = \#positiv_{mt} - \#negativ_{mt} \quad (3.2)$$

wobei m für das Modell und t für die Periode steht. Nach den Berechnungen fügen wir das Bitcoin-Niveau [OpC13] und weitere ökonomische Variablen hinzu, genauer im Kapitel 5. Danach bilden wir über alle Spalten die log-Renditen, da wir die Veränderung des Bitcoins prognostizieren wollen, siehe Formel (3.3).

$$R_{mt} = \log(S_{mt}) - \log(S_{mt-1}) \quad (3.3)$$

Die Zeitreihe besteht aus 233 Datenpunkten mit 81 Spalten, wovon die Hälfte die umgewandelten log-Renditen sind, siehe Tabelle 3.6. Jede Spalte steht für ein Sentiment eines Modells mit Ausnahme von BTC und BTC*. Mit * kennzeichnen wir auf die Umwandlung in die log-Rendite.

Tabelle 3.6: Zeitreihe bestehend aus Sentiments. Die Überschriften, außer das BTC-Niveau, die BTC-Rendite und die ökonomischen Variablen, spiegeln das Sentiment der jeweiligen Modelle wieder. Unter * verstehen wir die log-Rendite der jeweiligen Modelle. Die Zeitreihe besteht aus 233 Datenpunkten mit 81 Spalten.

#	RF	...	CNNSC	BTC	RF*	...	CNNSC*	BTC*
1	6.816	...	-3.202	13.296,79	-0,28	...	-0,02	-0,22
2	9.619	...	-673	13.912,88	0,34	...	0,04	0,08
:	:	..	:	:	:	..	:	:
232	9.370	...	1.526	6.932,66	0,04	...	-0,03	-0,11
233	8.464	...	1.164	6.981,95	-0,10	...	-0,05	0,05

KAPITEL 4

Deep Learning

Deep Learning (DL) löste in den vergangenen Jahren einen großen Hype aus. DL sorgte für eine regelrechte Faszination bezüglich der Funktionalität, die dem menschlichen Gedächtnis sehr ähneln soll, aus. Auf einen Schlag konnten Probleme mit mathematischem Hintergrund gelöst werden, die mit herkömmlichen maschinellen Lernmethoden unmöglich erschienen. Dabei zeigte das Verfahren in der jüngsten Vergangenheit, dass die Erkennung eines Bildes [Rus15], die Spracherkennung [Gra13] oder auch Spiele wie Schach [Sil17a] und Go [Sil17b] geeignete Herausforderungen für DL-Algorithmen darstellen.

Die Geschichte von DL beginnt in den 1940er bis 1960er, in der die Methodik als *connectionism* und *cybernetics* bekannt war. Der erste Hype begann mit *connectionism* im Jahre 1970 als versucht wurde Neuronen zu simulieren. Danach holte *cybernetics* im Jahre 1990 auf und ist heute als Deep Learning bekannt. Einer der ersten Lern-Algorithmen ist das Künstliche Neuronales Netz (KNN), engl. Artificial Neural Network (ANN), siehe Abschnitt 4.2, das sich an den biologischen Neuronen orientiert. Die Begrifflichkeiten, unter der wir Neuronale Netze heute kennen, begannen um das Jahr 2006 [Goo16].

Dieses Kapitel bietet zu Beginn eine grobe Übersicht über die DL-Verfahren und führt die wichtigsten Begriffe ein. Danach betrachten wir die Netze KNN und Conventional Neural Network (CNN) genauer aus den Abschnitten 4.2 und 4.5. Zum Schluss bedienen wir uns an zwei Netzen, konstruieren sie nach und versuchen diese auf unserem Trainingsdatensatz aus dem Absatz 3.2.1 zu verbessern, siehe Abschnitt 4.7. Abschließend erfolgt eine Diskussion der Ergebnisse.

4.1 Übersicht & Definitionen

Um einen ersten Überblick über die Gesamtheit des Themas zu verschaffen, führen wir Begrifflichkeiten ein. Abbildung 4.1 zeigt eine Einteilung der DL-Verfahren in *supervised*, *unsupervised* und *Reinforcement Learning*. *Supervised Learning*, dt. überwachtes Lernen, benötigt immer einen Datensatz, der gelabelt ist, also (x,y) , mit $x \in X$ als Daten und $y \in Y$ als Label. Das Labeling wird im maschinellen Lernen als Target und in Deep Learning als Output bezeichnet. Beispielsweise könnten Katzenbilder mit dem Label *Katze* und Bananenbilder mit dem Label *Banane* versehen werden. *Unsupervised Learning*, dt. unüberwachtes Lernen, benötigt nur den Datensatz und kein Target. Das Verfahren versucht ähnliche Muster in den Daten zu erkennen und diese in Gruppen mit ähnlichen Eigenschaften

einzuordnen. Ein weiteres Themenfeld ist *Reinforcement Learning*, dt. bestärkendes Lernen, bei dem der Agent, der mittels sich immer wiederholender Szenarien eigenständig eine Strategie durch erhaltene Belohnungen und Bestrafungen erarbeitet. Der Agent benötigt in jeden Schritt die aktuelle Umgebung und die möglichen Aktionen. Führt er eine Aktion aus, ändert sich die Umgebung und dadurch entsteht eine Belohnung oder Bestrafung. Die neuen Informationen erhält der Agent und versucht mittels dieser seine Strategie zu entwickeln. Im Paper [Mni15] präsentieren die Autoren eine Liste von unterschiedlichen Videospielen, indem professionelle Spieler gegen Agenten spielen und vergleichen diese. Dabei stellte sich heraus, dass der Agent nach einer kurzen Trainingsphase wesentliche bessere Ergebnisse lieferte als professionelle Spieler. Beispielsweise handelt es sich um Spiele wie Video Pinball, Road Runner, Pong u. v. m. Der Fokus der Thesis liegt auf Supervised Learning, da unser Trainingsdatensatz gelabelt ist.

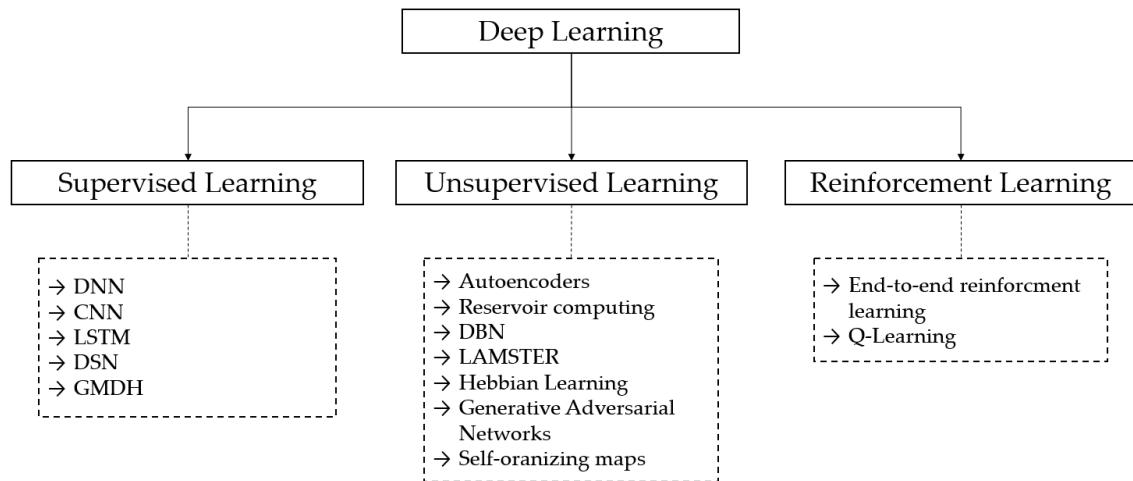


Abbildung 4.1: Klassifizierungsübersicht von DL-Algorithmen [Mni15] und [Goo16]. Zu sehen ist die grobe Einteilung in überwachtes (supervised), nicht überwachtes (unsupervised) und bestärkendes (reinforcement) Lernen.

Weiterhin führen wir in diesem Kapitel DL und die zugehörigen Begriffe ein.

4.1.1 Sentimentanalyse

Die Meinungen Anderer spielt nicht nur in der heutigen Zeit eine Rolle, auch vor dem Internet haben sich die Menschen Gedanken über andere Meinungen gemacht. Beispielsweise verglichen Unternehmen oder politische Parteien deren Eigenwahrnehmung mit der Außenwelt durch Zeitungen oder Mundpropaganda. Heute kann auf einen Schlag auf eine Vielzahl von Meinungen zugegriffen werden, weswegen die Bedeutung des Themengebietes *Sentiment* in den letzten Jahren stieg [Pan08].

Mäntylä M. et al. [Män16] untersuchen die Entwicklung der Sentimentanalyse. Dabei stellen sie fest, dass in den 90ern die Computerlinguistik und um das 20. Jh. die Meinungsanalyse stark vertreten sind. Sie untersuchen 6.996 Paper und finden heraus, dass 99 % davon nach 2004 veröffentlicht wurde.

Ein Sentiment ist eine Meinung über ein Thema. Mit der *Sentimentanalyse*, auch Sentiment Detection genannt, wird versucht die Meinung oder Stimmung eines geschriebenen Textes automatisch zu erkennen. Das Ergebnis einer solchen Analyse kann sich in verschiedener Weise äußern, wie 0 oder 1, Gut oder Schlecht, [1 – 9] oder sogar Gefühle wie Traurig/Glücklich [Pan08].

Die Sentimentanalyse ist ein Untergebiet von Text Mining und ein Teilgebiet der maschinellen Verarbeitung natürlicher Sprache, engl. Natural Language Processing (NLP). Mittlerweile erreichen die Wörterbücher einen Stand, mit dem ein Sentiment ohne Schwierigkeiten aus einfachen Sätzen extrahieren kann. Beispielsweise können die folgenden Sätze ohne Probleme in Positiv/Negativ unterteilt werden kann, sofern das Wörterbuch dafür ausgelegt ist.

- Heute ist das Wetter schön.
- Der Bitcoin steigt steil an.
- Das Fußballspiel war gestern fantastisch.

Schwerer fällt die Sentimentbildung, wenn Eigenheiten, wie Doppeldeutigkeiten oder Verneinungen, auftauchen.

- Heute ist das Wetter nicht schön.
- Sie hatten einen dekadenten Nachtisch.

Lexikalische Verfahren, wie vaderSentiment und sentimentr aus dem Anhang C.1, enthalten Logiken, die in vielen Fällen kompliziertere Satzstrukturen erkennen und entsprechend einordnen können.

4.1.2 Trainings- und Evaluationsdatensatz

Bevor wir in die Tiefen des DLs eintauchen, müssen wir die Begriffe *Trainingsdatensatz*, *Evaluationsdatensatz*, *Testdatensatz*, *Lernen*, *Trainieren* und *Fitting* klären.

Üblicherweise werden die Daten, die für das Training verwendet werden, in einem DL-Projekt nach [Ng15] in einen Trainings-, Evaluations- und Testdatensatz unterteilt. Der *Trainingsdatensatz* wird in einem DL-Netz verwendet, um das Modell zu trainieren bzw. zu fitten oder zu lernen. *Fitting* ist das englische Wort für Einpassen und bezieht sich in diesem Fall auf die Anpassung der Gewichte der einzelnen Neuronen des Netzes. Im DL-Jargon heißt das, „Wir fitten das Modell.“ oder „Wir trainieren das Modell.“. Während das Modell gefittet wird, wird nach jeder Epoche der Verlust aus dem Abschnitt 4.2.4 berechnet. Eine *Epoche* durchläuft den kompletten Datensatz. Meistens durchlebt ein Fitting-Prozess mehrere Epochen. Der errechnete Verlust kalkuliert sich auch aus dem Trainingsdatensatz. Mit dem *Evaluationsdatensatz* wird während dem Training geprüft, wie gut das Modell auf untrainierten Daten abschneidet. Nach dem Abschließen des Fittings wird der Messwert für den Testdatensatz berechnet. Der *Testdatensatz* ist die reale Umgebung in der das Modell eingesetzt wird. Sollten zu wenig Daten in einem Projekt vorhanden sein, kann der Evaluations- und Testdatensatz zusammengefasst werden.

Eine weitere große Frage ist die prozentuale Einteilung des Datensatzes in eine Trainings- und Evaluationsmenge. Die Unterteilung kann auf verschiedene Arten stattfinden und wir meistens in Prozent angegeben. Beispielsweise können die Daten direkt nach 75 % getrennt werden. Daraus ergibt sich eine Trainingsmenge von 75 % und eine Evaluationsmenge von 25 %. Weiterhin kann die Menge nach 75 % getrennt oder zufällig gezogen werden. Das Verhältnis von Trainings- und Evaluationsdatensatz ist nach Andrew Ng [Ng15] mit 80/20 solide. Die Auswahl des Verhältnisses hängt jedoch stark von der Art der Daten ab, weswegen in diesem Bezug keine genaue Aussage getroffen werden kann. Weiterhin hat sich herausgestellt, dass bei sehr großen Datensätzen eine Unterteilung von eher 99/1 sinnvoller erscheint, da genug Daten zur Validierung vorhanden sind.

Wir wählen ein zufälliges Verhältnis von 80/20 für die Trainings- und Evaluationsmenge. Unser Testdatensatz ist der gesamte Twitterdatensatz aus dem Kapitel 3.2. Wir erhalten eine Trainings- und Validierunggröße von 1.485 und 372 Datenpunkten. Die Trainingsmenge enthält 848 positive, 569 negative und 68 neutrale Tweets. Die Validierungsmenge enthält 192 positive, 158 negative und 20 neutrale Tweets. Wir verwenden diesen Splitt für das Training der Modelle.

4.1.3 Graphentheorie

Im Bereich der Neuronalen Netze ist es eine gängige Methode die Netze durch Graphen darzustellen. Bevor wir in dieses Themengebiet eintauchen, definieren wir Graphen nach [Goo16].

Ein Graph besteht aus einem oder mehreren Knoten und Kanten. Die Knoten werden durch Kanten verbunden. Angenommen in einem Graph existieren zwei *Knoten A* und *B*, dann können diese mittels einer *Kante* (A,B) verbunden werden, siehe Abbildung 4.2. Der Verbund der Kanten kann ungerichtet a) oder gerichtet b) und c) sein. Bei den *ungerichteten* Kanten existiert die Verbindung in beide Richtungen $(A,B) = (B,A)$, sind die Kanten *gerichtet* gilt $(A,B) \neq (B,A)$.

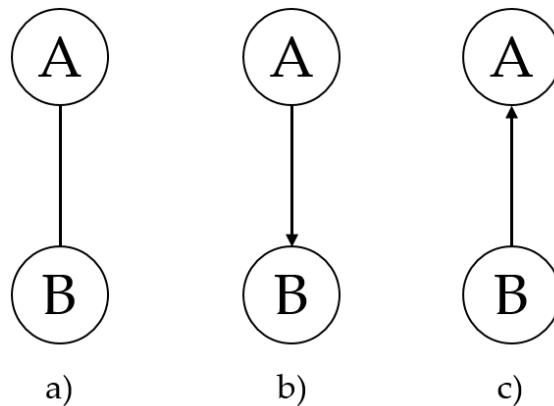


Abbildung 4.2: Darstellung eines Graphs. a) ist ein ungerichteter Graph, der eine Verbindung von (A,B) und (B,A) darstellt. b) zeigt einen gerichteten Graph mit einer Verbindung von (A,B) . c) ist die umgekehrte Variante von b) mit (B,A) .

Im Kontext der Neuronalen Netze werden meistens gerichtete Graphen verwendet. Hierbei wird der Knoten als *Variable* definiert, die ein Skalar, eine Matrix, ein Tensor oder andere Typen von Variablen sein kann. Weiterhin werden die Kanten als *Operationen* betrachtet. Die Operationen bestehen aus eingehenden Variablen und errechnen ein Ergebnis, das ein Skalar, Vektor, Matrix u. v. m. sein kann [Goo16].

4.1.4 Bias-Varianz-Tradeoff

In Data Mining- oder maschinellen Lern-Projekten stößt man früher oder später auf das Problem, wie mit dem Bias und der Varianz umgegangen werden soll, dem sogenannten Bias-Varianz-Tradeoff.

Abbildung 4.3 illustriert ein Klassifizierungsproblem mit dem Bias-Varianz-Tradeoff. Das Koordinatensystem a) zeigt ein Modell mit einem hohen Bias und einer niedrigen Varianz. Offensichtlich ist unser Modell schwach in Bezug auf die Güte der Klassifikation, auch *Underfitting* genannt. Unser Modell ist robust aber ungenau. Eine hohe Varianz zeigt b), indem das Modell einen niedrigen Bias und jeden Datenpunkt sauber trennt. Es liegt eine Überanpassung vor, auch *Overfitting* genannt. Die Klassifikation kann bei einem gering abweichenden Datenpunkt fehlschlagen. Anders gesagt, unser Modell ist gut in der Klassifizierung der Trainingsmenge, aber nicht robust gegenüber neuen Daten. c) zeigt einen ausgeglichenen Bias und Varianz. Diese Modelleigenschaften führen zu einer hoffentlich robusten und guten Prognose von neuen Werten.

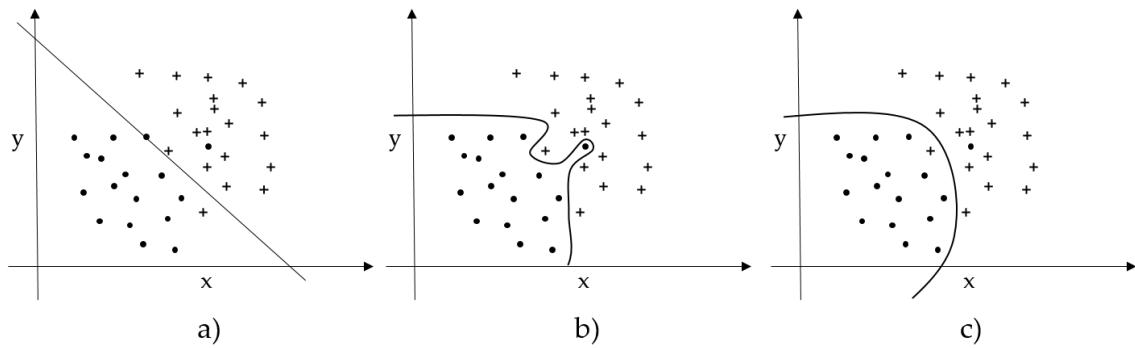


Abbildung 4.3: Die drei Plots zeigen den Bias-Varianz-Tradeoff. a) enthält einen hohen Bias und eine niedrige Varianz. b) zeigt einen niedrigen Bias, jedoch eine hohe Varianz. c) illustriert einen ausgeglichenen Bias und Varianz.

4.2 Künstliche Neuronale Netze

Dieses Unterkapitel dient zur Einführung der Neuronalen Netze, da sie ein Bestandteil von DL sind. Zu Beginn betrachten wir die einfachste Variante eines Neuronalen Netzes und werden über den Ablauf der Berechnungen des einfachen Netzes sprechen. Danach erweitern wir die Idee auf ein komplizierteres Netz und abstrahieren diese nach Andrew NG [Ng15].

Hinweis: Vektoren schreiben wir klein und fettgedruckt und Matrizen groß und fettgedruckt.

Ein Künstliches Neuronales Netz (KNN), hier bezeichnet als Neuronales Netz, wurde vom menschlichen Gehirn inspiriert. „Das Gehirn ist ein hochkomplexer, nicht-linearer und paralleler Computer.“, schreiben [Hay04, S. 23]. Das menschliche Gehirn organisiert seine Fähigkeiten durch Komponenten, bekannt als *Neuron* oder *Perzeptron*, die miteinander verbunden sind. Später sehen wir im Abschnitt 4.3.2, dass nicht jedes Neuron verwendet werden muss. Diese natürliche Inspiration dient als Grundlagen für KNNs. Abbildung 4.4 zeigt ein einfaches *Neuronales Netz* mit vier Features (Maschinelles Lernen), auch Inputs (Neuronales Netz) genannt, und einer verdeckten Schicht, engl. *Hidden Layer*. Der Begriff Layer repräsentiert die weiteren Schichten, die keine oder einfache Berechnungen durchführen, beispielsweise der Input- oder Output-Layer. w steht für die Verbindungen der Kanten zu den Neuronen, \mathbf{X} für die Inputs und b oder \mathbf{b} für den Bias, siehe Abschnitt 4.1.4. σ steht für die Aktivierungsfunktion, die wir im Unterkapitel 4.2.2 genauer untersuchen.

Um ein tieferes Verständnis zu erlangen, werden wir im Folgenden den Lern-Algorithmus mit der Abbildung 4.5 erarbeiten [Ng15]. Zwecks der Einfachheit verzichten wir in den kommenden Absätzen auf die Dimensionen der Parameter und Variablen, genauer im Abschnitt 4.2.1.

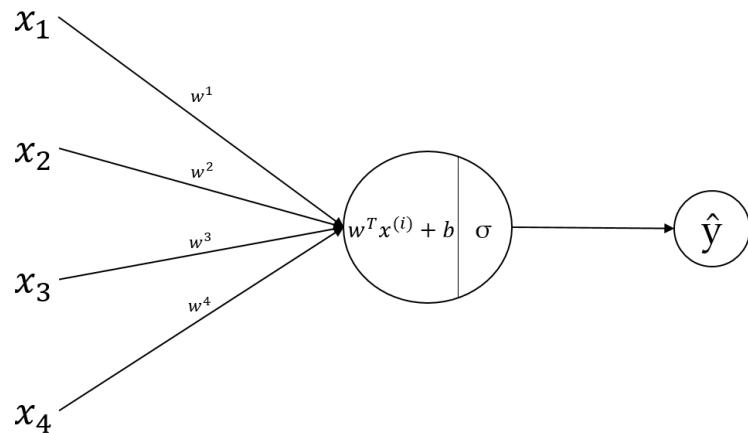


Abbildung 4.4: Ein Neuronales Netz mit vier Inputs, einem Hidden Layer mit einem Neuron, bzw. Perzeptron, und einem Output.

Wir beginnen mit einem gelabelten Trainingsdatensatz (\mathbf{X}, \mathbf{Y}), wobei \mathbf{X} der Input und \mathbf{Y} der Output darstellt. Als Nächstes werden die Nullvektoren initialisiert. Hat \mathbf{X} eine Dimension von 2 wird $\mathbf{w} = [[0][0]]$ und $b = 0$ initialisiert. Die Gewichtung \mathbf{w} und der Bias b

werden danach mit Zufallszahlen befüllt, beispielsweise mit $\mathbf{w} = [[1][2]]$ und $b = 2$, genauer im Abschnitt 4.2.3. Außerdem gilt zu beachten, dass b in diesem Sonderfall kein Vektor ist, da es sich nur um ein Neuron handelt. Nachdem die benötigten Parameter alle initialisiert wurden, können wir im nächsten Schritt die Forward- und Backward-Propagation betrachten. Wir beginnen mit der Forward-Propagation. Das heißt, wir berechnen, propagieren, die Werte des Netzes beginnend beim Input über den Hidden-Layer bis zum Output-Layer, in der Abbildung 4.4 von links nach rechts. Mathematisch wird der Hidden-Layer mit einem Neuron, bzw. Perzepron, mit der Formel (4.1) dargestellt.

$$\hat{y} = \sigma(z) = \sigma(\mathbf{w}^T \mathbf{x} + b), \quad (4.1)$$

wobei n die Anzahl der Neuronen widerspiegelt und $z = \mathbf{w}^T \mathbf{x} + b$. Mit \hat{y} berechnen wir den Verlust (4.2), engl. Loss. Die Verlustfunktion beschreiben wir mit $L(\hat{y}, y)$. Der Verlust, hier die Cross-Entropy [Ng15, Folie C1W2L03], wird über den verwendeten Datensatz berechnet, genauer im Abschnitt 4.2.4. Dadurch kann eruiert werden, ob sich das Modell verbessert oder nicht. \hat{y} sind unsere vorhergesagten und y unsere realen Ergebnisse.

$$L(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}) \quad (4.2)$$

Nachdem vorwärts propagiert wurde, propagieren wir als nächstes rückwärts, engl. *Backward-Propagation*. Dies geschieht, indem die Ergebnisse vom Output zum Input propagiert werden, auf der Abbildung 4.4 von rechts nach links. Die Back-Propagation wird berechnet, indem jeder einzelne Knoten jedes Layers abgeleitet wird, auch Gradientenverfahren [Ng15, Folie C1W3L09] genannt, zu sehen in den Formeln (4.3) und (4.4). Beide Formeln verwenden verkettete Ableitungen, um die Gewichte \mathbf{w} und b zu erneuern, siehe die Formeln (4.5) und (4.6). Dieser Vorgang wird in großen Netzen mit großer Tiefe und Breite sehr aufwendig.

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial z}{\partial \mathbf{w}} \frac{\partial a}{\partial z} \frac{\partial L}{\partial a} \quad (4.3)$$

$$\frac{\partial L}{\partial b} = \frac{\partial z}{\partial b} \frac{\partial a}{\partial z} \frac{\partial L}{\partial a} \quad (4.4)$$

Um den Ablauf zu vereinfachen, verzichten wir auf die Ableitung der Aktivierungsfunktion. Nachdem propagiert wurde, schließt sich die Lernphase mit

$$\mathbf{w} = \mathbf{w} - \alpha \frac{\partial L}{\partial \mathbf{w}} \quad (4.5)$$

und

$$b = b - \alpha \frac{\partial L}{\partial b} \quad (4.6)$$

an, wobei α die *Lernrate* widerspiegelt. Sie ist ein Hyperparameter (siehe den unteren

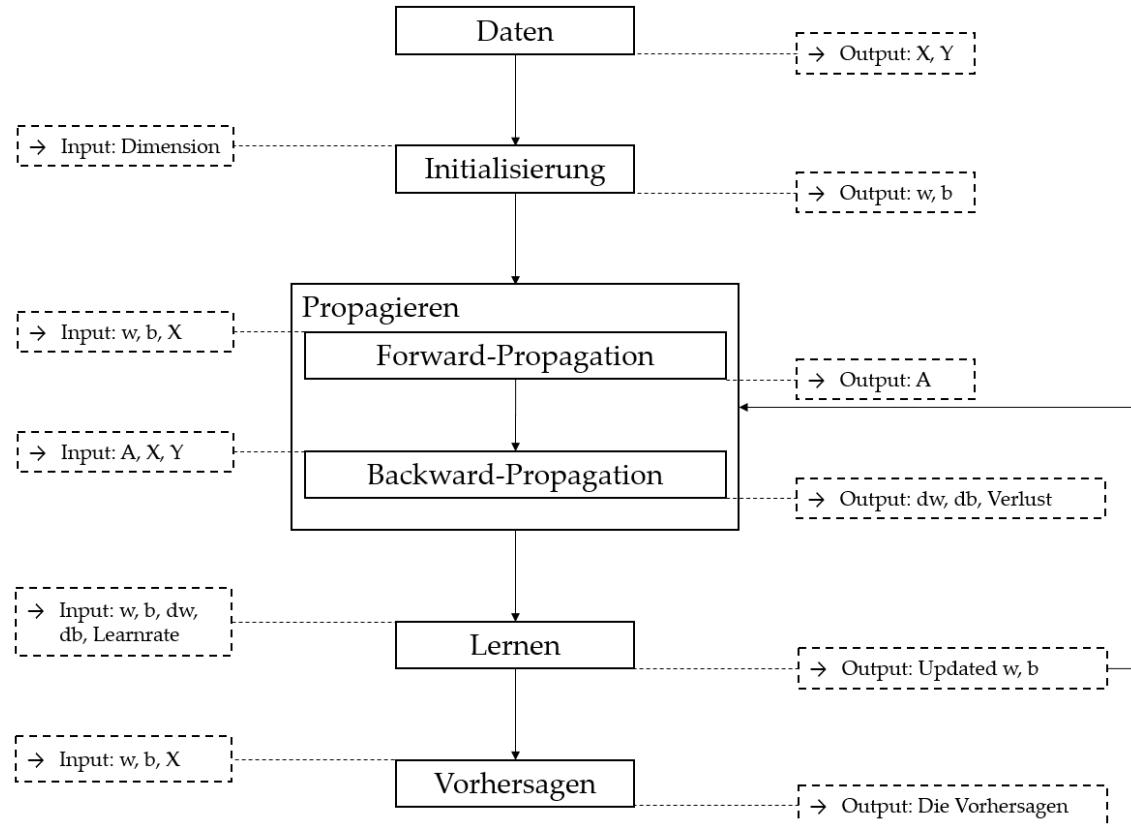


Abbildung 4.5: Schematischer Ablauf eines Lern-Algorithmus

Hinweis), der die Größe der Lernschritte beeinflusst. Anders gesagt, wie groß dürfen die Anpassungsschritte der Parameter w und b in jeder Lernphase sein. Die Lernrate sollte nicht zu klein und nicht zu groß gewählt werden. Abbildung 4.6 zeigt eine Lernkurve bei der das optimalste Modell bei $y = 0$ liegt. Der Algorithmus beginnt mit einem schlechten Modell und verbessert dieses von Lernphase zu Lernphase, auch oft als Iterationen bezeichnet. Angenommen wir wählen die Lernrate zu klein und limitieren die Anzahl der Epochen, dann besteht die Möglichkeit, dass die Lernrate nicht das optimale Modell erreicht, da der Algorithmus vorher die Lernphase bei erreichen der maximalen Epochenzahl abbricht. Wird die Lernrate optimal gewählt, konvergiert sie gegen das optimale Modell, siehe Abbildung 4.6 a). Wenn die Lernrate zu groß gewählt wird, besteht die Möglichkeit, dass das Training divergiert 4.6 b).

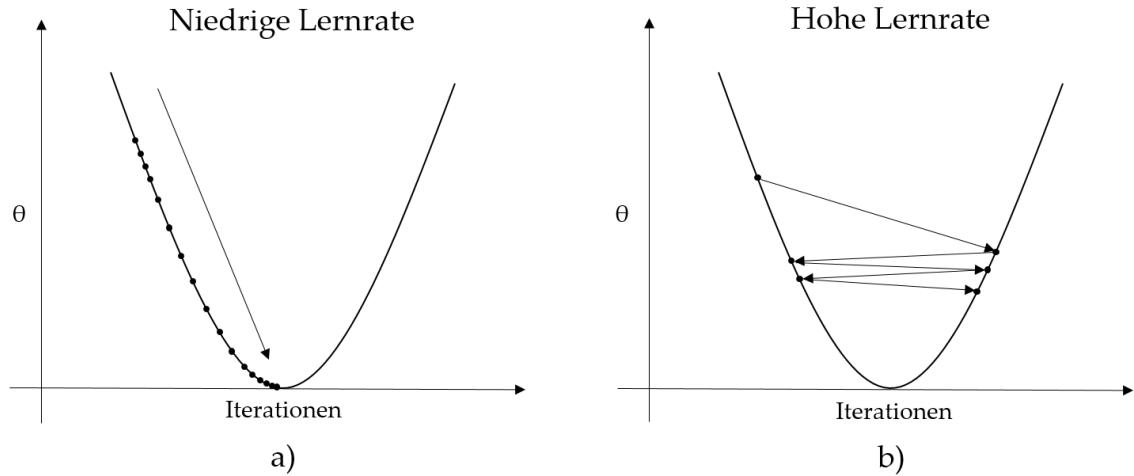


Abbildung 4.6: Auswirkung des Parameters Lernrate auf das Lernverhalten. Die x-Achse spiegelt die Anzahl der Iterationen und die y-Achse die Lernrate wieder. a) Die Lernrate ist gut eingestellt und konvergiert gegen das optimale Modell. b) Die Lernrate ist zu hoch eingestellt und die Gefahr der Divergenz besteht.

Abschließend können mittels dem Modell Prognosen erstellt werden.

Hinweis: Im DL existieren zwei Arten von Parametern. Die *Hyperparameter*, die die Art und Weise der Einstellungen der Parameter festlegen. Die *Parameter* sind in diesem Fall alle, die im Netz angepasst werden können. Beispielsweise legt die Parameterinitialisierung aus dem Abschnitt 4.2.3 (Hyperparameter) fest, wie die Gewichte (Parameter) initialisiert werden. Die Parameter können nochmals in zwei Kategorien eingeteilt werden, in *nicht trainierbare* oder *trainierbare* Parameter. Die trainierbaren Parameter verändern sich während den Training, dahingegen sind die nicht trainierbaren Parameter unveränderbar.

4.2.1 Sprung zu Deep Learning

Um Fehler und Verwechslungen im Bereich der Dimensionen zu vermeiden, vertiefen wir in diesem Abschnitt das Thema in einem erweiterten Neuronalen Netz.

Ein sehr häufiger Fehler, wenn mit DL-Modellen gearbeitet wird, ist das fehlende Verständnis mit welchen Dimensionen das Netz arbeitet. Die Abbildung 4.4 zeigt ein Neuronales Netz mit vier Inputs und einem Hidden Layer. Hierfür definieren wir im Rahmen der Thesis, dass $L = 1$ ist, wobei $l^{[0]}$ die Inputs und $l^{[1]}$ den ersten Hidden Layer widerspiegelt. L enthält die Tiefe bzw. die Anzahl der Layer und Hidden Layer und wird mit l spezifisch adressiert. Die jetzige Definition mittels L sorgt im weiterem Verlauf für eine einfache Übertragung auf die DL-Ebene.

Weiterhin bezeichnen wir die Neuronen mit $l_n^{[L]}$, wobei n für die Anzahl der Neuronen in einem Layer steht. Zurück zum Beispiel der Abbildung 4.4 bedeutet das, dass wir das eine Neuron mit $l_1^{[1]}$ adressieren können.

Abbildung 4.7 repräsentiert ein Neuronales Netz mit $L = 2$ Layern, wobei der erste Hidden Layer $l^{[1]}$ aus drei Neuronen besteht und der Hidden Layer $l^{[2]}$ aus einem Neuron, wobei $a_1^{[2]} = \hat{y}$ gilt.

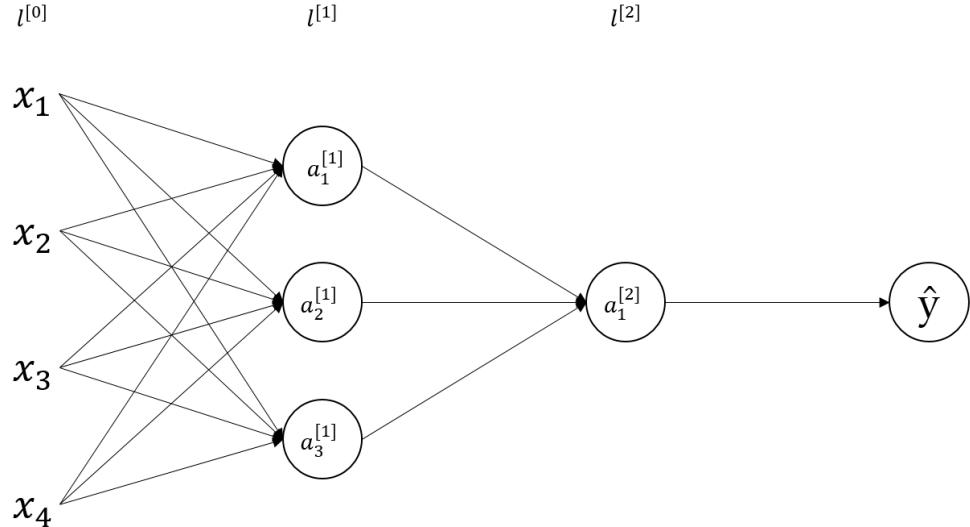


Abbildung 4.7: Das Neuronale Netz besteht aus $L = 2$ Layer. Der Hidden Layer $l^{[1]}$ besteht aus $n_1 = 3$ Neuronen. a_{n_1} bezeichnet das Ergebnis eines Neurons mit der Aktivierungsfunktion.

Wie bereits durch die Formel (4.1) beschrieben, berechnet allgemein ein Neuron die Linearkombination der Gewichte und der Inputs mittels $a = \mathbf{W}^T \mathbf{x} + \mathbf{b}$. Für die Abbildung 4.7 und dem ersten Neuron des Hidden Layers $l_1^{[1]}$ bedeutet dies

$$z_1^{[1]} = \mathbf{W}^{[1]} \mathbf{x} + \mathbf{b}^{[1]} = \begin{pmatrix} g_{1,1} & g_{1,2} & g_{1,3} \\ g_{2,1} & g_{2,2} & g_{2,3} \\ g_{3,1} & g_{3,2} & g_{3,3} \\ g_{4,1} & g_{4,2} & g_{4,3} \end{pmatrix}^T \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}. \quad (4.7)$$

$\mathbf{W}^{[1]}$ enthält die Gewichtungen der einzelnen Linearkombinationen, genauer im Kapitel der Parameterinitialisierung 4.2.3. Die Formel (4.7) beschreibt das Skalarprodukt der transponierten Matrix $\mathbf{W}^{[1]}$ mit dem Input \mathbf{x} . Danach wird der Bias addiert. Um $l_1^{[1]}$ in einem nicht-linearen Bereich zu transformieren, wird eine Aktivierungsfunktion aus dem Abschnitt 4.2.2 $a_1^{[1]} = \sigma(z_1^{[1]})$ angewendet. Dadurch ergibt sich der Ergebnisvektor (4.8) des ersten Hidden Layers $l^{[1]}$.

$$\mathbf{a}^{[1]} = \begin{pmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{pmatrix} = \sigma(\mathbf{z}^{[1]}) = \begin{pmatrix} \sigma(z_1^{[1]}) \\ \sigma(z_2^{[1]}) \\ \sigma(z_3^{[1]}) \end{pmatrix} \quad (4.8)$$

Nachdem der Hidden Layer $\mathbf{l}^{[1]}$ ausgerechnet wurde, erhalten wir unseren Vektor $\mathbf{a}^{[1]}$ im Raum $\mathbb{R}^{3 \times 1}$. Der nächste Schritt berechnet den Hidden Layer $a^{[2]}$ mit den Ergebnissen aus $a^{[1]}$. Bevor wir diese Berechnung durchführen, überlegen wir uns vorab, welche Dimension das Ergebnis unseres zweiten Hidden Layers haben wird. Der zweite Hidden Layer besteht lediglich aus einem Neuron $a_1^{[2]}$ und damit ergibt sich für das Ergebnis \mathbb{R} . Die Berechnung des zweiten Hidden Layers beschreibt die Formel (4.9).

$$\hat{y} = a^{[2]} = \sigma(z^{[2]}) = \sigma(\mathbf{w}^{[2]} \mathbf{x} + b^{[2]}) = \sigma\left(\begin{pmatrix} g_1 \\ g_2 \\ g_3 \end{pmatrix}^T \begin{pmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{pmatrix} + b\right) \quad (4.9)$$

Die Berechnung des zweiten Hidden Layers ähnelt im Prinzip den Berechnungen des ersten Hidden Layers. Der erste Hidden Layer verwendete die originalen eingehenden Daten, wohingegen der zweite Hidden Layer die Ergebnisse des ersten Hidden Layers verwendet.

Im Gegensatz zu dem eben betrachteten Neuronalen Netz diskutieren wir nun DL-Netze. DL-Netze bestehen aus mehreren Hidden Layer zwischen der Eingabeschicht und Ausgabeschicht, siehe Abbildung 4.8 der linke und rechte Teil. Zu beachten gilt, dass die Schwelle von Neuronalen Netzen und DL nicht eindeutig definiert ist. In dieser Thesis reden wir von der Eingabeschicht, wenn wir *Input Layer*, und von der Ausgabeschicht, wenn wir den *Output Layer*, meinen.

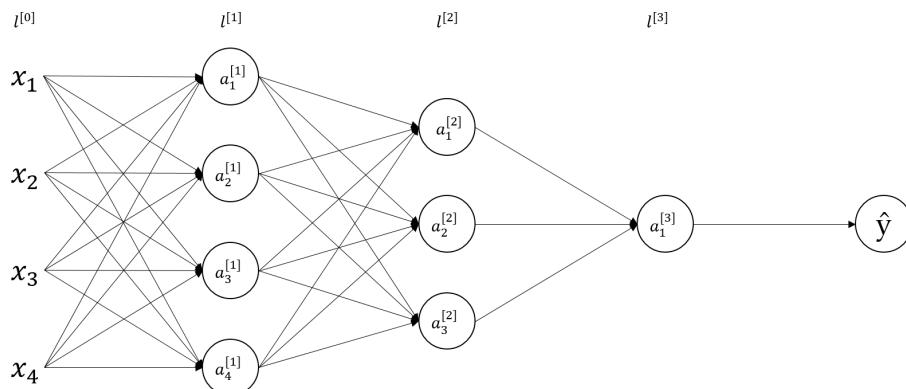


Abbildung 4.8: Diese Abbildung zeigt ein Neuronales Netz mit zwei Hidden Layer. Unser erster Hidden Layer $\mathbf{l}^{[1]}$ besteht aus vier Neuronen $n_1 = 4$ und der zweite Hidden Layer $\mathbf{l}^{[2]}$ enthält drei Neuronen $n_2 = 3$.

4.2.2 Aktivierungsfunktion

Fällt die Entscheidung in einem Projekt auf ein Neuronales Netz, ist ein wichtiges Kriterium die Aktivierungsfunktion, die der Aktivierung eines Neurons im menschlichen Gehirn ähnelt. Diese Funktion liegt in einem künstlichen Neuronalen Netz zwischen den Berechnungen des Modells und dem Output, wie in der Abbildung 4.4 als σ illustriert wurde [Ng15, Folie C1W3L06].

An dieser Stelle stellt sich die Frage, wozu die Aktivierungsfunktionen überhaupt benötigt werden? Angenommen anstatt einer gängigen Funktion, wie Sigmoid oder Tanh, verwenden wir eine lineare Funktion für alle Neuronen, also die Identität, , dann führt dies dazu, dass das Modell nur linear (kombinierte) Ergebnisse erzeugen kann [Ng15, Folie C1W3L07].

Die Linearität wird trotzdem häufig für das letzte Neuron vor dem Output verwendet, um Intervalle auszugeben. Beispielsweise kann dies der Kaufpreis eines Hauses oder die Leistung eines Autos sein. Da diese Beispiele nur positive Werte liefern, könnte der Output mit einer ReLU- oder Leaky ReLU-Funktion aktiviert werden, mehr in der Tabelle 4.1.

Tabelle 4.1 zeigt vier gängige Aktivierungsfunktionen. Die Formeln der Funktionen werden für die Forward-Propagation verwendet. Für die Back-Propagation wird deren Ableitung benutzt. Wie wir feststellen, besteht die Notwendigkeit nicht-lineare Aktivierungsfunktionen zu verwenden, da der Output sonst nur lineare Werte kombiniert. Beispielsweise können diese Aufgabe die Sigmoid- und Tanh-Aktivierungsfunktion übernehmen. Nach Abhängigkeit des Datensatzes sind die Aktivierungsfunktionen unterschiedlich gut geeignet, um die Abhängigkeit der Datensätze nachzubilden. Nach [Ng15] hat sich herausgestellt, dass in vielen Fällen die Tanh- gegenüber der Sigmoid-Funktion besser abschneidet.

Tabelle 4.1: Übersicht gängiger Aktivierungsfunktionen.

Aktivierungsfunktion	Formel	Ableitung
Sigmoid	$\frac{1}{1+e^{-z}}$	$1 - \sigma(z)$
Tanh	$tanh(z)$	$1 - (tanh(z))^2$
ReLU	$max(0,z)$	$\begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$
Leaky ReLU	$max(0,0.01z,z)$	$\begin{cases} 0,01z & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$

Das Thema Aktivierungsfunktion stellt ein aktuelles Forschungsgebiet dar. Drei Wissenschaftler vom Google Brain Team veröffentlichten 2017 eine Möglichkeit [Ram17], die eine Suche nach neuen Aktivierungsfunktionen ermöglicht. Neben einigen neuen Aktivierungsfunktionen, finden sie die Aktivierungsfunktion „Swish“, die sich aus $f(x) = x * sigmoid(\beta x)$ definiert. Die Wissenschaftler vergleichen Swish mit der gängigen ReLU-Funktion und erhalten als Ergebnis eine deutliche Tendenz einer besseren Klassifikationsgüte.

4.2.3 Parameterinitialisierung

Im Abschnitt 4.2 haben wir die Initialisierung der Parameter für \mathbf{W} und \mathbf{b} nicht näher spezifiziert. In diesem Kapitel betrachten wir drei unterschiedliche Verfahren, mit denen eine Parameterinitialisierung durchgeführt werden kann [Ng15, Folie C2W1L11].

Die erste Variante ist die Initialisierung mit Nullen.

$$\mathbf{W}^{[l]} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & 0 \end{pmatrix}, \mathbf{b}^{[l]} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}, \quad (4.10)$$

wobei l den Layer widerspiegelt. Angenommen unser Neuronales Netz verwendet als Aktivierungsfunktion ReLU aus dem Abschnitt 4.2.2 und wir wenden diese auf unsere Null-Matrix (4.10) an. Dabei stellt sich heraus, dass jedes Neuron die gleichen Berechnungen durchführt, wie schon im vorigen Abschnitt 4.2.2 beschrieben. Werden die Berechnungen vorwärts propagiert, wird jedes Neuron dieselben Ergebnisse liefern, da die Input Layer an jedes Neuron die selben Daten einfließen lassen. Werden die Berechnungen zurück propagiert, wird jeder Gradient ebenfalls mit den gleichen Werten berechnet. Dadurch, dass alle Neuronen symmetrisch sind, stellt sich heraus, dass die diese Art von Parameterinitialisierung das Verhalten eines einzelnen Neurons widerspiegelt [Ng15, Folie C2W1L11].

Eine bessere Variante ist die zufällige Initialisierung der Parameter. Dies geschieht, indem ein Zufallsgenerator Zufallszahlen basierend auf der Standardnormalverteilung generiert und diese mit 10 multipliziert, damit die Gewichte nicht zu klein sind, da sonst numerische Probleme auftreten können. Mit

$$\mathbf{W}^{[l]} = \begin{pmatrix} \mathcal{N}(\mu, \sigma^2) * 10 & \mathcal{N}(\mu, \sigma^2) * 10 & \cdots & \mathcal{N}(\mu, \sigma^2) * 10 \\ \vdots & \vdots & \ddots & \mathcal{N}(\mu, \sigma^2) * 10 \\ \mathcal{N}(\mu, \sigma^2) * 10 & \mathcal{N}(\mu, \sigma^2) * 10 & \cdots & \mathcal{N}(\mu, \sigma^2) * 10 \end{pmatrix} \quad (4.11)$$

und

$$\mathbf{b}^{[l]} = \begin{pmatrix} \mathcal{N}(\mu, \sigma^2) * 10 \\ \vdots \\ \mathcal{N}(\mu, \sigma^2) * 10 \end{pmatrix} \quad (4.12)$$

initialisieren wir unsere Gewichte und laut Andrew [Ng15, Folie C2W1L11] hat sich herausgestellt, dass diese Art der Initialisierung bessere Ergebnisse liefert. Der Grund der Besserung sind die verschiedenen Gewichtungen von \mathbf{W} und \mathbf{b} , denn jetzt unterscheiden sich die Berechnungen der Neuronen. Propagiert man vorwärts, wird jedes Neuron mit unterschiedlichen Gewichtungen bestückt. Bei der Back-Propagation entsteht derselbe Effekt, das dem Netzwerk erlaubt die unterschiedlichen Gewichtungen den Daten anzupassen.

Nach Andrew [Ng15, Folie C2W1L11] soll die *HE Initialisierung* bessere und schnellere Ergebnisse liefern. Dazu müssen wir in den Formeln (4.11) und (4.12) den Mutliplikator mit $\sqrt{\frac{2}{\dim(l-1)}}$ ersetzen, wobei $\dim(l-1)$ für die Dimension des vorigen Layers steht. Somit ergeben sich die Initialisierungen (4.13) und (4.14).

$$\mathbf{W}^{[l]} = \begin{pmatrix} \mathcal{N}(\mu, \sigma^2) * \sqrt{\frac{2}{\dim(l-1)}} & \mathcal{N}(\mu, \sigma^2) * \sqrt{\frac{2}{\dim(l-1)}} & \cdots & \mathcal{N}(\mu, \sigma^2) * \sqrt{\frac{2}{\dim(l-1)}} \\ \vdots & \vdots & \ddots & \mathcal{N}(\mu, \sigma^2) * \sqrt{\frac{2}{\dim(l-1)}} \\ \mathcal{N}(\mu, \sigma^2) * \sqrt{\frac{2}{\dim(l-1)}} & \mathcal{N}(\mu, \sigma^2) * \sqrt{\frac{2}{\dim(l-1)}} & \cdots & \mathcal{N}(\mu, \sigma^2) * \sqrt{\frac{2}{\dim(l-1)}} \end{pmatrix} \quad (4.13)$$

$$\mathbf{b}^{[l]} = \begin{pmatrix} \mathcal{N}(\mu, \sigma^2) * \sqrt{\frac{2}{\dim(l-1)}} \\ \vdots \\ \mathcal{N}(\mu, \sigma^2) * \sqrt{\frac{2}{\dim(l-1)}} \end{pmatrix} \quad (4.14)$$

Diese Initialisierung (4.13) und (4.14) wird auch *Xavier Initialisierung* genannt und wurde von den Autoren Xavier Glorot et al. [Glo10] entwickelt.

Für detailliertere Informationen evaluiert Kumar [Kum17] die Auswirkungen der Parameterinitialisierung für Neuronale Netze mit verschiedenen Aktivierungsfunktionen.

4.2.4 Kostenfunktion

Ein weiterer wichtiger Aspekt ist die Bewertung der Klassifikation. Ein Ziel des Neuronalen Netzes ist der Versuch einem gegebenen Datensatz $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$, wobei m Anzahl der Daten widerspiegelt, $\hat{y}^{(1)} \approx y^{(1)}$ anzunähern. Dieser Unterschied wird Verlust genannt.

Neben vielen Verlustfunktionen stellen die *Cross-Entropy-Verlustfunktion* [Ng15, Folie C1W2L03] vor, die gegeben ist durch:

$$L(\hat{y}, y)_{CE} = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})). \quad (4.15)$$

Die Kostenfunktion (4.16) definieren wir als Verlust über die gesamten Daten.

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m L_{CE}(\hat{y}^{(i)}, y^{(i)}) \quad (4.16)$$

Rosasco et al. [Ros03] verglichen 2003 den Einfluss unterschiedlicher Verlustfunktionen. Dabei stellte sich heraus, dass gerade bei Klassifizierungsproblemen die *hinge*-Verlustfunktion die beste sei [Ros03, S. 14]. Weitere Verlustfunktionen sind die quadratische- und die logistische Verlustfunktion, die getestet wurden.

4.3 Regularisierungen

Nachdem wir unser erstes Neuronales Netz kennengelernt haben, schauen wir uns als nächstes Regularisierungstechniken an. Im Abschnitt 4.1.4 wurde der Bias-Variance-Tradeoff besprochen, auf den wir jetzt zurückgreifen werden. In DL-Projekten besteht oft das Problem, dass der Trainingsdatensatz overfitted wird, demnach entsteht eine hohe Varianz und das Modell schneidet schlecht auf dem Testdatensatz mit einem hohen Bias ab. Die Frobinius Norm aus dem Abschnitt 4.3.1 und das Dropout-Verfahren aus dem Abschnitt 4.3.2 helfen den Overfit in unserem Modell zu regulieren und somit den Bias und die Varianz anzupassen. Weiterhin entwickelten Yao et al. im Jahr 2007 [Yao07] eine Regel mit dem Namen Early Stopping, die die Trainingsphase bei einem möglichen Optimum beendet.

4.3.1 Frobiniusnorm

Die Frobiniusnorm ist definiert durch

$$\|\mathbf{W}^{[l]}\|_F^2 := \sum_{i=1}^m \sum_{j=1}^n |w_{ij}^{[l]}|^2, \quad (4.17)$$

wobei $\mathbf{W}^{[l]}$ die Gewichtungsmatrix von Layer l ist und F die Frobiniusnorm kennzeichnet. Wie in der Ridge- oder Lasso-Regression werden wir die Frobiniusnorm als sogenannten Strafterm verwenden. Im Abschnitt 4.2.4 lernen wir die Kostenfunktionen kennen und addieren die Frobiniusnorm, sodass die Formel (4.18) entsteht [Ng15].

$$\underbrace{\frac{1}{m} \sum_{i=1}^m L(\hat{y}^{[i]}, y^{[i]})}_{\text{Kostenfunktion}} + \underbrace{\frac{\lambda}{2m} \sum_{i=1}^l \|\mathbf{W}^{[i]}\|_F^2}_{\text{Strafterm}} \quad (4.18)$$

stellt, wie im Abschnitt der Kostenfunktion 4.2.4 bereits erwähnt, mit dem Tuning Parameter λ ein Minimierungsproblem dar. In diesem Fall müssen zwei Terme minimiert werden, die Kostenfunktion und die Straffunktion. Wählt man λ groß, wird der rechte Term groß, weswegen die Kostenfunktion klein wird. Wählt man λ gegen Null, entfällt der Strafterm.

4.3.2 Dropout

Ein weiterer Ansatz gegen Overfitting ist Dropout [Ng15, Folie C2W1L07]. Hierbei werden zufällige Neuronen in einem Netzwerk ausgeschaltet. Die Anzahl der zufälligen Neuronen, die ausgeschaltet werden sollen, wird in Prozent angegeben und liegt meistens in einem Bereich von 0-50 %.

In der Beispielabbildung 4.8 könnte $l^{[1]}$ mit einer Droprate von 0,5 und $l^{[2]}$ mit 0,33 bestückt werden. Dadurch würden im Layer $l^{[1]}$ zwei Neuronen und im Layer $l^{[2]}$ ein Neuron entfallen.

4.4 Optimierungen

Ein tiefes Neuronales Netz, das aus vielen Layern besteht, kann im schlimmsten Fall extrem groß und inperfomant in Bezug auf die Berechnungszeiten werden.

4.4.1 Mini-Batch

DL-Netze sind meistens so konzipiert, dass eine große Menge an Trainingsdaten benötigt wird und oft große Netze im Einsatz sind. Dadurch können Arbeitsspeicherprobleme entstehen. Weiterhin dauern alle Berechnungen mit allen Trainingsdatensätzen des gesamten Netzes sehr lang. Eine Lösung ist Mini-Batch [Ng15, Folie C2W2L01]. Die Idee ist, den Trainingsdatensatz in kleinere Teile aufzuteilen, sodass der Arbeitsspeicher nicht alle Daten aus dem Trainingsdatensatz halten muss. Wir definieren hierzu

$$\mathbf{x} = [\underbrace{x^{(1)}, \dots, x^{(s)}}_{\mathbf{x}^{\{1\}}}, \dots, \underbrace{x^{(m-s-1)}, \dots, x^{(m)}}_{\mathbf{x}^{\{\frac{s}{m}\}}}] \quad (4.19)$$

und

$$\mathbf{y} = [\underbrace{y^{(1)}, \dots, y^{(s)}}_{\mathbf{y}^{\{1\}}}, \dots, \underbrace{y^{(m-s-1)}, \dots, y^{(m)}}_{\mathbf{y}^{\{\frac{s}{m}\}}}] \quad (4.20)$$

für die Unterteilung des Testdatensatzes in Mini-Batches, wobei s für die Größe der Mini-Batches und m für die Größen des Testdatensatzes steht. Mit hochgestellten geschweiften Klammern beschreiben wir die Mini-Batches $\mathbf{x}^{\{t\}}$ und $\mathbf{y}^{\{t\}}$, wobei t für die Position des Mini-Batches steht. Die folgenden zwei Extremfälle sind möglich:

- $s = 1$ bedeutet $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)}) = (x^{\{1\}}, y^{\{1\}}), \dots, (x^{\{m\}}, y^{\{m\}})$, wobei jeder einzelne Datensatz durch das Netzwerk propagiert wird. Diese Einstellung wird stochastischer Gradientenabstieg, engl. stochastic gradient descent, genannt.
- $s = m$ bedeutet $(\mathbf{x}^{\{1\}}, \mathbf{y}^{\{1\}}) = (\mathbf{x}, \mathbf{y})$, wodurch der gesamte Datensatz auf einen Schlag propagiert wird. Diese Einstellung wird *Batch Gradient Descent* genannt.

Nach Andrew Ng [Ng15] liegt die Wahrheit zwischen $1 < s < m$. Je nach dem, welche Hardware zur Verfügung steht, kann es von Vorteil sein zweier Potenzen als Mini-Batch-Größe zu verwenden. Weiterhin existieren zur Zeit keine wissenschaftlichen Vorgehensweisen zur Auswahl des Hyperparameters, da die gängige Hardware zu unterschiedlich ist.

4.4.2 Weitere Optimierungsverfahren

Die folgende Liste enthält weitere gängige Optimierungsverfahren.

- *Gradient descent with momentum* [Ng15, Folie C2W2L06] setzt *Exponentially weighted moving averages* [Ng15, Folie C2W2L03] ein, um den Lernprozess zu glätten.
- *Adam* [Ng15, Folie C2W2L08] kombiniert *RMSProp* [Ng15, Folie C2W2L07] mit *Gradient descent with momentum* und kann nach Andrew NG das Optimum eines Modells schneller finden.

4.5 Convolutional Neural Network

Schichten wie KNN aus dem Abschnitt 4.2 enthalten in der Regel eine hohe Dichte an Verknüpfungen/Gewichten, wenn Regularisierungen wie Dropout ausgeschlossen werden [Li18, Folie 5-61]. Weiterhin haben diese Art von Schichten den Beigeschmack, dass in den meisten Fällen die gesamten Neuronen oder Daten betrachtet werden, siehe Abbildung 4.7. Angenommen wir erhalten als eingehende Daten Bilder mit einer Pixelgröße von 4x4 und verwenden ein Neuronales Netz mit vier Neuronen aus dem Abschnitt 4.2.1. In dieser Situation dient jeder Pixel als Input. In diesem Beispiel existieren bereits $4 * 4 * 4 = 64$ unterschiedliche Parameter im ersten Layer. Vergrößern wie das Bild, steigt die Anzahl der Parameter extrem. Ein Vorteil der Convolutional Neuronalen Netze ist, dass die Anzahl der Parameter weniger schnell steigt, als in einem Neuronalen Netz [Li18, Folie 5-61].

Convolutional Neural Networks, dt. Faltendes Neuronales Netzwerk und CNN abgekürzt, wurde von Matsugu et al. im Jahr 2013 [Mat03] erforscht. Dabei stellt sich heraus, dass sich CNNs besonders gut für die Verarbeitung von natürlicher Sprache eignet [Kim14, S. 1746].

Ein CNN nimmt als Input Bildformate oder ähnlich umgewandelte Formate entgegen, die mittels eines Kernels weiterverarbeitet werden. Der *Kernel* ist wie ein Fenster zu interpretieren, das lokal über die Daten fährt. Er weist Ähnlichkeiten mit den Filtern der Bildverarbeitung auf, siehe Abbildung 4.9. Einer der Nebeneffekte kann die Reduktion der Komplexität sein, in unserem Beispiel um die Hälfte der Pixel. Dies kann bei hochauflösenden Bildern oder vielen Daten das Netz beschleunigen [Li18, S. 5–26].

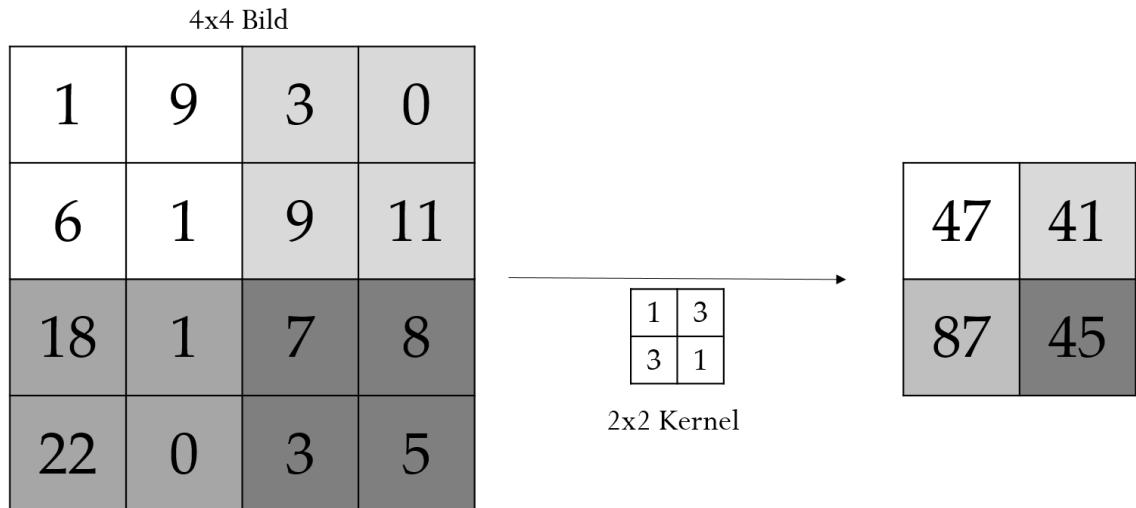


Abbildung 4.9: Einfaches CNN Beispiel (Pooling). Auf der linken Seite erhält das CNN ein 4x4 Bild. Die Graustufen spiegeln die Fenster wieder, indem der 2x2 Kernel Berechnungen durchführt. Der Kernel iteriert über das Bild in zweier Schritten, stride = 2, von links nach rechts und von oben nach unten. Als Ergebnis erhalten wir wiederum eine 2x2 Matrix. Die Berechnung für die weiße Fläche könnte wie folgt aussehen: $1 * 1 + 9 * 3 + 6 * 3 + 1 * 1$.

Das CNN bietet verschiedene Hyperparameter, wie die *Größe des Kernels*. Gängige Größen nach [Li18, Folie 5-61] sind 2x2, 3x3, 5x5 und 7x7. Ein weiterer Hyperparameter ist die *Schrittweite*, engl. stride, die angibt in welchen Sprüngen der Kernel über die Daten iteriert. In Abbildung 4.9 ist stride = 2 und eine Kernelgröße von 2x2 gewählt worden. Beispielsweise bei einem Kernel der Größe 3x3 mit einem stride = 3 würde ein Fehler auftreten, da die von CNN erwartete Dateninformationen nicht mit den tatsächlichen Datenformat übereinstimmt. Eine weitere Einstellungsmöglichkeit ist der *Rand*, engl. padding. Wird ein Rand zugelassen, z. B. padding = 1, spannt man um das Bild links, rechts, oben und unten zusätzliche Zeilen mit Nullen. Deswegen kann der Kernel links oben bei der 1 beginnen und enthält drei Nullen. Das Ergebnis ist eine 3x3-Matrix. Dieses Verfahren wird auch Pooling genannt.

Weiterhin können mehrere Kernels verwendet werden, wodurch sich das Resultat aufschichtet. Fügen wir der Abbildung noch vier weitere Kernels hinzu, erhalten wir ein Ergebnis im Format 2x2x5. Vorteilhaft an dieser Methodik ist, dass für jeden Kernel nach unterschiedlichen lokalen Strukturen gesucht werden kann. Je mehr CNNs hintereinander geschaltet sind, also je tiefer das Netzwerk wird, desto komplexere lokale Strukturen können erkannt werden. Der aktuelle Trend geht in Richtung eines sehr tiefen CNN mit kleinen Kernels, wie 2x2 oder 3x3, und einer dicht besiedelten Schicht vor dem Output [Li18, Folie 5-78].

CNN nahm in der jüngeren Vergangenheit stark an Popularität zu und unterschiedliche Forschungsteams arbeiten heute an der Verbesserung der Kombination von verschiedenen CNN-Architekturen. Stark motiviert sind die Teams durch den Wettbewerb *Large Scale Visual Recognition Challenge (LSVRC)* [Rus15], der seit 2010 jährlich ausgetragen wird. Bei dieser Herausforderung werden verschiedene Bilder mit Objekten bereitgestellt, welche die Teams versuchen müssen zu klassifizieren und somit Objekte zu erkennen. Diese Klassifikation und Objekterkennung mittels unterschiedlichen CNN-Architekturen funktioniert besonders gut [Li18, S. 9].

AlexNet [Kri12], VGG [Wan15], GoogLeNet [Sze15] und ResNet [He15] sind die Architekturen, die besonders gut bei LSVRC abgeschnitten haben. Auffallend ist bei AlexNet, dass diese eine Fehlklassifikationsrate von 16,4 % mit 8 Layer erreicht haben, wohingegen der Gewinner ResNet 152 Layer benötigte [Li18, S. 9–22]. Die Autoren betonen aber ausdrücklich, dass die Genauigkeit nicht unbedingt steigt, wenn das DL-Netz immer tiefer strukturiert wird. Beispielsweise erreichte GoogLeNet eine Fehlklassifikationsrate von 6,7 % mit lediglich 22 Layern.

Abbildung 4.9 stellt ein Beispiel für Pooling dar, welches die x- und y-Achse verkleinert. Dadurch hat das Netz ausschließlich die Chance kleiner zu werden. Jedoch gibt es Anwendungen, wie die Objekterkennung, die genau das Gegenteil verlangen. Hierbei wird versucht pixelgenau Bereiche zu finden, die einem Objekt zugeordnet werden. Eine Idee ist Unpooling [Li18, S. 11–26], das ein umgekehrtes Verfahren von Pooling darstellt.

4.6 Recurrent Neural Network

In den vorigen Abschnitten KNN 4.2 und CNN 4.5 lernen wir zwei unterschiedliche Layer kennen. KNN besteht aus Neuronen, die im normalen Fall voll miteinander verbunden sind. CNN verwendet Kernel, die die Lokalitäten ausnutzen. Beide Verfahren ignorieren größtenteils die zeitliche Komponente der Inputs. Dies wird besonders interessant, sollten die Daten (Sequenzen) in irgendeiner Art und Weise voneinander abhängig sein. Dieses Kapitel führt das Recurrent Neural Network (RNN) ein.

RNNs nutzen sogenannte self-connected Hidden Layer. Diese Art von Layer ermöglicht ein Gedächtnis, welches auf vorige Informationen der Daten (Sequenz) zugreifen kann. Abbildung 4.10 zeigt das handgeschriebene Wort *defence*. Das ganze Wort ist einfach zu erkennen, wird jedoch der einzelne Buchstabe n betrachtet, fällt dies durch den fehlenden Kontext schwerer. Diese Idee nehmen die RNNs auf und versuchen den Kontext zu einzubinden [Gra08, S. 5].

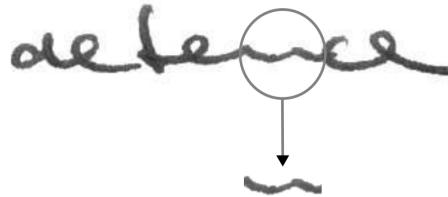


Abbildung 4.10: Handgeschriebenes Beispiel [Gra08, S. 5] (Long Short-Term Memory). Illustriert das Problem einer Sequenz anhand des Buchstabens n .

Besonders gut stellt sich RNN in dem Bereichen handgeschriebene Texte [Gra08], Spracherkennung [Sak14], Zeitreihenanalyse [Oed06], [Gil01] u. v. m. heraus.

Long Short-Term Memory

Ein RNN-Layer, der sich als besonders gut herausstellt, ist Long Short-Term Memory (LSTM), der von Hochreiter et al. [Hoc97] 1997 erforscht wurde. Rund 10 Jahre nach seiner Entdeckung erfährt dieses Verfahren seinen Durchbruch im Bereich Spracherkennung [Fer07].

Eine LSTM-Zelle nimmt als Input den vorigen Zellstatus c_{t-1} , den vorigen versteckten Zellstatus h_{t-1} und den Teil der Sequenz x_t entgegen. h_{t-1} und x_t werden als Vektor so gestapelt, dass die Gleichung (4.21) [Li18, S. 10–98] entsteht, wobei \mathbf{W} die Gewichtsmatrix repräsentiert. Abbildung 4.11 zeigt den kompletten schematischen Ablauf dieser Zelle.

$$\begin{pmatrix} f \\ i \\ g \\ o \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \mathbf{W} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \quad (4.21)$$

Die Gleichung (4.21) berechnet 4 unterschiedliche Gates, wobei σ in diesen Fall eine Aktivierungsfunktion, oft Sigmoid, repräsentiert. Diese Gates besitzen die Möglichkeit den Informationsfluss zu lenken, sodass Informationen vergessen oder hinzugefügt werden können. f repräsentiert das *Forget Gate*, wie viel von der vorigen Zelle vergessen werden soll, i das *Input Gate*, wie viel als Input zugelassen werden soll, o das *Output Gate*, wie viel der Außenwelt mitgeteilt werden soll und g für *Gate Gate*, wie viel in die folgende Input Cell geschrieben werden soll [Li18, Folie 10-98]. Die Formeln (4.22) und (4.23) repräsentieren die Outputs der Zelle, wobei \odot für elementweise Multiplikation steht.

$$c_t = f \odot c_{t-1} + i \odot g \quad (4.22)$$

$$h_t = o \odot \tanh(c_t) \quad (4.23)$$

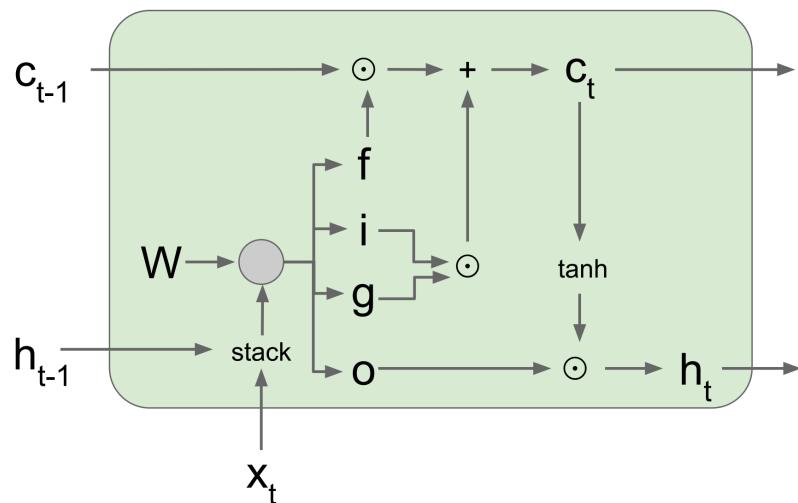


Abbildung 4.11: Die Abbildung zeigt den Aufbau einer LSTM-Zelle [Li18, Folie 10-98]. Von links nach rechts werden die Daten propagiert. c_{t-1} und c_t stellen den vorigen und jetzigen Zellstatus dar. h_{t-1} und h_t repräsentieren den vorigen und jetzigen versteckten Zellstatus. x_t ist unser Datenpunkt, W die Gewichtungsmatrix und mit f (Forget), i (Input), g (Gate) und o (Output) repräsentieren wir die Gates.

Liu et al. [Liu16] präsentieren eine Methode, die Textklassifizierung durch RNNs realisiert. Zuerst wenden sie eine rekurrente Struktur an, um kontextuelle Informationen zu extrahieren. Danach verwenden sie Max-Pooling (CNN), um die Wörter zu erhalten, die die wichtigste Rolle für die Klassifikation spielen.

4.7 Ergebnisse

In diesem Teil diskutieren wir die Ergebnisse der Abschnitte HDLTex [4.7.1](#) und CNN-SC [4.7.2](#), die jeweils unterschiedliche Architekturen aus den Abschnitten KNN [4.2](#) und CNN [4.5](#) nutzen. Beginnend stellen wir das Originalmodell der Paper [\[Kow17\]](#) und [\[Kim14\]](#) vor, dieses wenden wir an und ziehen Rückschlüsse. Anschließend verbessern wir diese Modelle durch Variation der Hyperparameter. Schlussendlich vergleichen wir die Ergebnisse.

Hinweis 1: In einem Neuronalen Netz wird die Parameterinitialisierung aus Abschnitt [4.2.3](#) zufällig gewählt, weswegen wir durch Zufall ein sehr gutes oder schlechtes Modell finden können. Deswegen entscheiden wir uns 1.000 unterschiedliche initialisierte Modelle zu generieren, um eine Stabilität zu gewährleisten. Ein weiterer Grund ist der im Verhältnis kleine Trainingsdatensatz aus dem Abschnitt [3.2.1](#). Im Gegensatz zu den Vergleichsverfahren werden wir in diesem Abschnitt die Ergebnisse von zehn zufällig ausgewählten Modellen mitteln und Konfidenzintervalle bilden.

Hinweis 2: Wir teilen unseren Trainingsdatensatz in die Klassen -1, 0 und 1 ein. Während wir die Modelle, sowohl die Vergleichs- und Deep Learning-Verfahren, trainieren, bemerken wir, dass die Klasse 0 die Ergebnisse nach unserer Sicht nicht beeinflussen, weswegen wir diese nachfolgend nicht beachten. Einen Vorschlag für das weitere Vorgehen ist im Kapitel Ausblick [6](#) zu finden.

Hinweis 3: Die Modelle werden mittels Keras [\[Cho15\]](#), ein Paket aus Python, umgesetzt. Alle Anforderungen sind im Anhang [A.2](#) zu finden.

4.7.1 HDLTex

Zu Beginn nehmen wir Bezug auf ein etabliertes KNN, siehe Abschnitt [4.2.1](#). Danach implementieren und modifizieren wir das Netz, sodass wir eine bessere und stabilere Klassifikation auf unsere Anwendung erreichen.

Einer der wenigen, meistzitierten und neusten Paper mit Bezug auf Text Mining ist *HDL-Tex: Hierarchical Deep Learning for Text Classification* [\[Kow17\]](#). Kamram et al. erkannten ein Problem in der Dokumentenklassifizierung. Aus ihrer Sicht leidet die Qualität der Klassifikation je mehr Dokumente dem Netz hinzugefügt werden. Sie versuchen diese Herausforderung mit einer Kombination der drei Architekturen KNN, CNN und RNN zu lösen, wovon wir nur das KNN-Modell entnehmen. Im Paper erwähnen sie, dass deren Netze für große Datenmengen entwickelt wurden, weswegen wir bezogen auf unseren Fall Overfitting erwarten. Ihr Datensatz besteht aus 46.985 Dokumenten aus sieben unterschiedlichen Domänen, wie Biologie oder Informatik [\[Kow17, S. 5\]](#). Das Modell besteht aus acht Hidden Layern mit jeweils 1.024 Neuronen, siehe Abschnitt [4.7](#). Bis auf den Dropout aus dem Abschnitt [4.3.2](#), der auf 0,5 gesetzt wurde, und die Batchsize von 128 werden die Standardeinstellungen von Keras verwendet. Keras ist ein Python-Bibliothek, die eine einheitliche Schnittstelle zu unterschiedlichen DL-Frameworks bietet [\[Cho15\]](#). Als Optimierungsfunktion wurde Adam gewählt, siehe Abschnitt [4.4.2](#). Wenden wir das Modell auf unseren Daten an, erhalten wir ca. 8 Millionen einstellbare Parameter, siehe Code im Anhang [D.1](#). Als Input verwenden wir eine One-Hot-Kodierung, siehe Abschnitt [3.2.2](#).

Wie für die Vergleichsverfahren führen wir ebenfalls die selben Datenbereinigungsschritte aus dem Abschnitt 3.2.2 durch. Wir trainieren 1.000 unterschiedliche Modelle zwecks der Stabilität, da die Netze die Gewichtsinitialisierung zufällig wählen, vgl. Abschnitt 4.2.3. Um den Trainingsverlauf messen und nachvollziehen zu können, entscheiden wir uns für die Metrik *Genauigkeit*, hier y-Achse. Die Genauigkeit ist die Anzahl der richtig klassifizierten Datenpunkte durch die Gesamtanzahl [Pow15, S. 39]. Eine weitere äquivalente Möglichkeit ist der Verlust, jedoch harmoniert die Genauigkeit mit unseren späteren Vergleichen wesentlich besser. Die x-Achse spiegelt die Anzahl der Epochen wieder. Die grüne Linie repräsentiert die Trainings- und die blaue die Testmenge. Die gestrichelten Linien repräsentieren symmetrisches 95 % Konfidenzintervall. Im linken Teil der Abbildung 4.12 ist das originale Netz aus dem Paper [Kow17, S. 5] und auf der rechten Seite unser angepasstes Modell zu sehen.

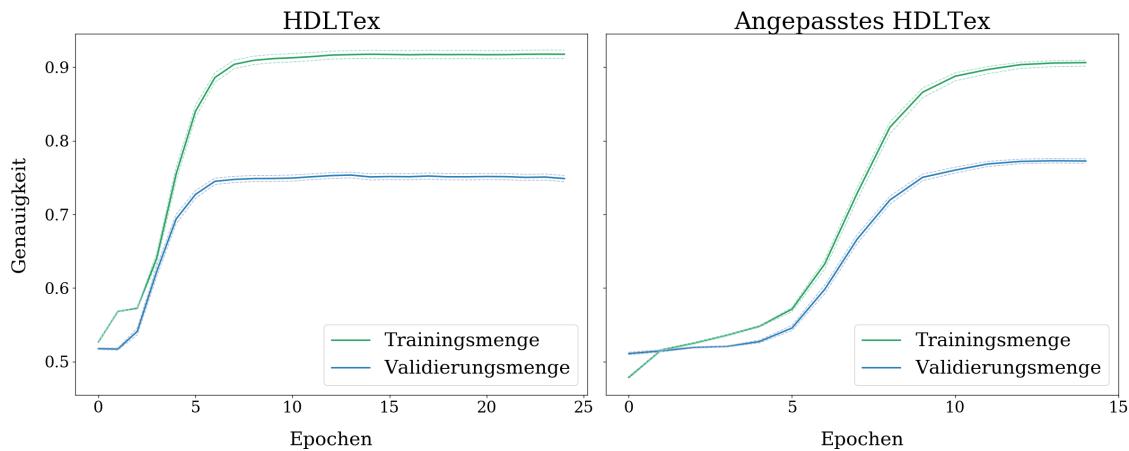


Abbildung 4.12: KNN: Verlauf der Genauigkeit. Der linke Plot repräsentiert den Genauigkeitsverlauf über 25 Epochen des originalen Modells [Kow17, S. 5]. Der rechte Plot zeigt den Verlauf für unser angepasstes Modell.

Durch die Abbildung 4.12 fällt im originalen Modell auf, dass sich nach spätestens 10 Epochen das Netz auf $\approx 90\%$ Genauigkeit der Trainings- und $\approx 75\%$ der Testmenge einpendelt. Weiterhin können wir Overfitting vermuten. Dies kann anhand des großen Unterschiedes der Genauigkeit der Trainings- und Testmenge erahnt werden. Im Prinzip wird das Modell wesentlich besser auf die Trainings- als auf die Testmenge optimiert. Maßnahmen wären Cross-Validation, kleineres Netz oder höhere Restriktionen (z. B. höhere Dropout-Raten). Weiterhin ist die Varianz der unterschiedlichen Modelle auffällig. Je größer diese ist, desto instabiler bzw. mehr Variabilität weisen die Ergebnisse auf. Dies kann in der Größe der Abstände der Konfidenzbänder abgelesen werden. Für eine genauere Betrachtung sind beide Plots in einer größeren Variante im Anhang D.3.1 zu finden.

Wir nehmen uns vor, die Anzahl der Parameter des Modells, das Overfitting und die Varianz zu verringern. Die Anzahl der Parameter verringern wir, indem die Anzahl der Neuronen verringert wird. Dabei beobachten wir, dass das Modell ab einen gewissen Punkt sehr schlecht und instabil wurde. Die Anzahl der Neuronen fängt beim ersten Hidden

Layer bei 128 an und steigt mittels Zweierpotenzen auf 1.024 an. Danach halbieren wir die Anzahl der Neuronen wieder, sodass der letzte Hidden Layer wieder aus 128 Neuronen besteht. Dadurch reduzieren wir die Gesamtzahl der trainierbaren Parameter auf ca. 2,1 Millionen. Außerdem versuchen wir das Overfitting mittels höheren Dropout-Raten von 0,75 und weiteren L2-Restriktionen im Hidden Layer 4 mit 0,01, im Hidden Layer 5 mit 0,015 und im Hidden Layer 6 mit 0,01 zu verhindern, genauer im Code D.2. Als Input verwenden wir eine One-Hot-Kodierung aus dem Abschnitt 3.2.2.

Wenn wir die zwei Plots der Abbildung 4.12 vergleichen, können wir erahnen, dass unser Modell leicht stabiler und genauer in der Klassifizierung ist. Mittels den Boxplots in Abbildung 4.13 erkennen wir ab, dass wir die Varianz des Recalls und den F-Score der Klasse -1 verbessern konnten. Das heißt, wir konnten die Varianz einen relevanten Tweet zu finden, verringern und Wahrscheinlichkeit einen relevanten Tweet zu finden, erhöhen. Alle weiteren Werte bewegen sich auf einem ähnlichen Niveau.

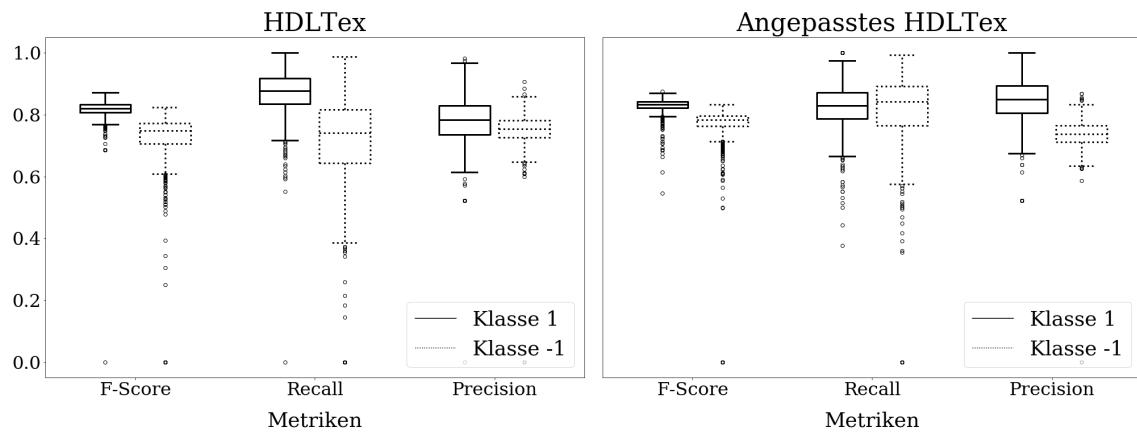


Abbildung 4.13: KNN: Güte. Beide Plots zeigen die Metriken F-Score, Recall und Precision (links das originale HDLTex- und rechts das angepasste HDLTex-Modell).

Betrachten wir die Ergebnisse auf der gesamten Genauigkeit, erhalten wir bei beiden Verfahren Ausreißer. Der Median von unserem Modell ist etwas höher. Sowohl die Whiskers und die Quartile sind komprimierter und ähneln einer Normalverteilung, genauer im Anhang in der Abbildung D.7. Jedoch bestätigte der Shapiro-Wilk-Test [Sha65] dies in beiden Ergebnissen nicht. Um die beiden Stichproben zu vergleichen, entscheiden wir uns für den Wilcoxon-Vorzeichen-Rang-Test (WVR-Test) [Bün94, S. 96] und den Varianz-Test [How60]. Der WVR-Test prüft die zentralen Tendenzen und der Varianz-Test die Varianz zweier Stichproben. Der WVR-Test führt zu einem p-Wert von $1,9e - 10$, der Varianz-Test einen p-Wert von 0,016 und beide weisen somit auf einen Unterschied beider Stichproben hin. Ziehen wir den Receiver Operating Characteristic (ROC)-Chart 4.14 und die Abbildung 4.13 unseren Erkenntnissen hinzu, können wir unser Vorhaben ein stabileres und genaueres Modell gefunden zu haben, bestätigen.

Der ROC-Chart [Faw04] zeigt ebenfalls eine nicht zufällige Klassifizierung. Der Vorteil eines ROC-Charts gegenüber der Genauigkeit ist, dass die Uneigengleichheit der Klassengröße durch relative Häufigkeitsverteilung, hier True Positiv Rate (TPR) und False Positiv Rate (FPR), eliminiert wird. Bezogen auf den Chart 4.14 heißt das: Je weiter unsere Kurve links oben liegt, desto besser ist die Qualität der Klassifikation. Ein weiterer sehr beliebter Wert zu dem Thema ist Area Under Curve (AUC). Mit AUC wird die Fläche unter der Kurve im ROC-Chart angegeben. Nähert sich der AUC-Wert 0,5 an, kann von einer zufälligen Klassifikation ausgegangen werden. Der beste AUC-Wert ist 1 oder -1.

Vergleichen wir die Ergebnisse mit den lexikalischen Verfahren C.1.3, konnte kein KNN-Modell eine Verbesserung der Klassifikationsqualität aufweisen. Gegenüber den maschinellen Verfahren C.2.3 ist das angepasste HDLTex-Modell ebenbürtig. Die genauen Messwerte sind in Anhang D.3 zu finden.

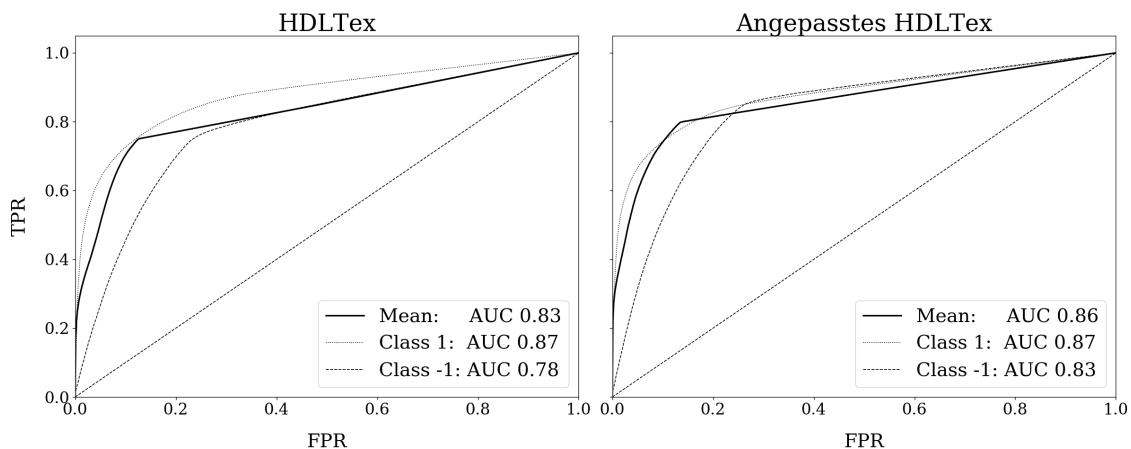


Abbildung 4.14: KNN: ROC-Chart. Der linke Plot zeigt die Ergebnisse als ROC-Chart des originalen HDLTex- und rechts das angepasstes Modell.

4.7.2 CNNSC

Die zweite Layer-Variante CNN, die weitaus beliebter in der Klassifikation von Text ist, verwendet Kim Y. im Paper “Convolutional Neural Networks for Sentence Classification” [Kim14]. Wir übernehmen das CNNSC-Modell. Ziel ist, ein einfaches und verbessertes Modell durch Anpassung der Hyperparameter zu finden.

Kim nutzt die Word2Vec-Vektoren aus dem Abschnitt 3.2.2 von Google News mit 100 Millionen Wörtern. Diese vortrainierten Wortvektoren bestehen aus einer Dimension von 300 Gewichten. Anders gesagt: Als Eingabe verwendet sein Netz einen Wortvektor von 300 Gewichten pro Wort über sein gesamten Vokabular, wodurch eine Gewichtungsmatrix der Länge des Vokabulars entsteht. Unser Vokabular sind die gesamten unterschiedlichen Wörter, die in unserem Trainingsdatensatz aus dem Abschnitt 3.2.1 vorkommen. Das Netz besteht aus einem CNN-Layer, Max-Pooling-Layer und einem KNN-Layer, siehe Abbildung 4.15. Der CNN-Layer besteht aus 100 Kernels der Größe 3, einen Drop-Out von

50 % und einer Regularisierung von 3. Die Anzahl der einstellbaren Parameter summiert sich auf ca. 2 Millionen. Die Autoren testen die Modelle auf unterschiedlichen Daten, wie Filmebewertung oder Stanford Sentiment Treebank [Kim14, S. 1748]. Ein Vorteil, der einem wegen unserer geringen Anzahl an Trainingsdaten ins Auge sticht, ist die geringe Anzahl an einstellbaren Parametern im Gegensatz zum vorigen HDLTex-Modell aus dem Abschnitt 4.7.1. Einer der Hauptunterschiede spiegelt sich bereits im Input-Layer

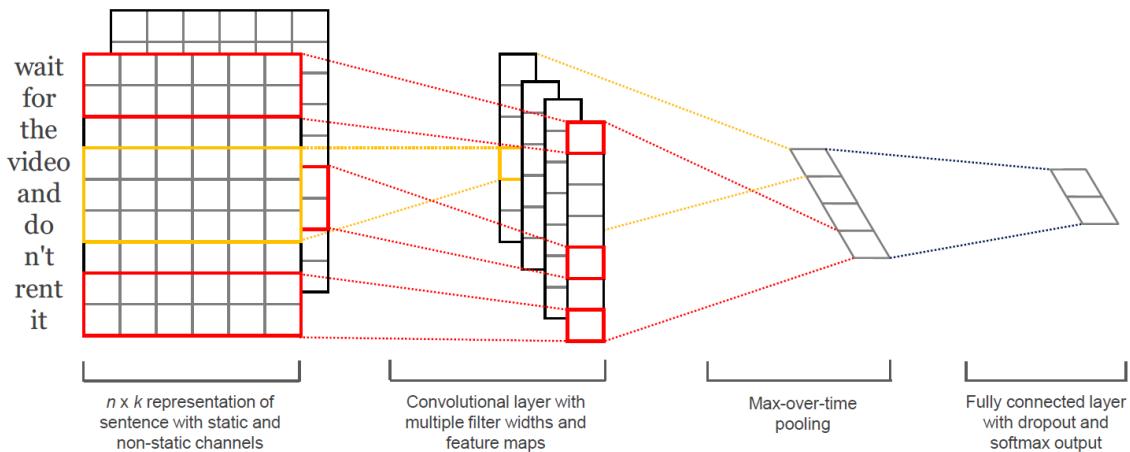


Abbildung 4.15: CNN: Referenzmodell [Kim14, S. 1747]. Beginnend links nach rechts ist der Input eine Gewichtungsmatrix von 300 und der Länge des Vokabulars. Danach folgt der CNN-Layer, Max-Pooling-Layer und das KNN.

wieder. Während wir im HDLTex-Modell eine One-Hot-Kodierung verwenden, nutzen wir für das CNNSC-Modell vortrainierte Word2Vec-Wortvektoren. Im Paper verwenden die Autoren die vortrainierten Vektoren auf Basis von Google News [Kim14, S. 1748], die wir ebenfalls begutachten. Weiterhin untersuchen wir die bereitgestellten Wortvektoren von GloVe [Pen14], die basierend auf alle Wikipedia-Artikel, zwei unterschiedliche Common Crawl Quellen und Tweets von Twitter erstellt wurden, siehe Übersichtstabelle 3.5. Die besten Zwischenergebnisse liefern die vortrainierten Twitter-Vektoren mit einer Gewichtslänge von 200, weshalb wir uns für diese entscheiden. Außerdem entfernen wir den Regularisierer, erhöhen die Anzahl der Kernel von 100 auf 128 und verringern die Größe der Kernels von 3 auf 1. Die Anzahl der einstellbaren Parameter summiert sich auf ca. 1,3 Millionen.

Abbildung 4.16 zeigt den Trainingsverlauf von CNNSC. Der linke Plot zeigt den Verlauf des originalen Modells und rechts unser angepasstes auf der Trainings- und Validierungsmenge. Zuerst fällt auf, dass die Konfidenzbänder, im Gegensatz zu den Ergebnissen aus KNN 4.7.1, kaum sichtbar sind. Das liegt an der geringen Abweichung der einzelnen Modelle. Nach einem genauen Blick scheint die Varianz der Modelle von CNNSC qualitativ größer zu sein, als im angepassten Modell. Außerdem benötigt das CNNSC eine längere Trainingsphase von mindestens 15 Epochen, um das optimale Modell zu erreichen, wohingegen unser angepasstes Modell bereits nach 5 Epochen optimal ist. Für eine genauere Betrachtung sind beide Plots in einer größeren Variante in Anhang D.3.1 zu finden.

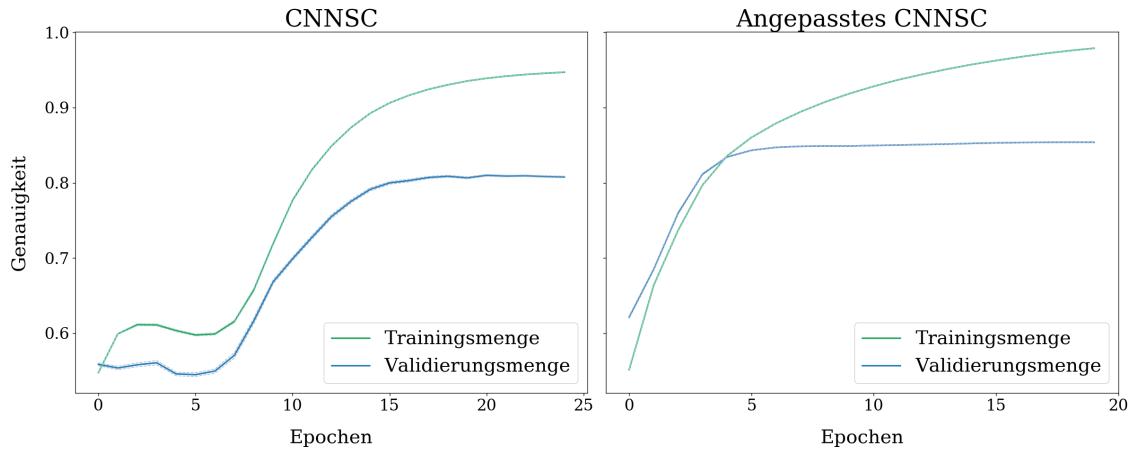


Abbildung 4.16: CNN: Verlauf der Genauigkeit. Der linke Plot zeigt das originale CNNSC Modell aus “Convolutional Neural Networks for Sentence Classification“ [Kim14, S. 1748]. Rechts ist unser angepasstes Modell zu sehen.

Abbildung 4.17 stellt die Güte mit Hilfe von Boxplots gegenüber. Wieder sind im Vergleich zu HDLTex wesentliche Verbesserungen im angepassten (rechts) und originalen (links) CNNSC zu vermerken. Besonders sticht die verbesserte Varianz beider Varianten heraus. Bis auf den Recall der Klasse 1 vermerken wir Verbesserungen. Besonders sticht der Recall mit einem verbesserten Median und einer verbesserten Varianz heraus. Anders ausgedrückt, scheinen unsere Änderungen die Trefferquote der gefundenen negativen Tweets in Bezug auf jede Klasse des Testdatensatzes sich verbessert zu haben. Die Ausreißer sind ebenfalls weniger geworden. Außerdem bestätigt uns der Boxplot über die Genauigkeit ebenfalls die Verbesserung mittels des angepassten Modells, siehe Abbildung D.10 im Anhang.

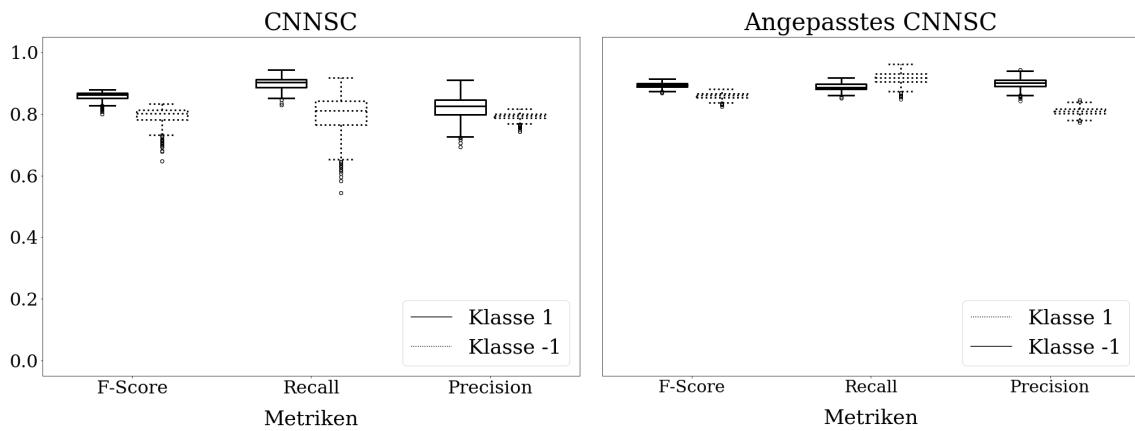


Abbildung 4.17: CNN: Güte. Beide Plots zeigen die Metriken F-Score, Recall und Precision (links das originale CNNSC- und rechts das angepasste CNNSC-Modell).

Nach der explorativen Analyse untermauern wir unsere Vermutung mittels statistischer Tests. Der Shapiro-Wilk-Test bestätigt die Normalverteilung beider Stichproben. Der WVR-Test errechnet einen p-Wert von $2,88e - 165$ und der Varianz-Test von $1,26e - 52$. Die Tests bestätigen unsere Vermutung und zeigen, dass die Unterschiede signifikant sind.

Die ROC-Kurve [4.18](#) zeigt ebenfalls, dass die Klassifikation nicht zufällig ist. Das originale CNNSC-Modell erreicht bessere Werte als die HDLTex-Modelle. Ein weiterer Vorteil ist die geringere Anzahl der Parameter der CNNSC-Modelle, woraus sich eine schnelle Berechnung der Sentiments ableitet. Außerdem erreichen wir mittels dem angepassten CNNSC-Modell ähnlich gute Ergebnisse wie mit vaderSentiment. Dabei sollten wir beachten, dass wir die Trennung der Tweets mittels vaderSentiment durchführen. Weiterhin sind die genauen Messwerte in Anhang [D.3](#) zu finden.

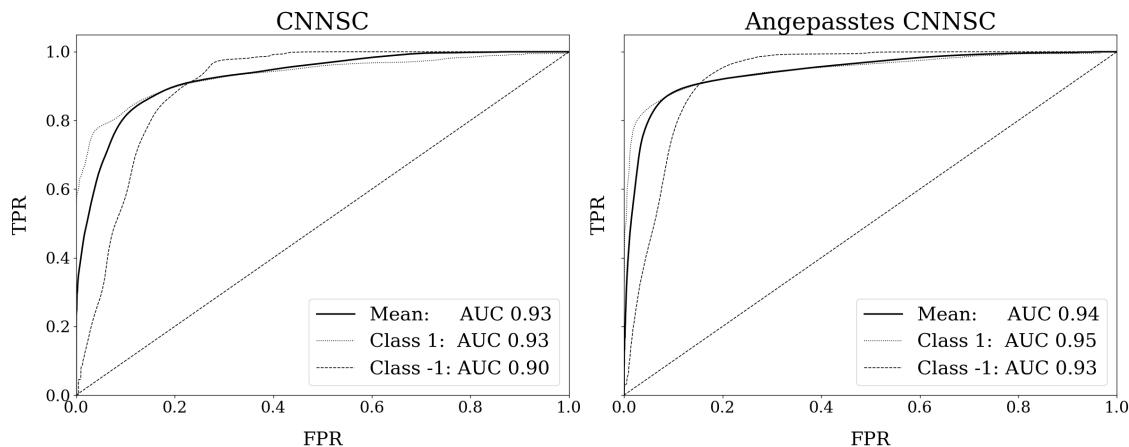


Abbildung 4.18: CNN: ROC-Chart. Der linke Plot zeigt die Ergebnisse als ROC-Chart des originalen HDLTex- und rechts das angepasste Modell.

4.7.3 Vergleich aller Modelle

In Anhang Vergleichsmethoden [C](#) untersuchen wir die lexikalischen Verfahren und trainieren die üblichen maschinellen Verfahren. Danach trainieren wir die originalen und angepassten HDLTex- und CNNSC-Modelle. Deren Ergebnisse diskutieren wir in diesem Kapitel.

Die Tabelle [4.2](#) zeigt eine Übersicht der Ergebnisse von allen Verfahren für die Klasse -1 (negativ) und die Klasse 1 (positiv). Wir bedienen uns an den Messwerten Genauigkeit oder Präzision (Prec von Precision), Trefferquote (Rec ova Recall) und F-Maß (F1 von F1-Score). Die Zeilen 1-2 repräsentieren das CNNSC- und das angepasste Modell, 3-4 das originale und angepasste CNNSC-Modell und die Zeilen 5-8 enthalten die Ergebnisse der maschinellen und der lexikalischen Vergleichsverfahren.

Eine Hypothese, die sich in den Ergebnissen aus der [4.2](#) widerspiegelt, ist das gute Abschneiden von vaderSentiment aus dem Anhang [C.1.1](#). Eine Ursache könnte in der Vorselektion der Tweets durch vaderSentiment vor der Annotation im Abschnitt [3.2.1](#) liegen. Bemerkenswert finden wir die Resultate von sentimentr aus dem Anhang [C.1.2](#),

das direkt nach dem angepassten CNNSC-Modell und vadersentiment anschließt. Eine Vermutung ist, dass sentimentr und vaderSentiment ähnliche Algorithmen für die Bewertung der Sätze verwenden. Die Verfahren, die am wenigsten gut auf unseren Fall passen, sind beide HDLTex-Modelle aus dem Abschnitt 4.7.1. Weiterhin enttäuschen die üblichen maschinellen Lernverfahren aus dem Anhang C.2. Wahrscheinlich aus dem Grund, weil wir diese lediglich zum Vergleich herangezogen haben und nur GridSearch [Ped11] für die Parameterfindung verwenden. Erst das originale und angepasste CNNSC-Modell ergeben ähnlich gute Ergebnisse im Vergleich zu den lexikalischen Verfahren, das unserer Meinung nach an zwei Möglichkeiten liegen kann. Hierbei müssen wir beachten, dass die Messwerte der Durchschnitt von 1.000 unterschiedlichen CNNSC-Modellen sind. Zum einen verwendet das CNN-Layer Kernels, die Lokalitäten erkennen, ähnlich wie ein horizontaler vs. vertikaler Kantendetektor unterschiedliche Muster erkennt. Anstatt mit einem Bild arbeiten wir mit einer *Gewichtsmatrix*, die wir ebenfalls als Matrix verwenden und somit wieder Ähnlichkeiten zu einem Bild aufweisen. Weiterhin verwenden wir als Input-Layer ein Embedding-Layer, der mit vortrainierten Wortvektoren von Google News für das originale und Twitter für unser angepasstes Modell arbeitet. Die zweite Möglichkeit, warum das Netz gut abschneidet, könnte an der *Einfachheit des Netzes* liegen. Wir haben lediglich 2 Millionen einstellbare Parameter im originalen und 1,3 Millionen im angepassten Modell. Gerade die Einfachheit der Modelle auf den wenigen Daten kann in diesem Fall zur Verbesserung der Klassifikation führen.

Tabelle 4.2: Prognoseergebnisse aller Verfahren. Die Zeilen 1-2 repräsentieren das CNNSC- und das angepasste Modell, 3-4 das originale und angepasste CNNSC-Modell und die Zeilen 5-8 enthalten die Ergebnisse der maschinellen und der lexikalischen Vergleichsverfahren. Die Ergebnisse der DL-Modelle sind der Durchschnitt von 1.000 unterschiedlich initialisierten Modellen.

#	Methoden	Klasse -1			Klasse 1			Insgesamt	
		Prec	Rec	F1	Prec	Rec	F1	F1	Acc
1.	CNNSC	0,79	0,80	0,79	0,82	0,90	0,86	0,78	0,80
2.	ang. CNNSC	0,81	0,92	0,86	0,90	0,89	0,89	0,83	0,85
3.	HDLTex	0,71	0,69	0,69	0,77	0,87	0,81	0,72	0,75
4.	ang. HDLTex	0,72	0,80	0,75	0,84	0,83	0,83	0,75	0,77
5.	RandomForest	0,80	0,68	0,73	0,88	0,85	0,86	0,76	0,73
6.	SVM	0,74	0,84	0,79	0,83	0,83	0,83	0,77	0,79
7.	vaderSentiment	0,77	0,96	0,85	0,95	0,85	0,90	0,83	0,85
8.	sentimentr	0,79	0,82	0,80	0,91	0,85	0,87	0,80	0,79

Unsere Ergebnisse zeigen, dass die lexikalischen Verfahren bereits gute Klassifizierer sind, gerade auf Basis einer kleinen Trainingsmenge. Wir zeigen, dass die DL-Modelle nach der aktuellen Forschung, hier CNN in Verbindung mit Embedding-Layer, ebenfalls das Niveau der lexikalischen Verfahren erreichen können. Schlussendlich sind wir positiv gestimmt, sollten wir die Trainingsmenge erhöhen, werden die DL-Modelle die anderen Verfahren ab einem Gewissen Punkte dominieren.

KAPITEL 5

Bitcoin und Sentiment

Prognosen waren bereits in der Antike ein großes Thema, indem Opfermaterialien verwendet wurden, um die Zukunft vorherzusagen [Mau05]. Im Prinzip versuchte der Mensch aus den vergangenen Erfahrungen die Zukunft, hier meist Warnungen, zu deuten. Wird dieses Prinzip auf die heutige Zeit übertragen, versuchen wir nichts anderes. Wir greifen auf Erfahrungswerte zu, hier der gegebene Twitterdatensatz in Form einer Zeitreihe aus dem Abschnitt 3.2.3, und versuchen anhand dieser und weiterer Informationen die Zukunft zu deuten.

Zu Beginn dieses Kapitels führen wir Zeitreihen ein, danach betrachten wir unsere Zeitreihe explorativ, anschließend untersuchen wir Korrelationen zwischen Bitcoin und unserem Sentiment und zum Ende des Kapitels bilden wir lineare Regressionen mit unterschiedlichen Prädiktoren, um Bitcoin durch unser Sentiment zu erklären.

Im Gegensatz zu den herkömmlichen Datensätzen, deren Ergebnisse weitestgehend unabhängig voneinander sind, spielt die Abhängigkeit der Datenpunkte einer Zeitreihe die elementare Rolle bei der Analyse. In den meisten Situationen sind Zeitreihen aufeinander folgende Messwerte, beispielsweise die stündliche Messung der Temperatur oder das Prognostizieren von Auslastungen. Dabei hängt die Qualität der Prognose von unterschiedlichen Faktoren ab [Hyn14, Kapitel 1.1].

- Wie vollständig ist der Datensatz verstanden?
- Wie viele Daten sind vorhanden?
- Die Vorhersage kann unsere Vorhersage beeinflussen.

Unsere Zeitreihe liegt meistens tabellarisch vor, die mindestens eine Spalte einer Zeitangabe und einen Messwert (Beobachtung) enthält [Hyn14, Kapitel 2.1]. Dieser Datensatz lässt sich leicht mittels eines Zeitdiagramms, Beobachtungen gegen die Zeit aufgetragen, darstellen [Hyn14, Kapitel 2.2], siehe Abbildung 5.1.

Weiterhin tauchen in Zeitreihen Muster auf, die in diesem Kontext wohl definiert werden müssen, auch Komponenten der Zeitreihe genannt [Hyn14, Kapitel 2.3]. Ein *Trend* beschreibt die Gesamtansicht der Reihe der Beobachtung, der steigen oder fallen kann. Der Begriff *Saisonal* definiert immer wieder regelmäßig auftretende Effekte, beispielsweise die Vierjahreszeiten. Der Unterschied zwischen Saisonalität und *Periodizität* ist die Unregelmäßigkeit und die höhere Fluktuation von gewöhnlich mehr als zwei Jahren. Beispiele für die Begrifflichkeiten sind in der Abbildung 5.1 zu finden.

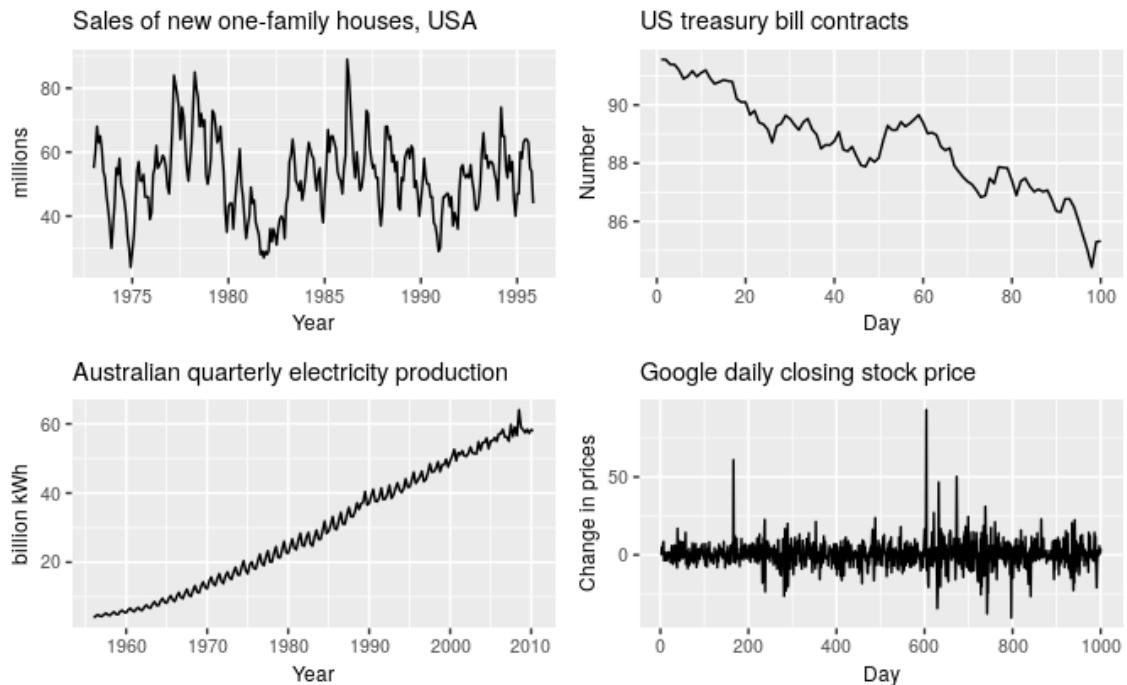


Abbildung 5.1: Komponenten der Zeitreihen [Hyn14, Abbildung 2.3]. *Links oben* repräsentiert die Verkäufe von Einfamilienhäusern in den USA und zeigt starke saisonale und zyklische (6–10 Jahre) Muster auf, jedoch keinen Trend. *Rechts oben* zeigt die Staatsanleihe von der United States, die lediglich einen Abwärtstrend aufweist. *Links unten* zeigt die Produktion von Strom in Australien, die einen steigenden Trend mit einem starken saisonalen Effekt aufweist. *Rechts unten* werden die Aktienpreise von Google angezeigt, die keine Effekte aufzeigen.

Als nächstes lernen wir extrem einfache Prognosemodelle kennen, um ein Gefühl für Prognosen mit Zeitreihen zu erhalten. Beispielsweise sieht die saisonale naive Methode (5.1), engl. seasonal naïve method, vor, den letzten Wert auf saisonaler Ebene als Prognosewert zu verwenden.

$$\hat{y}_{T+h|T} = y_{T-h-m(k+1)}, \quad (5.1)$$

wobei $\hat{y}_{T+h|T}$ die kurze Variante von y_{T+h} bedingt auf y_1, \dots, y_T ist. m gibt die Periode des Trendes und k für $h - 1/m$ als ganze Zahl an. Betrachten wir beispielsweise eine Zeitreihe mit einer Periode von einem ganzen Jahr und wollen den zukünftigen Wert für Februar vorhersagen, so entnehmen wir für die Prognose den letzten Wert von Februar aus dem Vorjahr. Weitere Verfahren sind die Durchschnittsmethode oder die naive Methode [Hyn14, Kapitel 3.1].

Als nächstes überlegen wir uns eine weitere Darstellung oder Zerlegung der Zeitreihe. Nach Hyndman R. [Hyn14, Kapitel 6.1] besteht die Möglichkeit die Zeitreihe in Komponenten aufzuteilen, wie Trend T_t , Saison S_t und Zyklisch R_t , siehe Formal (5.2).

$$y_t = S_t + T_t + R_t \quad (5.2)$$

In späteren Analysen erlaubt uns die Formel den Trend T_t , im weiteren Verlauf der Thesis als *Linearer Zeittrend* und engl. Linear Time Trend bezeichnet, durch Hinzufügen eines Prädiktors mit einer inkrementellen, aufsteigenden und endlichen Folge zu eliminieren.

Nach der Einführung betrachten wir unsere Zeitreihe. Unsere erste Intuition ist die Darstellung von dem Bitcoin-Niveau und der Bitcoin-Rendite gegen den jeweiligen Prädiktor. Ein Prädiktor repräsentiert diejenige Variable oder Spalte, die zum Vorhersagen der abhängigen Variable, in Deep Learning Target, dient. Eine interessante Ausgabe erhalten wir beim Gegenüberstellen von Bitcoin-Niveau gegen das CNNSC-Sentiment, das aus dem originalen CNNSC-Modell aus dem Abschnitt 4.7.2 generiert wird, siehe Abbildung 5.2. Die x-Achse repräsentiert den Zeitraum vom 10.01. - 31.08.2018 und die y-Achse das zwischen -1 und 1 normierte CNNSC-Sentiment (gestrichelt) bzw. das Bitcoin-Niveau (durchgängig). Wir glätten beide Kurven mit dem Savitzky-Golay-Filter [Sav64], da beide Kurven einem Signal nahe kommen, mit einer Fensterlänge von 75 und einem Polynomgrad von 3. Der eindimensionale Filter fährt über die Sequenz mit einer definierten Fensterlänge, einem definierten Polynomgrad und berechnet in jedem Schritt die polynomiale Regression. Das BTC-Niveau zeigt einen leichten Abwärtstrend. Weiterhin fällt bereits visuell auf, dass Verläufe zeitlich oft parallel ansteigen und wieder sinken. Dies versuchen wir mit weiteren Analysen im Folgenden zu untermauern.

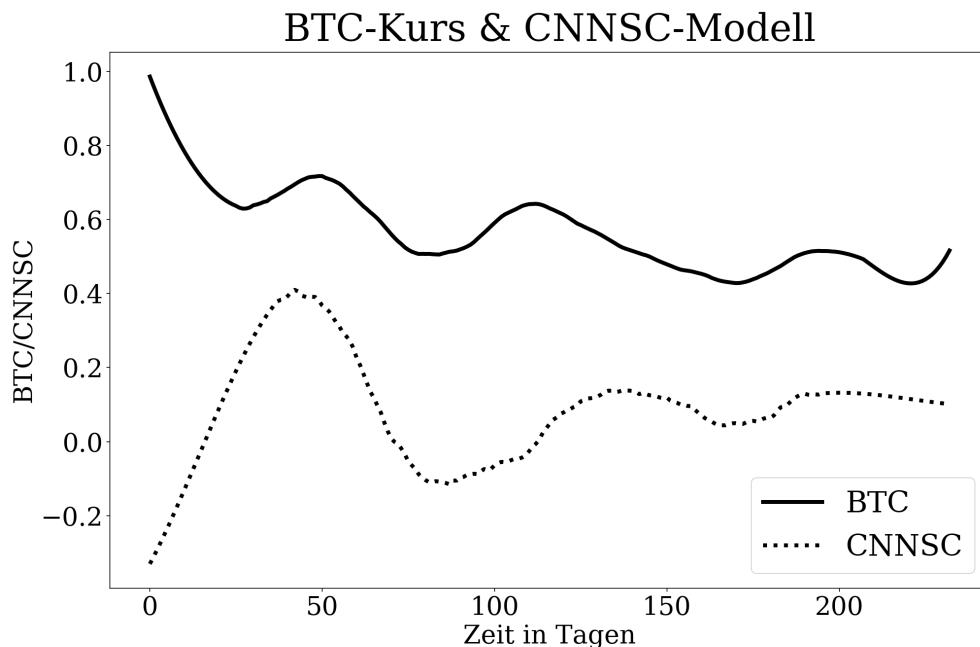


Abbildung 5.2: BTC-Kurs und CNNSC-Modell. Die x-Achse repräsentiert die Tage in einem Zeitraum vom 10.01. - 31.08.2018 und die y-Achse das Bitcoin-Niveau und den errechneten Sentiment vom CNNSC-Modell.

Da durch unseren CNNSC-Sentiment eine interessante Abbildung 5.2 entsteht und unsere Zeitreihe aus 78 Prädiktoren besteht, grenzen wir unsere Analysen auf das CNNSC-Sentiment und Bitcoin ein. Die Korrelation des CNNSC- und Bitcoin-Niveaus und der CNNSC- und Bitcoin-Rendite beträgt 0,11 und 0,044. Weiterhin beobachten wir vielversprechendere Korrelationen bis zu 0,57 und 0,085 mit unseren anderen Sentiments, genauer im Anhang in den Abbildungen E.1 und E.2.

Wie Cindy K. [Soo17] versuchen wir mit unterschiedlichen multiplen linearen Regressions [Hyn14, Kapitel 5.1] Zusammenhänge zwischen unserem CNNSC-Sentiment und der Bitcoin-Rendite zu finden. Nach einigen Tests erhalten wir hohe Signifikanzen aus dem Modell (5.3).

$$\Delta B = \Delta S_0 + \Delta S_1 + \Delta S_2 + \Delta x \quad (5.3)$$

Δ beschreibt die Variable als log-Rendite, beispielsweise berechnet sich unser Ziel $\Delta B = b_t - b_{t-1}$, wobei ΔB für die Bitcoin-Rendite und ΔS_t für die CNNSC-Sentiment-Rendite steht. Die Formel spiegelt sich in der Tabelle 5.1 wieder. In diesem Fall repräsentiert ΔS_0 unsere heutige Sentiment-Rendite, ΔS_1 die gestrige und ΔS_2 die vor zwei Tagen. Δx bindet die ökonomischen Variablen linearer Zeittrend, Anzahl der Tweets, das Volumen und die Marktkapitalisierung von Bitcoin ein, mit dem Zweck, bestimmte Effekte zu eliminieren. In Prinzip betrachten wir die letzten zwei Tage des Sentiments inkl. heute und versuchen linear einen Bezug zur BTC-Rendite zu finden.

Nach der Bildung unserer multiplen linearen Regression erhalten wir unsere Ergebnisse in Form von R^2 und Signifikanz der Koeffizienten. Da wir mit einer Zeitreihe arbeiten, verwenden wir für die Koeffizienten den NeweyWest-Schätzer [New86], der gegen Autokorrelation und Heteroskedastizität robust ist. Unsere Ergebnisse erhalten wir in der Tabelle 5.1 in Spalte (1) ohne ökonomische Prädiktoren. Alle drei unserer Prädiktoren sind hoch signifikant. Dadurch erhalten wir ein angepasstes R^2 von 10,2 % mit 231 Beobachtungen. Der untere zugehörige Wert von unseren Koeffizienten entspricht den Standardfehlern. Dem Modell aus der Spalte (2) fügen wir einen *linearen Zeittrend* hinzu, der den Trend aus unseren Prädiktoren ΔS_0 , ΔS_1 und ΔS_2 entfernt, wodurch sie sich nur geringfügig ändern. Ein weiterer Verdacht ist, dass die Anzahl der Tweets pro Tag stark im Modell vertreten sein könnte, weswegen der Prädiktor *#Tweets* in Spalte (3) diese versucht zu eliminieren. Trotzdem bleiben unsere Prädiktoren, die aus dem Sentiment stammen, hoch signifikant, weswegen wir einen geringen Einfluss vermuten. Spalte (4) und (5) fügen wir das Volumen und die Marktkapitalisierung von Bitcoin hinzu, die aus S_0 Informationen ziehen, sodass die Signifikanz von S_0 sich auf sehr signifikant reduziert.

Unsere Haupthypothese ist das Nachweisen einer Verbindung zwischen dem Sentiment von Twitter und dem Bitcoin-Niveau. Mittels unserer statistischen Verfahren können wir diese Hypothese mit einem Sentiment aus unserem Modell bestätigen und nachweisen, dass das Niveau sich zu teilen aus der Meinung der Twitter-Nutzer zusammensetzt. Dadurch erweitern wir die Theorie von Keynes J. [Key18], die von einer nicht rationalen Ökonomie ausgeht, indem wir seine Gedanken auf neuartige Systeme übertragen, wie Twitter und Bitcoin.

Tabelle 5.1: Drei Tage Sentiment-Prognose für die Bitcoin-Rendite. S_0 steht für die heutige Rendite, S_1 die gestrige Rendite und S_2 für die Rendite vor zwei Tagen des CNNSC-Sentiments. Die Koeffizienten berechnen wir mit NeweyWest und der zugehörige Standardfehler steht jeweils in der unteren Zeile. Der *Lineare Zeittrend* eliminiert den Trend, *#Tweets* repräsentiert die Anzahl der Tweets pro Tag, *Handelsvolumen* ist die Betragsmenge pro Tag, die gehandelt wird, und *Marktkapitalisierung* der Gesamtwert des Bitcoins.

	log-Rendite Bitcoin				
	(1)	(2)	(3)	(4)	(5)
ΔS_0	0,183*** (0,062)	0,183*** (0,069)	0,170*** (0,065)	0,188** (0,075)	0,182** (0,072)
ΔS_1	0,217*** (0,050)	0,216*** (0,050)	0,199*** (0,047)	0,209*** (0,052)	0,223*** (0,048)
ΔS_2	-0,171*** (0,047)	-0,174*** (0,045)	-0,180*** (0,043)	-0,182*** (0,046)	-0,181*** (0,044)
Linearer Zeittrend		✓	✓	✓	✓
#Tweets			✓	✓	✓
Volumen				✓	✓
Marktkapitalisierung					✓
Beobachtungen	231	231	231	231	231
Angepasstes R^2	0,102	0,117	0,121	0,127	0,127

Notiz:

* p < 0,1; ** p < 0,05; *** p < 0,01

KAPITEL 6

Ausblick

Wie in jedem Projekt müssen an einem gewissen Punkt Entscheidungen getroffen werden, wofür die Abzweigungen zu groß oder die Ressourcen zu knapp sind. Mit diesem Wissen werfen wir einen Rückblick auf die Thesis und diskutieren weitere Optionen, die uns während dem Projekt auffallen und entscheidende Einflüsse auf das Ergebnis haben könnten.

Einer der ersten größten Einflussfaktoren auf das Ergebnis ist die Größe des *Trainingsdatensatzes* aus dem Abschnitt 3.2.1. Die Annotation der Tweets von unabhängigen Individuen ist kostspielig und abhängig vom Budget. Unser Budget reicht für 1,857 siebenfach annotierte Tweets. Uns ist bewusst, dass die Anzahl der annotierten Tweets für DL-Verfahren wenig ist. Jedoch veröffentlichten Qurat Til Ain et al. [Ain17] 2017 eine Liste von Projekten mit DL-Verfahren, die erfolgreich mit einer ähnlich geringen Anzahl an annotierten Texten funktionieren. Nichtsdestotrotz erreichen DL-Netze ihren Höhepunkt erst ab einer gewissen Menge an Trainingsdaten. Neben der Annotation selektieren wir die Tweets bereits stark mit vaderSentiment aus dem Anhang C.1.1 vor. Unsere Idee besteht darin möglichst viele Informationen stark ausgerichteter Tweets in positiver und negativer Richtung zu entnehmen. Eine mögliche Verbesserung könnte das Hinzuziehen neutraler Tweets sein, die den DL-Netzen ermöglichen die Tweets neutral zu klassifizieren. Ursprünglich versuchen wir dies über einen Threshold zu lösen, der ab einem Schwellenwert die Tweets als neutral klassifiziert. Hierbei steht zur Diskussion keine vorselektierten Tweets annotieren zu lassen, da diese wahrscheinlich eine hohe neutrale Anzahl von Tweets enthalten. Diese Daten könnten zu unserem Trainingsdatensatz gemischt werden. Weiterhin spielt der Threshold in unserem Projekt eine Rolle, da wir zu Beginn versuchen, die Tweets in drei Klassen positive/negativ/neutral einzuteilen. Dieser Threshold ist ebenfalls ein Projekt-Tuning-Paramter, der das Ergebnis stark beeinflussen kann. Neben dem Threshold könnten die Tweets in mehr als drei Klassen verfeinert werden. Ein mögliches Szenario könnte stark positiv/positive/neutral/negativ/stark negativ sein. Eine weitere Hilfe, um die Qualität des Trainingsdatensatz zu erhöhen, sind statistische Qualitätsverfahren, siehe Krippendorf Alpha im Abschnitt 3.2.1. Zwecks der Größe im Bezug auf DL-Lernverfahren unseres Trainingsdatensatz, haben wir uns entschieden diese zu ignorieren, da wir die Menge der annotierten Tweets verringern würden. Ansonsten fällt uns beim Erstellen der Abbildung B.1 Verteilung der Hashtags auf, dass dies ein weiteres Feature für die Vorhersage oder Korrelation sein könnte. Hierbei könnte man die themenbezogenen Hashtags gruppieren und ihre Anzahl in Relation zueinander setzen. Dabei könnte ein Wert abfallen, der als Indikator für die Beliebtheit der jeweiligen Kryptowährung steht.

Weitere Projekt-Tuning-Parameter finden wir in der *Datenbereinigung* aus dem Abschnitt 3.2.2 des Twitter-Testdatensatzes. Im zweiten Schritt entfernen wir die Retweets. Wir sehen den Schritt insofern kritisch, da die Retweets ebenfalls im Netz gestreut werden und wiederum einen zusätzlichen Einfluss entfalten. Die Überlegung besteht darin, die Retweets dem Datensatz nicht zu entfernen und gleich, schwächer zu gewichten oder sogar als Feature einfließen zu lassen. Außerdem fällt uns auf, dass eine beträchtliche Menge an Bots, eine Software, die automatisch Aufgaben erfüllt [Dun08, S. 1], in den Tweets vorhanden sind. Die Überlegung ist diese Tweets ebenfalls zu entfernen oder anders zu gewichten. Daneben konvertieren wir die Tweets zu Lowercase, wodurch wir wahrscheinlich Informationen verlieren. Weiterhin könnte der Datensatz und die Modelle durch Feature Engineering angereichert werden, beispielsweise mittels externer und öffentlichen Datenquellen, wie Knowledge Trees. Ähnlich zu Word2Vec aus dem Abschnitt 3.2.2 ist Doc2Vec, wobei der Tweet als Dokument betrachtet werden könnte. Weiterhin könnte eine detaillierte Auseinandersetzung im Bereich Natural Language Processing (NLP) sich positiv auf das Ergebnis auswirken. In Python existieren dazu bereits Bibliotheken [Gar17]. Word2Vec sind vortrainierte Word-Vektoren, die mit einem sehr großen textuellen, unstrukturierten Datensatz trainiert wurden. Da unser Datensatz aus 1,36 Milliarden Wörtern besteht, kam die Überlegung einen eigenen und zugeschnittenen Word-Vektor speziell auf Kryptowährung zu erstellen. Bei unserem Download der Tweets aus dem Abschnitt 3.2 verwendeten wir unterschiedliche Hashtags, die auch zu anderen Kryptowährungen gehören könnten, vgl. Abschnitt B.1. Diese Tweets wurden nicht gefiltert. Eine Filterung könnte das Endergebnis beeinflussen. Generell besteht unserer Meinung nach eine große Möglichkeit im Bereinigungs-Schritt weitere relevante Informationen bezüglich des Projektes zu entdecken.

Neben der Wahl der *DL*-Netze aus dem Abschnitt 4.7 können wir aus Zeitgründen das Standardverfahren Cross-Validation nicht implementieren. Wir sind uns sicher, dass dieses Verfahren eine große Auswirkung auf die Stabilität der Netze hat. Eine weitere, in der Community anerkannte Möglichkeit Cross-Validation und zudem das Finden von guten Hyperparameter eines Netzes zu ermöglichen, ist die Methode *GridSearchCV* vom Python-Package *sklearn* zu verwenden. Hierbei führt die Methode selbstständig Cross-Validation durch und versucht damit die besten Hyperparameter des Netzes zu finden. Abgesehen von dem pre-trained Word-Embedding können ebenfalls pre-trained KNNs oder CNNs verwendet werden. Dabei besteht die Möglichkeit, den letzten oder mehr Layer zu löschen und dafür seine eigenen Layer zu verwenden. Die übrigen pre-trained Layer können fixiert werden, also auf nicht *trainable* gesetzt werden. Dadurch wird verhindert, dass das komplette Netz neu trainiert werden muss und die Informationen aus dem pre-trained Netz werden nicht eliminiert. Weiterhin könnten anstatt KNNs und CNNs auch RNNs verwendet werden. Unser ausgewähltes RNN-Modell bietet den Vorteil, dass die Tweets als Sequenzen verwendet werden, sodass eine zeitliche Komponente einen Vorteil bringen könnte. Weiterhin erfreut sich die Textklassifikation im Bereich der CNNs großer Beliebtheit, indem auf zahlreiche Modellvorschläge zurückgegriffen werden kann. Beispielsweise arbeiten X. Zhang et. al. [Zha16] an einem CNN-Layer, der auf Buchstaben-Level arbeitet anstatt auf Wörtern oder K. Kowsari et. al. [Kow18] zeigen eine Möglichkeit n-Modelle parallel zu generieren und diese mit einem Voting-Verfahren das wahrscheinlichste Ergebnis wählen zu lassen.

Das Thema *Zeitreihenanalyse* des Abschnittes 5 wurde aus zeitlichen Gründen nicht ausführlich behandelt. Neben der großen Maschinerie an Zeitreihenmethodiken könnte man die Zeitreihe granularer oder größer aufbauen. Unsere Zeitreihe repräsentiert für jeden Datenpunkt einen Tag. Da die Anzahl der Tweets in die Millionen geht, könnte man einen Datenpunkt auf eine, acht oder zwölf Stunden feiner granulieren. Außerdem besteht die Möglichkeit, die Zeitreihe nicht mit ± 1 pro Tweet zu kummulieren. DL-Modelle und andere Verfahren werfen als Ergebnis Wahrscheinlichkeiten oder Ausrichtungen aus, aus denen wiederum kummuliert oder der Schnitt/Median berechnet werden kann. Wir finden mittels dem originalen CNNSC-Sentiment einen hoch signifikanten Nachweis für den Einfluss auf Bitcoin. Wegen dem Ergebnis sind wir zuversichtlich, dass weitere wichtige Informationen über die Stimmung in der Zeitreihe enthalten sind, da wir nur ein von 48 Modellen betrachten.

Schlussendlich können wir behaupten, dass wir lediglich an der Oberfläche kratzen und eine Vielzahl an Tuning-Parameter angepasst werden können, die das Endergebnis beeinflussen.

Literatur

- [Ain17] AIN, QURAT TIL, MUBASHIR ALI und AMNA RIAZ: „Sentiment Analysis Using Deep Learning Techniques: A Review“. *International Journal of Advanced Computer Science and Applications* (2017), Bd. (siehe S. 49).
- [Bün94] BÜNING, HERBERT und GÖTZ TRENKLER: *Nichtparametrische statistische Methoden*. Hrsg. von BÜNING, HERBERT und GÖTZ TRENKLER. Walter de Gruyter, 1994 (siehe S. 37).
- [Cha18] CHAN, SOPHIA und ALONA FYSHE: „Social and Emotional Correlates of Capitalization on Twitter“. *Association for Computational Linguistics* (2018), Bd. (siehe S. 10).
- [Cha78] CHARLES, KINDLEBERGER: „Manias, Panics, and Charts: A History of Financial Crises“. *Oxford University Press* (1978), Bd. (siehe S. 3).
- [Cho15] CHOLLET, FRANÇOIS: *Keras*. 2015. URL: www.keras.io (siehe S. 35).
- [Cor95] CORTES, CORINNA und VLADIMIR VAPNIK: „Support-Vector Networks“. *Machine Learning* (Sep. 1995), Bd. 20(3): S. 273–297 (siehe S. 75).
- [Dun08] DUNHAM, KEN und JIM MELNICK: *Malicious Bots: An Inside Look into the Cyber-Criminal Underground of the Internet*. Hrsg. von DUNHAM, KEN und JIM MELNICK. CRC Press, 2008 (siehe S. 50).
- [Faw04] FAWCETT, TOM: „ROC Graphs: Notes and Practical Considerations for Data Mining Researchers“. *HP Laboratories* (2004), Bd. (siehe S. 38).
- [Fer07] FERNÁNDEZ, SANTIAGO, ALEX GRAVES und JÜRGEN SCHMIDHUBER: „An Application of Recurrent Neural Networks to Discriminative Keyword Spotting“. *Proceedings of the 17th International Conference on Artificial Neural Networks*. ICANN'07. Porto, Portugal: Springer-Verlag, 2007: S. 220–229 (siehe S. 33).
- [For11] FORT, KARËN: *Amazon Mechanical Turk: Gold Mine or Coal Mine?* Hrsg. von FORT, KARËN. Association for Computational Linguistics, 2011 (siehe S. 7).
- [Gar01] GARBER, PETER M.: *Famous First Bubbles: The Fundamentals of Early Manias*. Hrsg. von GARBER, PETER M. The MIT Press, 2001 (siehe S. 3).
- [Gar17] GARRETTE, DAN, PETER LJUNGLÖF und JOEL NOTHMAN: *Natural Language Toolkit*. 2017. URL: www.nltk.org (siehe S. 50).
- [Gil01] GILES, C. LEE, STEVE LAWRENCE und AH CHUNG TSOI: „Noisy Time Series Prediction using Recurrent Neural Network and Grammatical Inference“. *Kluwer Academic Publisher* (2001), Bd. (siehe S. 33).

- [Glo10] GLOROT, XAVIER und YOSHUA BENGIO: „Understanding the Difficulty of Training Deep Feedforward Neural Networks“. *Journal of Machine Learning Research* (2010), Bd. (siehe S. 28).
- [Gon14] GONCALVES, POLLYANNA, MATHEUS ARAUJO, FABRICIO BENEVENUTO und MEEYOUNG CHA: „Comparing and Combining Sentiment Analysis Methods“. *CoRR* (2014), Bd. abs/1406.0032 (siehe S. 4).
- [Goo16] GOODFELLOW, IAN, YOSHUA BENGIO und AARON COURVILLE: *Deep Learning*. Hrsg. von GOODFELLOW, IAN, YOSHUA BENGIO und AARON COURVILLE. MIT Press, 2016 (siehe S. 15, 16, 18, 19).
- [Gra13] GRAVES, A., A. MOHAMED und G. HINTON: „Speech Recognition with Deep Recurrent Neural Networks“. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. Mai 2013: S. 6645–6649 (siehe S. 15).
- [Gra08] GRAVES, ALEX, MARCUS LIWICKI und SANTIAGO FERNÁNDEZ: „A Novel Connectionist System for Unconstrained Handwriting Recognition“. *IEEE* (2008), Bd. (siehe S. 33).
- [Har13] HARRIS, DAVID MONEY und SARAH L. HARRIS: *Digital Design and Computer Architecture*. Hrsg. von GREEN, TODD. Second. Elsevier, Inc., 2013 (siehe S. 12).
- [Hay04] HAYKIN, SIMON: *Neural Networks - A Comprehensive Foundation*. Hrsg. von HAYKIN, SIMON. Pearson Education, 2004 (siehe S. 20).
- [He15] HE, KAIMING, XIANGYU ZHANG, SHAOQING REN und JIAN SUN: „Deep Residual Learning for Image Recognition“. *CoRR* (2015), Bd. abs/1512.03385 (siehe S. 32).
- [Hoc97] HOCHREITER, SEPP und JÜRGEN SCHMIDHUBER: „Long Short-Term Memory“. *Neural Computation* (1997), Bd. 9(8): S. 1735–1780 (siehe S. 33).
- [Hof06] HOFMANN, MARTIN: „Support Vector Machines—Kernels and the Kernel Trick“. *Notes* (2006), Bd. 26 (siehe S. 75).
- [How60] HOWARD, LEVENE: „In Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling“. *Stanford University Press* (1960), Bd. (siehe S. 37).
- [Hut14] HUTTO, C. J.: *VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text*. Hrsg. von HUTTO, C. J. Association for the Advancement of Artificial Intelligence, 2014 (siehe S. 71, 72).
- [Hyn14] HYNDMAN, R.J. und G. ATHANASOPOULOS: *Forecasting: Principles and Practice*. OTexts, 2014 (siehe S. 43, 44, 46).
- [Inc09] INC., MONGODB: *MongoDB*. 2009. URL: www.mongodb.com (siehe S. 59).
- [Key18] KEYNES, JOHN MAYNARD: *The General Theory of Employment, Interest and Money*. Springer, 2018 (siehe S. 3, 46).
- [Kim14] KIM, YOON: „Convolutional Neural Networks for Sentence Classification“. *Association for Computational Linguistics* (2014), Bd. (siehe S. 13, 31, 35, 38–40).

- [Kim18] KIM, YOUNG BIN ET AL: „When Bitcoin Encounters Information in an Online Forum: Using Text Mining to Analyse User Opinions and Predict Value Fluctuation“. *PLoS ONE* (2018), Bd. (siehe S. 4).
- [Kow17] KOWSARI, KAMRAN, DONALD E. BROWN und MOJTABA HEIDARYSAFA: „Hierarchical Deep Learning for Text Classification“. *arXiv* (2017), Bd. (siehe S. 35, 36).
- [Kow18] KOWSARI, KAMRAN, MOJTABA HEIDARYSAFA, DONALD BROWN und KIANA JAFARI MEIMANDI: „Random Multimodel Deep Learning for Classification“. *Research Gate* (2018), Bd. (siehe S. 50).
- [Kri03] KRIPPENDORF, KLAUS: „Reliability in Content Analysis: Some Common Misconceptions and Recommendations“. *Human Communication Research* (2003), Bd. (siehe S. 9).
- [Kri12] KRIZHEVSKY, ALEX, ILYA SUTSKEVER und GEOFFREY E. HINTON: „ImageNet Classification with Deep Convolutional Neural Networks“. (2012), Bd.: S. 1097–1105 (siehe S. 32).
- [Kum17] KUMAR, SIDDHARTH KRISHNA: „On Weight Initialization in Deep Neural Networks“. *Arxiv* (2017), Bd. (siehe S. 28).
- [Li18] LI, FEI-FEI: „Convolutional Neural Networks for Visual Recognition“. *Convolutional Neural Networks for Visual Recognition*. 2018 (siehe S. 31–34).
- [Liu16] LIU, PENGFEI, XIPENG QIU und XUANJIING HUANG: „Recurrent Neural Network for Text Classification with Multi-Task Learning“. *Twenty-Fifth International Joint Conference on Artificial Intelligence* (2016), Bd. (siehe S. 34).
- [Lom02] LOMBARD, MATTHEW, JENNIFER SNYDER-DUCH und CHERYL C. BRACKEN: „Content Analysis in Mass Communication: Assessment and Reporting of Intercoder Reliability“. *Communication Faculty Publications* (2002), Bd. (siehe S. 9).
- [Man09] MANNING, CHRISTOPHER, PRABHAKAR RAGHAVAN und HINRICH SCHÜTZE: *An Introduction to Information Retrieval*. Hrsg. von MANNING, CHRISTOPHER, PRABHAKAR RAGHAVAN und HINRICH SCHÜTZE. Cambridge University Press, 2009 (siehe S. 13).
- [Män16] MÄNTYLÄ, MIKA VIKING, DANIEL GRAZIOTIN und MIKKI KUUTILA: „The Evolution of Sentiment Analysis - A Review of Research Topics, Venues, and Top Cited Papers“. *CoRR* (2016), Bd. abs/1612.01556 (siehe S. 3, 16).
- [Mar13] MARTÍEZ-CÁMARA, E.: *SINAI: Machine Learning and Emotion of the Crowd for Sentiment Analysis in Microblogs*. Hrsg. von MARTÍEZ-CÁMARA, E. Association for Computational Linguistics, 2013 (siehe S. 10).
- [Mat03] MATSUGU, MASAKAZU, KATSUHIKO MORI, YUSUKE MITARI und YUJI KANEDA: „Subject Independent Facial Expression Recognition with Robust Face Detection using a Convolutional Neural Network“. *Neural Networks* (2003), Bd. 16(5). Advances in Neural Networks Research: IJCNN '03: S. 555–559 (siehe S. 31).

- [Mau05] MAUL, STEFAN: „Omina und Orakel : A. Mesopotamien“. *De Gruyter* (2005), Bd. (siehe S. 43).
- [Mik13] MIKOLOV, TOMAS, KAI CHEN, GREG CORRADO und JEFFREY DEAN: „Efficient Estimation of Word Representations in Vector Space“. *arXiv* (2013), Bd. (siehe S. 12).
- [Mni15] MNIH, VOLODYMIR, KORAY KAVUKCUOGLU, DAVID SILVER, ANDREI A. RUSU und JOEL VENESS: „Human-level Control through Deep Reinforcement Learning“. *Nature Publishing Group* (2015), Bd. (siehe S. 16).
- [Nak08] NAKAMOTO, SATOSHI: „Bitcoin: A Peer-to-Peer Electronic Cash System“. (2008), Bd. (siehe S. 5).
- [Nee13] NEETHU, M. S. und R. RAJASREE: „Sentiment Analysis in Twitter Using Machine Learning Techniques“. *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*. Juli 2013: S. 1–5 (siehe S. 75).
- [New86] NEWEY, WHITNEY K und KENNETH D WEST: *A Simple, Positive Semi-definite, Heteroskedasticity and Autocorrelationconsistent Covariance Matrix*. 1986 (siehe S. 46).
- [Ng15] NG, ANDREW: *Deep Learning Specialization*. 2015. URL: www.deeplearning.ai (siehe S. 17, 18, 20, 21, 26–30).
- [Oed06] OEDA, SHINICHI, IKUSABURO KURIMOTO und TAKUMI ICHIMURA: „Time Series Data Classification Using Recurrent Neural Network with Ensemble Learning“. *Knowledge-Based Intelligent Information and Engineering Systems*. Hrsg. von GABRYS, BOGDAN, ROBERT J. HOWLETT und LAKHMI C. JAIN. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006: S. 742–748 (siehe S. 33).
- [OpC13] OPCO, COINMARKETCAP: *Coin Market Cap*. CoinMarketCap. 2013. URL: www.coinmarketcap.com (siehe S. 14).
- [Pan08] PANG, BO und LILLIAN LEE: „Opinion Mining and Sentiment Analysis“. *Foundations and Trends® in Information Retrieval* (2008), Bd. (siehe S. 16, 17).
- [Par14] PARMAR, Hitesh, SANJAY BHANDERI und GLORY SHAH: *Sentiment Mining of Movie Reviews using Random Forest with Tuned Hyperparameters*. Juli 2014 (siehe S. 74).
- [Ped11] PEDREGOSA, F. u. a.: „Scikit-learn: Machine Learning in Python“. *Journal of Machine Learning Research* (2011), Bd. 12: S. 2825–2830 (siehe S. 42, 66, 76).
- [Pen14] PENNINGTON, JEFFREY, RICHARD SOCHER und CHRISTOPHER D. MANNING: *GloVe: Global Vectors for Word Representation*. 2014. URL: nlp.stanford.edu/projects/glove (siehe S. 13, 39).
- [Pow15] POWERS, DAVID MARTIN WARD: „Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness and Correlation“. *International Journal of Machine Learning Technology* (2015), Bd. (siehe S. 36).

- [Qui85] QUINLAN, J.R.: „Induction of Decision Trees“. *Centre for Advanced Computing Sciences* (1985), Bd. (siehe S. 73).
- [Ram17] RAMACHANDRAN, PRAJIT, BARRET ZOPH und QUOC V. LE: „Searching for Activation Functions“. *CoRR* (2017), Bd. (siehe S. 26).
- [Rea05] READ, JONATHON: „Using Emoticons to Reduce Dependency in Machine Learning Techniques for Sentiment Classification“. *Proceedings of the ACL Student Research Workshop*. ACLstudent '05. Ann Arbor, Michigan: Association for Computational Linguistics, 2005: S. 43–48 (siehe S. 4).
- [Rin18] RINKER, TYLER: *Sentimentr*. 2018. URL: www.github.com/trinker/sentimentr (siehe S. 72).
- [Ros03] ROSASCO, L., E. DE VITO, A. CAPONNETTO, M. PIANA und A. VERRI: „Are Loss Functions All the Same?“ *Neural Computation* (2003), Bd. (siehe S. 28).
- [Rus15] RUSSAKOVSKY, OLGA, JIA DENG, HAO SU, JONATHAN KRAUSE, SANJEEV SATHEESH, SEAN MA, ZHIHENG HUANG, ANDREJ KARPATHY, ADITYA KHOSLA, MICHAEL BERNSTEIN, ALEXANDER C. BERG und LI FEI-FEI: „ImageNet Large Scale Visual Recognition Challenge“. *International Journal of Computer Vision (IJCV)* (2015), Bd. 115(3): S. 211–252 (siehe S. 15, 32).
- [Sak14] SAK, HASIM, ANDREW W. SENIOR und FRANÇOISE BEAUFAYS: „Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition“. *CoRR* (2014), Bd. abs/1402.1128 (siehe S. 33).
- [Sav64] SAVITZKY, ABRAHAM. und M. J. E. GOLAY: „Smoothing and Differentiation of Data by Simplified Least Squares Procedures.“ *Analytical Chemistry* (1964), Bd. 36(8): S. 1627–1639 (siehe S. 2, 45).
- [Sha65] SHAPIRO, S. S. und M. B. WILK: „An Analysis of Variance Test for Normality“. *Biometrika* (1965), Bd. (siehe S. 37).
- [Sil17a] SILVER, DAVID, THOMAS HUBERT, JULIAN SCHRITTWIESER, IOANNIS ANTONOGLOU, MATTHEW LAI, ARTHUR GUEZ, MARC LANCTOT, LAURENT SIFRE, DHARSHAN KUMARAN, THORE GRAEPEL, TIMOTHY P. LILLICRAP, KAREN SIMONYAN und DEMIS HASSABIS: „Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm“. *CoRR* (2017), Bd. abs/1712.01815 (siehe S. 15).
- [Sil17b] SILVER, DAVID, JULIAN SCHRITTWIESER, KAREN SIMONYAN, IOANNIS ANTONOGLOU, AJA HUANG, ARTHUR GUEZ, THOMAS HUBERT, LUCAS BAKER, MATTHEW LAI, ADRIAN BOLTON u. a.: „Mastering the Game of Go without Human Knowledge“. *Nature* (2017), Bd. 550(7676): S. 354 (siehe S. 15).
- [Soo17] SOO, CINDY K.: „Quantifying Sentiment with News Media across Local Housing Markets“. *The Review of Financial Studies* (2017), Bd. (siehe S. 3, 46).
- [Stu08] STUFFT, DONALD: *PIP Installs Packages*. 2008. URL: www.pypi.org/project/pip/ (siehe S. 60).

- [Sze15] SZEGEDY, CHRISTIAN, WEI LIU, YANGQING JIA, PIERRE SERMANET, SCOTT REED, DRAGOMIR ANGUELOV, DUMITRU ERHAN, VINCENT VANHOUCKE, ANDREW RABINOVICH u. a.: „Going Deeper with Convolutions“. Cvpr. 2015 (siehe S. 32).
- [Tau10] TAUSZIK, YLA R und JAMES W PENNEBAKER: „The Psychological Meaning of Words: LIWC and Computerized Text Analysis Methods“. *Journal of language and social psychology* (2010), Bd. 29(1): S. 24–54 (siehe S. 4).
- [Ten15] TENSORFLOW: *TensorBoard: Visualizing Learning*. 2015. URL: www.tensorflow.org/guide/summaries_and_tensorboard (siehe S. 81, 85).
- [The13] THELWALL, MIKE: „Heart and Soul: Sentiment Strength Detection in the Social Web with Sentistrength“. *Cyberemotions: Collective emotions in cyberspace* (2013), Bd. (siehe S. 4).
- [Twi18] TWITTER: *Häufig gestellte Fragen für neue Nutzer*. German. Twitter. Okt. 2018. URL: help.twitter.com/de/new-user-faq (siehe S. 6).
- [Twi17] TWITTER: *Q3 2017 Letter to Shareholders*. Techn. Ber. Twitter International Company, 2017 (siehe S. 6).
- [Wan15] WANG, LIMIN, SHENG GUO, WEILIN HUANG und YU QIAO: „Places205-VGGNet Models for Scene Recognition“. *CoRR* (2015), Bd. abs/1508.01667 (siehe S. 32).
- [Yao07] YAO, YUAN, LORENZO ROSASCO und ANDREA CAPONNETTO: „On Early Stopping in Gradient Descent Learning“. *Springer-Verlag* (2007), Bd. (siehe S. 29).
- [Zha16] ZHANG, XIANG, JUNBO ZHAO und YANN LECUN: „Character-level Convolutional Networks for Text Classification“. *arXiv* (2016), Bd. (siehe S. 50).
- [Zhe18] ZHENG, ALICE: *Mastering Feature Engineering*. Hrsg. von ZHENG, ALICE. O'Reilly, 2018 (siehe S. 12, 13).

A Technische Rahmenbedingung

In diesem Anhang werden die verwendeten Technologien zusammengetragen.

A.1 Downloadskript

Das Downloadskript läuft auf einem Raspberry PI 2 mit dem Betriebssystem Raspbian GNU/Linux 9. Die Kernel Version ist 4.14.30-v7.

Als Server-API verwenden wir Apache 2 mit der PHP-Version 7.0. Mehr Informationen zu PHP liegen im Anhang *Anforderungen/phpinfo().html*. Zu Beginn verwenden wir eine MongoDB 3.4 [Inc09] als Speicherort, jedoch fällt die Datenbank nach einer Größe von zwei GB einer Collection aus, weswegen wir uns schlussendlich für die Speicherung direkt auf das Dateisystem entscheiden.

A.2 Python

Der Hauptteil der Arbeit beschäftigt sich mit der Programmiersprache Python. Wir arbeiten mit der Python-Version 3.5 und in der Entwicklungsumgebung Jupyter 5.5.0. Im folgenden ist die Liste der verwendeten Bibliotheken aufgelistet.

- gensim==3.5.0
- graphviz==0.8.4
- gym==0.10.5
- h5py==2.7.1
- Keras==2.1.5
- matplotlib==2.2.0
- numpy==1.15.0
- numpydoc==0.7.0
- pandas==0.23.3
- pandoctfilters==1.4.2
- pickleshare==0.7.4
- Pillow==5.1.0
- pydot==1.2.4
- pydotplus==2.0.2

- scikit-learn==0.19.1
- scipy==1.0.0
- tensorboard==1.8.0
- tensorflow==1.8.0

Um die Installation zu erleichtern, liegt im Anhang die gekürzte Anforderungsdatei *Anforderungen/requirements_compr.txt* die mittels PIP [Stu08], ein Paketverwaltungssystem, ausführbar ist. Die Datei *Anforderungen/requirements_full.txt* enthält alle Bibliotheken, die auf dem System installiert sind.

B Beschreibung der Daten

Dieser Teil des Anhangs enthält weitere Informationen über die Datensätze aus dem Kapitel 3. Außerdem präsentieren wir die Teile des Programmiercodes für das Downloadsyste aus dem Abschnitt Twitter 3.2, den Code für die Generierung der Daten aus dem Abschnitt des Trainingsdatensatzes 3.2.1 und die Datenbereinigung- und Feature Engineering-Verfahren aus dem Abschnitt 3.2.2. Weiterhin beschreiben wir die Schritte der FI in Verbindung mit der Datenbereinigung.

Um einen weiteren Eindruck der Daten zu erhalten, zeigt die Abbildung B.1 die am häufigst gesetzten Hashtags der Twitterer. Die x-Achse repräsentiert die Hashtags in einer absteigenden Reihe, die y-Achse zeigt die Anzahl der Hashtags in Millionen und die Prozentangaben drückt die prozentuale Verteilung aus.

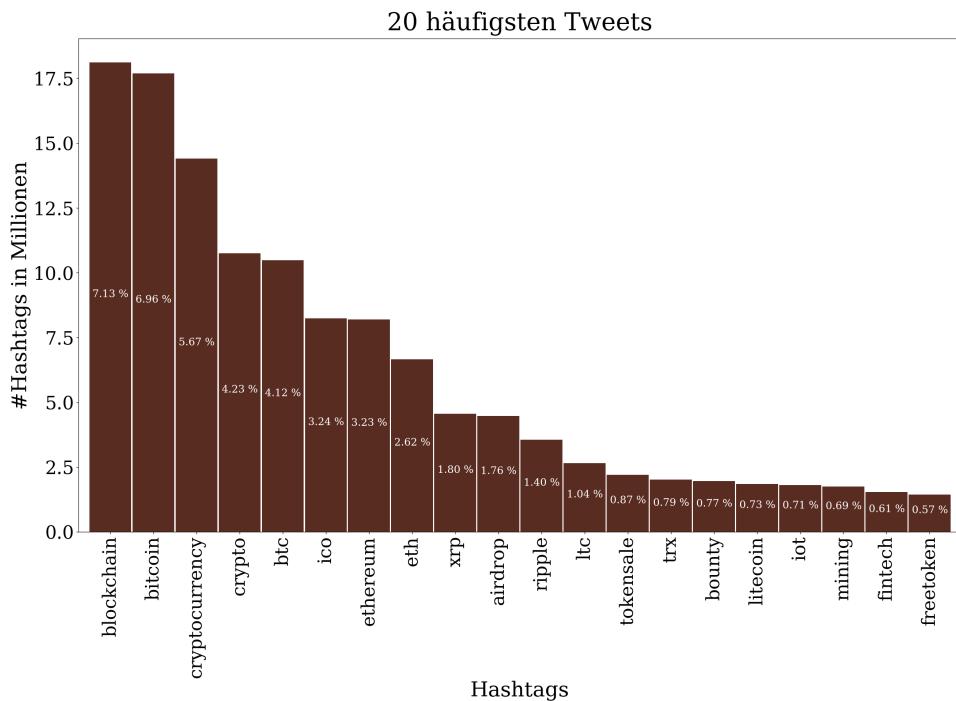


Abbildung B.1: Verteilung der Hashtags - Top 20. Auf der x-Achse listen wir die 20 häufigsten Hashtags absteigend auf. Die y-Achse zeigt die Häufigkeit der Hashtags über dem gesamten Datensatz in Millionen. Die prozentuale Angabe in den Balken spiegelt den Anteil der gesamten Hashtags wieder.

B.1 Twitter-Download-System

In diesem Teil erläutern wir wichtige Teile der PHP-Download-Software.

Im Listing B.1 sehen wir den kompletten Ablauf des Download-Skriptes. Zu Beginn definieren wir Hashtags, über die in Zeile 9 iteriert wird. Die Zeile 12 ist der Download aus Listing B.2. Zeile 15-18 benennt die Internas des assoziativen Arrays um. Die Zeilen 21-24 sind zuständig für das Abspeichern der Tweets. Unsere Überlegung ist, die heruntergeladenen Tweets nicht nur in eine Datei zu speichern, sondern mit maximal 15.000 Tweets aufgeteilt in mehrere Dateien zu speichern. Der Grund liegt in der Weiterverarbeitung, da wir davon ausgehen, dass am Ende des Projektes mehrere Gigabyte an Tweets heruntergeladen worden sind. Die Größe könnte den Arbeitsspeicher überfüllen. Übersteigt eine Datei die 15.000 Tweets archivieren wir diese in einem separaten Ordner.

```

1 <?php
2 // Searching for hashtags
3 $hashtags = array('#bitcoin', '#cryptocurrency');
4
5 // Initiate class
6 $twits = new PullTweets();
7 $countNewEntries = array();
8
9 // Iterate over hashtags
10 foreach ($hashtags as $hashtag) {
11     // Download hashtags
12     $twitsData = $twits->getTweets($hashtag);
13
14     // Clean tweets
15     $cleanTwitsData = array();
16     foreach ($twitsData['statuses'] as $twitDate) {
17         $cleanTwitsData[] = $twits->cutTweet($twitDate);
18     }
19
20     // Saving tweets to files
21     $pf = new ProcessFiles($args);
22     $pf->setHashTag($hashtag);
23     $pf->setTwitterData($cleanTwitsData);
24     $pf->removeExistTwitts();
25 }
26 ?>

```

Listing B.1: Dieses Code-Beispiel zeigt den Download der Tweets. Dabei wird über die Tweets in Zeile 3 iteriert mittels der Funktion *getTweets()*, die Tweets runtergeladen. Mittels Zeile 16 und 17 werden die Tweets bereinigt und die Zeilen 21-24 speichern die neuen Tweets.

Im folgenden Listing B.2 zeigen wir den Download der Tweets. Die Zeilen 10 und 11 konstruiert die Anfrage der letzten 100 Tweets in der erweiterten Version. Die Zeilen 14-18 stellen den eigentlichen Download dar. Zuerst wird die Twitter-API-Klasse initiiert, danach der obigen Anfrage übergeben und schlussendlich die Abfrage abgeschickt.

```

1 <?php
2 /*
3  * Download of the Last 100 Tweets
4  * $hashtag @string
5  *
6  * return JSON
7 */
8 function getTweets($hashtag) {
9     // Building the query
10    $getfield =
11        '?q=' . $hashtag . '&result_type=recent&count=100&tweet_mode=extended';
12
13    // Perform the request
14    $twitter = new TwitterAPIExchange($this->settings);
15    $data = $twitter
16        ->setGetfield($getfield)
17        ->buildOauth($this->url, $this->requestMethod)
18        ->performRequest();
19    return json_decode($data, true);
20 }
21 ?>

```

Listing B.2: Dieser Code zeigt den eigentlichen Download der Tweets.

Im unteren Listing B.3 sehen wir die eingestellten Cronjobs. *Crypto fast* sind die Hashtags, die alle zwei Minuten abgefragt werden, weil sonst mehr als 100 Tweets geschrieben worden wären und dadurch Tweets verloren gehen würden. 100 Tweets sind das Maximum einer Twitter-Abfrage. *Crypto* wird alle 8 Minuten abgefragt.

```

1 # Crypto fast
2 */2 * * * * /usr/bin/php /var/crons/sentiments/crypto_fast.php > /var/crons/
3   sentiments/logs/crypto_fast.log
4
5 # Crypto
6 */8 * * * * /usr/bin/php /var/crons/sentiments/crypto.php > /var/crons/sentiments
7   /logs/crypto.log

```

Listing B.3: Dieser Cronjob ruft alle zwei Minuten die Datei *crypto_fast.php* und alle 8 Minuten *crypto.php* auf.

Die serverseitig ausführbaren Dateien liegen im Anhang im Ordner *PHP/*.

Hashtags

Nach den folgenden Hashtags suchen wir mittels der Twitter-API.

- #cryptocurrency, #blockchain, #mining
- #bitcoin, #btc, #bitcointalk, #bitcoins
- #xrp, #ripple, #ETH, #ethereum, #ltc

B.2 Annotierte Tweets

In diesem Teil des Anhangs wird der Code erläutert, wie die Konvertierung von dem AMT-Datensatz zu unserem Lexikon stattfindet, siehe Listing B.4.

Nachdem wir alle Tweets laden, entfernen wir die überflüssigen Spalten, Zeile 5. Danach errechnen wir in den Zeilen 8-11 die Standardabweichung und den Durchschnitt. Die Zeile 17 fügt die errechneten Werte mit der zugehörigen ID zusammen. In den Zeilen 20-26 berechnen wir den Sentiment mit einem Threshold.

```

1 # Get all tweets
2 df = get_all_tweets()
3
4 # Remove not needed columns
5 df = df.drop(columns=['Answer.tweet_id', ..., 'SubmitTime'])
6
7 # Getting the std
8 grouped_id_std = df.groupby(['HITId']).std()
9
10 # Getting mean of tweet sentiment
11 grouped_id = df.groupby(['HITId']).mean()
12
13 # Getting the tweet
14 grouped_id_tweet = df.groupby(['HITId']).max()
15
16 # Merge frames
17 grouped_id = pd.concat([grouped_id, grouped_id_std, grouped_id_tweet], axis=1,
   sort=False)
18
19 # Simple sentiment
20 threshold = 0.15 # Threshold for neutral
21 conditions = [
22     (grouped_id['sentiment_mean'] < -threshold), # negative
23     (grouped_id['sentiment_mean'] > threshold), # positive
24     (grouped_id['sentiment_mean'] >= -threshold) & (grouped_id['sentiment_mean'] <=
   threshold)]
25 choices = [-1, 1, 0] # neutral
26 grouped_id['sentiment_simple'] = np.select(conditions, choices, default=0)
```

Listing B.4: Die Ergebnisse aus AMT werden mittels diesem Code in eine tabellarische Form umgewandelt, sodass die Verfahren diese verarbeiten können.

Die ausführbare Python-Datei ist im Anhang unter dem Pfad *AMT/init_data.py* zu finden.

Die Tabelle B.1 beschreibt die einzelnen Spalten des Trainingsdatensatzes aus Kapitel 3.2.1. Der vollständige Trainingsdatensatz befindet sich im Anhang *Data/Trainingsdatensatz/-training_data.csv*.

Tabelle B.1: Eine vollständige Beschreibung der Trainingsdaten generiert im Kapitel 3.2.1.

Spalte	Beschreibung
<i>sentiment_mean</i>	Die Durchschnittsbewertung der sieben Turkers.
<i>sentiment_std</i>	Die Standardabweichung der sieben Turkers.
<i>tweet</i>	Der Text des Tweetes.
<i>id</i>	Repräsentiert die AMT-ID.
<i>sentiment_simple</i>	Mittels der Spalte <i>sentiment_mean</i> nehmen wir eine Einteilung in die Klassen 0, -1 und 1 vor. Dies dient den maschinellen Lernverfahren als Target.

B.3 Datenbereinigung & Feature Engineering

In diesem Anhang zeigen wir Teile des Codes für die Datenbereinigung aus dem Abschnitt B.3.1 und die logischen Schritte der Feature Importance im Abschnitt B.3.2.

B.3.1 Datenbereinigung

Dieser Anhang enthält die wichtigsten Teile des Skriptes, die im Kapitel 3.2.2 für die Datenbereinigen verwenden werden. Das Listing B.5 zeigt den Teil der Funktion, die für die Bereinigung der Tweets zuständig ist. In Zeile 2-3 entfernen wir alle Retweets und die Zeilen 5-23 zeigen jeden einzelnen Bereinigungsschritt. Das System arbeitet mit einer konfigurierbaren Variable, die uns ermöglicht die Bereinigungsschritte ein und aus zu schalten. Der Abruf dieser Config ist in den Zeilen 2, 6, 10, 14, 18 und 22 zu sehen. Das vollständige Skript liegt im Anhang *Code/Cleaning/*.

```

1 # If it starts with RE remove the whole tweet
2 if self.config['preparingData']['removeRetweets'] and tweet['full_text'].startswith('RT '):
3     return False
4
5 # UPPERCASE to losercase
6 if self.config['preparingData']['UPPERCASElosercase']:
7     tweet['full_text'] = tweet['full_text'].lower()
8
9 # Remove URL
10 if self.config['preparingData']['removeURL']:

```

```

11     tweet['full_text'] = re.sub(r"(https|http):\/\/{2}[\d\w-]+(\. [\d\w-]+)
12     *(?::(?:\/[^s\/]*))?", "", tweet['full_text'])
13
14 # Replace user
15 if self.config['preparingData']['replaceOtherUser']:
16     tweet['full_text'] = re.sub(r"@[a-zA-Z0-9_]+", "", tweet['full_text'])
17
18 # Remove HTML
19 if self.config['preparingData']['removeHTMLTags']:
20     tweet['full_text'] = re.sub(r"<.*?>", "", tweet['full_text'])
21
22 # Remove hashtags
23 if self.config['preparingData']['removeHashtags']:
24     tweet['full_text'] = re.sub(r"(\$|#)[a-zA-Z]+ ?", "", tweet['full_text'])

```

Listing B.5: Dieser Code listet die einzelnen Bereinigungsschritte der Tweets auf.

B.3.2 Feature Importance

In diesem Teil besprechen wir den Weg des Ergebnisses aus dem Abschnitt 3.2.2. Eine unserer Überlegungen war die Wichtigkeit der Trennung bezogen auf die Features herauszufinden. Um diese Beurteilung vorzunehmen, verwendeten wir den RF aus dem Abschnitt C.2.1. Da der RF die Features zum Entscheiden nutzt, versucht er diese Features zu nehmen, die die höchsten Informationsgehalter besitzen. Als Nebenergebnis kann RF die sogenannte FI generieren. Dies ist im Endeffekt eine Liste des Informationsgehaltes der Features. Diese wollen wir möglichst sinnvoll wählen. Die empirische Verbesserung ist in der Tabelle B.2 zu sehen.

Die Parameter für RF sind im Listing B.6 zu sehen, die wir mit GridSearch [Ped11] ermitteln.

```

1 param_grid = {
2 'bootstrap': [True],
3 'max_features': ['auto'],
4 'min_samples_leaf': [1],
5 'min_samples_split': [2],
6 'n_estimators': [75],
7 'random_state': [42]
8 }

```

Listing B.6: Empfohlene Einstellung nach dem GridSearch.

Die Tabelle B.2 zeigt die 25 wichtigsten Features des Modells von oben nach unten absteigend des Informationsgehaltes des FIs sortiert. Die Spalte (1) repräsentiert die FI ohne Datenvorbereitung. Auffällig sind Eigennamen wie *ETH*, *Blockchain* oder *Bitcoin*, die im Zuge einer Blase leicht als wichtiges Feature festsetzen können. Weiterhin verwendet das Modell Stopwords wie *to* oder *the* zum Klassifizieren, die wir wegen des geringen Informationsgehaltes entfernen möchten.

Im ersten Schritt, hier Spalte (2), der Datenvorbereitung wandeln wir alle großen Buchstaben in kleine um. Dieser Schritt sorgt dafür, dass wir groß- und kleingeschriebene Wörter vereinheitlichen. Dadurch reduzieren wir die ungewollten Wörter, in der Tabelle fettgedruckt, von 13 auf 11. Spalte (3) zeigt den Effekt durch die Entfernung der Stopwörter. Dadurch erhalten wir nur noch Eigenwörter und HTML-Code, die unerwünscht sind. Als nächstes entfernen wir die Hashtags, zu sehen in der Spalte (4) und erhalten dadurch eine wesentlich geringere Anzahl an Eigennamen. In Spalte (5) und (6) entfernen wir die URLs und dem HTML-Code, Spalte (7) die Twitter-Nutzer und Spalte (8) die übrigen Eigennamen. Die dazugehörigen Abbildungen sind im Anhang *Feature Importance/* zu finden.

Tabelle B.2: In dieser Tabelle ist die Bereinigung mittels der Feature Importance aufgelistet. Die Spalten (1) bis (8) spiegeln die kumulierten Bereinigungsschritte der Datenbereinigung mittels der Feature Importance wieder. Spalte (1): *Keine Datenbereinigung*, Spalte (2): *Konvertierung in Kleinbuchstaben*, Spalte (3): *Entfernen der Stopwörter*, Spalte (4): *Entfernen der Hashtags*, Spalte (5) und (6): *Entfernen der URLs und dem HTML-Code*, Spalte (7): *Entfernen der Twitter-Nutzer* und Spalte (8): *Entfernen der restlichen Eigennamen*.

Feature Importance							
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
great	great	great	great	great	great	great	great
project	block-chain	project	project	project	project	project	good
good	project	ico	good	good	join	join	join
ICO	bitcoin	good	free	free	good	good	best
ETH	ico	block-chain	join	best	free	free	scam
br	scam	scam	best	join	best	team	free
and	eth	best	scam	team	scam	scam	fraud
Great	best	eth	team	scam	bonus	best	bad
Bitcoin	free	fraud	br	bonus	team	bonus	success
for	good	join	sucess	opport-unity	bad	opport-unity	bonus
free	br	free	bonus	success	opport-unity	success	fake
Block-chain	success	br	bad	bad	fraud	bad	opport-unity
the best	join	ethereum	crash	thanks	bitcoin	fraud	crash
Join	and	bitcoin	br https	fake	success	love	hate
Singapore	cyber-security	team	bitcoin	fraud	fake	future	future
best	team	bonus	successful	tokens	love	bitcoin	successful
team	bonus	thanks	opport-unity	bitcoin	crash	platform	friends
to	this	success	fraud	friends	help	thanks	sure
luck	the best	br https	future	luck	amazing	amazing	thanks
success	to	airdrop	thanks	amazing	thanks	singapore	love
future	fraud	crash	amazing	singapore	successful	fake	amazing
will	for	bad	scams	crash	platform	crash	help
block-chain	ethereum	ripple	platform	platform	luck	successful	ban
scam	xrp	platform	love	scams	singapore	win	scams
join	very	xrp	fake	love	friends	friends	fuck

B.4 Zeitreihe

Die folgende Liste beschreibt alle Spalten, die der Zeitreihendatensatz enthält. Der Teilname der Spalten *original* oder *edit* unterscheidet das originale Modell vom Angepassten. Endet ein Spaltenname mit *_log_returns* spiegelt die Spalte die log-Rendite wieder. Enthält ein Spaltenname *raw* meinen wir, dass die Wahrscheinlichkeiten der Modelle verwendet wurden anstatt der absoluten Stimmung (positiv, neutral oder negativ). *pos* und *neg* stehen für positiv und negativ, die in der jeweiligen Spalte kumuliert sind.

- Menge der Tweets mit *number_tweets* und *number_tweets_log_returns*.
- Kumulierte Länge der Tweets mit *commu_tweet_length* und *commu_tweet_length_log_returns*.
- Durchschnittliche Länge der Tweets mit *avg_len_tweets* und *avg_len_tweets_log_returns*.
- Menge der gesetzten Hashtags mit *commu_hashtags* und *commu_hashtags_log_returns*.
- Durchschnittliche Länge der Hashtags mit *avg_len_hashtags* und *avg_len_hashtags_log_returns*.
- Menge der Freunde der Twitterer mit *commu_friends* und *commu_friends_log_returns*
- Menge der durchschnittlichen Freunde der Twitterer mit *avg_num_friends* und *avg_num_friends_log_returns*
- Menge der Follower der Twitterer mit *commu_followers* und *commu_followers_log_returns*
- Sentiment mit vaderSentiment aus dem Abschnitt C.1.1 mit *vader* und *vader_log_returns*
- Sentiment mit sentimentr aus dem Abschnitt C.1.2 mit *sentimentr* und *sentimentr_log_returns*
- Sentiment mit RF aus dem Abschnitt C.2.2 mit *rf* und *rf_log_returns*
- Sentiment mit Support Vector Machines (SVM) aus dem Abschnitt C.2.1 mit *svm* und *svm_log_returns*
- Sentiment mit HDLTex aus dem Abschnitt 4.7.1 mit *hdltex_original_all*, *hdltex_original_pos*, *hdltex_original_neg*, *hdltex_original_all_raw*, *hdltex_original_pos_raw*, *hdltex_original_neg_raw*, *hdltex_original_all_log_returns*, *hdltex_original_pos_log_returns*, *hdltex_original_neg_log_returns*, *hdltex_original_all_raw_log_returns*, *hdltex_original_pos_raw_log_returns* und *hdltex_original_neg_raw_log_returns*.

- Sentiment mit CNNSC aus dem Abschnitt 4.7.2 mit *cnnsc_original_all*, *cnnsc_original_pos*, *cnnsc_original_neg*, *cnnsc_original_all_raw*, *cnnsc_original_pos_raw*, *cnnsc_original_neg_raw*, *cnnsc_original_all_log_returns*, *cnnsc_original_pos_log_returns*, *cnnsc_original_neg_log_returns*, *cnnsc_original_all_raw_log_returns*, *cnnsc_original_pos_raw_log_returns* und *cnnsc_original_neg_raw_log_returns*.
- *linear_time_trend*, *linear_time_trend_log_returns*, *volume*, *volume_log_returns*, *market_cap* und *market_cap_log_returns* zählen zu unseren ökonomischen Variablen.
- Bitcoin mit *btc* und *btc_log_returns*

B.5 Trainings- und Validierungsdatensatz

Der Python-Code B.7 illustriert die Aufteilung des Trainingsdatensatzes auf dem Abschnitt 3.2.1. Vor dem Split werden die Tweets nach aufbereitet, vgl. Abschnitt 3.2.2. Danach nutzen wir die *sklearn*-Bibliothek, um die Daten in eine Trainings- und Testmenge zu teilen. Dabei ist unsere Trainings- 80 % und Testmenge 20 % groß. Wir setzen den Seed auf 0.

```

1 # Cleans the data
2 tweet_list = preprocessing_own(df['tweet'].tolist(), preprocessing_config)
3
4 # Splits
5 X_train, X_test, y_train, y_test = train_test_split(
6 tweet_list, df['sentiment_simple'], test_size=0.20, random_state=0)
```

Listing B.7: Split des Trainings- und Testdatensatzes.

Die ausführbare Python-Datei ist im Anhang unter dem Pfad *Python/Split_Lexicon.py* zu finden.

C Vergleichsmethoden

In Deep Learning (DL) aus dem Kapitel 4 stellen wir Modelle vor, die wir mit etablierten Verfahren vergleichen wollen. Damit wir eine Intuition für die Güte eines Modells gewinnen, verwenden wir in diesem Anhang gängige Verfahren für das Bilden eines Sentiments. Begonnen wird mit erweiterten lexikalischen Verfahren, die bereits logische Operationen enthalten, siehe Abschnitt C.1. Danach verwenden wir zwei Verfahren aus dem Gebiet des maschinellen Lernens aus dem Abschnitt C.2. Beide Abschnitte schließen jeweils mit ihren Ergebnissen.

C.1 Sentiment Pakete

Einer der intuitivsten Ideen ein Sentiment über Texte zu bilden, ist über Wörterbücher. Im Sentimentbereich ist ein Wörterbuch eine Liste von möglichst allen und wichtigsten Wörtern, die in irgendeiner Art und Weise eine Bewertung des Inhaltes erlauben. Diese kann aus 0/1, positiv/negativ, eine Liste von 0 bis 9 oder sogar emotionale Wörter, wie *traurig* oder *glücklich*, bestehen.

Zu Beginn stellen wir die verschiedenen Verfahren vaderSentiment und sentimentr vor, die als Basis ein Wörterbuch verwenden, siehe die Abschnitte C.1.1 und C.1.2. Neben den Wörterbüchern verwenden die Verfahren weiterführende Logiken, die die Klassifikation verbessern. Nach der Beschreibung wenden wir die Verfahren auf unseren Testdatensatz aus dem Abschnitt 3.2.1 an und evaluieren die Ergebnisse. Der Fokus liegt auf der Anwendung der Methoden und deren Ergebnisse.

C.1.1 vaderSentiment

Valence Aware Dictionary for sEntiment Reasoning (VADER) [Hut14] verwendet ebenfalls ein Wörterbuch, das sie selbst mittels AMT generieren, vgl. Abschnitt 3.2.1. Dieses Wörterbuch verbinden sie mit Erweiterungen wie generalisierten Regeln, beispielsweise Verneinung. Hutto und Gilbert vergleichen ihre Ergebnisse mit elf unterschiedlichen Verfahren der gängigen Praxis. Sie erreichen mit einer Kombination von qualitativen und quantitativen Regeln ihren Goldstandard, der mittels 20 verschiedenen menschlichen Testern validiert wird. Außerdem betonen die Autoren, dass VADER nicht nur durch die Tester bestätigt wird, sondern sie auch übertrifft.

Zu Beginn erstellen die Autoren eine Liste von verschiedenen gut etablierten Wörterbüchern und lexikalischen Eigenschaften in Bezug zu Mikroblogs vor. Mit lexikalischen Eigenschaften sind ganze Wörter, Wortstämme oder Präfixe gemeint. Außerdem fügten sie den Listen eine volle Liste von Emojis bei, sodass sie nach der Recherche eine Liste von 9.000 verschiedenen lexikalischen Eigenschaften generierten. Als nächstes extrahierten sie mittels

unterschiedlichen Verfahren die wichtigsten lexikalischen Eigenschaften, genauer [Hut14, S. 220]. Danach ließen sie diese Eigenschaften mittels AMT von unabhängigen Menschen bewerten. Nach dieser Bewertung behalten sie die Eigenschaften, die nicht als neutral bewertet wurden, sodass noch 7.500 übrig bleiben. Mit diesem Wörterbuch klassifizierten sie als nächstes 400 Tweets nach negativ und positiv und lassen diese nochmals von Experten bewerten. Nach diesem aufwendigen Lernverfahren entstehen verschiedene Erkenntnisse, wie das Erkennen der Intensität eines Satzes, genauer [Hut14, S. 221]. Nach der Erkennung von Fehlklassifikationen und Implementierung der gewonnenen Heuristiken, simulieren sie den Fehler und versuchten dadurch die Klassifikation zu verbessern.

C.1.2 sentimentr

Tyler Rinker entwickelte für seine eigenen Bedürfnissen ein R-Paket *sentimentr* [Rin18], das die Stimmung eines Satzes erkennen soll. Er erklärt aus seiner Sicht, dass das Problem ein Trade-Off zwischen Geschwindigkeit und Genauigkeit ist. Bestimmt der Algorithmus schnell ein Sentiment, so ist er in der Regel ungenauer und umgedreht. Tyler versucht mit *sentimentr* die Problematik auszubalancieren. Genau wie VADER bedient sich *sentimentr* an generalisierten Regeln wie Verneinung, Wortverstärker, Abschwächungsartikel u. v. m.

C.1.3 Ergebnisse

In diesem Abschnitt untersuchen wir die Ergebnisse der Verfahren vaderSentiment und *sentimentr* aus den Abschnitten C.1.1 und C.1.2. Dabei ergibt sich eine neue Hypothese, dass vaderSentiment besonders gut abschneiden wird, weil wir bei der Erstellung des Trainingsdatensatzes im Abschnitt 3.2.1 die Tweets stark in positiv und negativ mittel vaderSentiment aufgeteilt.

Wie im Trainingsdatensatz wurde für vaderSentiment und *sentimentr* ein Threshold für die Klasse 0 festgelegt. Der Threshold wurde auf $\pm 0,05$ festgelegt, da die Ergebnisse von *sentimentr* generell kleiner sind. Wir sind uns bewusst, dass dieser Threshold ebenfalls eine Stellschraube ist darstellt, siehe Ausblick 6.

Wie erwartet schneidet vaderSentiment besser ab als *sentimentr*. Abbildung C.1 zeigt beide ROC-Kurven. Der Anteil der als korrekt positiv klassifizierten Tweets (TPR) und der fälschlich als negativ klassifizierten Tweets (FPR) ist beim vaderSentiment leicht höher. Beide Verfahren haben Schwierigkeiten neutrale Tweets zu erkennen. Visuell scheint vaderSentiment die Tweets besser zu klassifizieren.

Neben dem visuellen Eindruck bestätigen die Metriken aus der Tabelle C.1 ebenfalls die stärkere Klassifikation seitens *sentimentVader*. Dies untermauert unsere anfängliche Hypothese. Weiterhin fällt die starke Trennung zwischen positiv und negativ des Lexikons auf. Aus diesem Grund ist die Klasse 0 schwächer.

Die Klasse -1 repräsentiert die negativen Tweets, Klasse 0 die neutralen und Klasse 1 positiven. Weiterhin verwenden wir Prediction (Prec), Recall (Rec) und F1-Score (F1) als Messwert. Prec, dt. Genauigkeit, spiegelt die von der Klasse richtig klassifizierten Objekte der Zielklasse wieder. Rec, dt. Trefferquote, bezeichnet den Wert der überhaupt klassifizierten Objekte der Zielklasse.

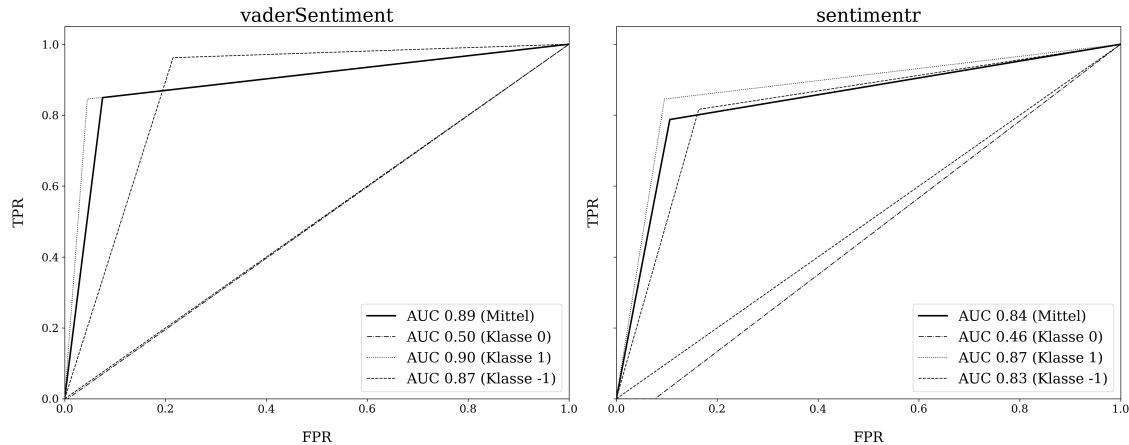


Abbildung C.1: ROC-Charts der lexikalischen Methoden. Der linke ROC-Chart zeigt das Ergebnis vom vaderSentiment und rechts sentimentr.

Tabelle C.1: *Lexikalische Verfahren*: Die Zeilen 1 und 2 stellen die Messergebnisse der Verfahren vaderSentiment und sentimentr dar.

Methoden	Klasse -1			Klasse 1			Insgesamt	
	Prec	Rec	F1	Prec	Rec	F1	F1	Acc
vaderSentiment	0,77	0,96	0,85	0,95	0,85	0,90	0,83	0,85
sentimentr	0,79	0,82	0,80	0,91	0,85	0,87	0,80	0,79

C.2 Maschinelle Lernverfahren

Neben den lexikalischen Methoden aus dem Abschnitt C.1 existieren die maschinellen Methoden. Diese Art von Methoden versuchen mit statistisch mathematischen Fundamenten Muster in Daten zu finden. Dabei wird der Datensatz nicht auswendig gelernt, sondern es besteht die Möglichkeit die erkannten Muster auf weitere Daten anzuwenden.

In diesem Kapitel ziehen wir die zwei Verfahren Random Forest und Support Vector Machines für das Bilden eines Sentiment zum Vergleich heran, siehe die Abschnitte C.2.1 und C.2.2. Gegen Ende des Kapitels präsentieren wir die Ergebnisse der maschinellen Methoden.

C.2.1 Random Forest

Das Klassifizierungsverfahren RF siedelt sich im Themenfeld des maschinellen Lernens an. RF verwendet sogenannte Entscheidungsbäume [Qui85], die 1985 von J. R. Quinlan vorgestellt wurden. Ein Entscheidungsbaum entscheidet je nach Regel welche Klassenzugehörigkeit ein Objekt zugeordnet werden kann. Abbildung C.2 zeigt zwei Entscheidungsbäume, die jeweils versuchen einen Mensch die Klasse Kind und Erwachsener anhand der Körpergröße zuzuordnen. Baum 1 klassifiziert einen Menschen als Erwachsenen, wenn er $\geq 100cm$ ist, wohingegen Baum 2 ihn schon ab $\geq 95cm$ als Erwachsenen einstuft. Werden die Bäume

mit Daten befüllt, erhalten wir unterschiedliche Ergebnisse, die sich einem Mehrheitsvotum stellen müssen. Ist ein Mensch beispielsweise 80 cm groß, würde der Baum 1 und Baum 2 als Kind klassifizieren. Diese beiden Entscheidungen werden danach einem Mehrheitsvotum unterzogen, das in diesem Fall ebenfalls den Menschen als Kind klassifizieren würde.

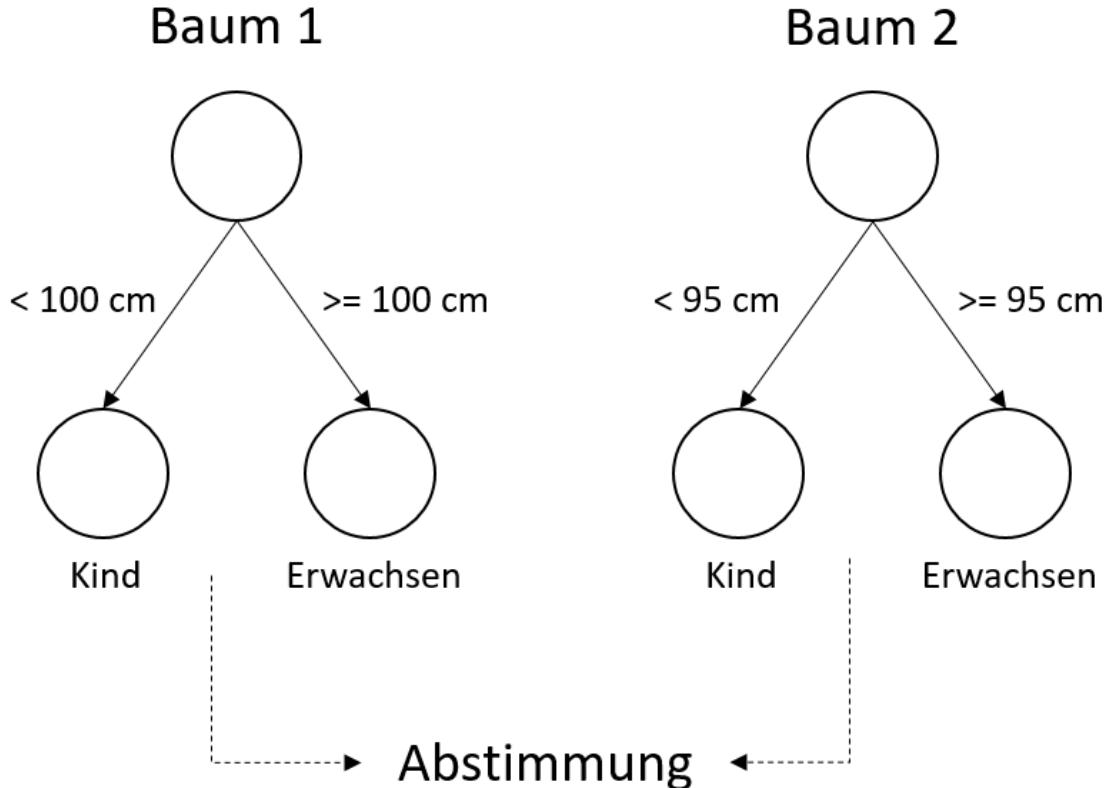


Abbildung C.2: Beispiel für einen Random Forest. In diesem Fall besteht der RF aus zwei Entscheidungsbäumen, die jeweils versuchen einen Menschen anhand der Körpergröße in die Klassen Kind oder Erwachsener einzuteilen. Der Threshold für Baum 1 liegt bei 100 cm und für Baum 2 bei 95 cm. Nachdem die beiden Bäume eine Entscheidung getroffen haben, werden diese einen Mehrheitsvotum unterzogen.

Die einfache Darstellung von RF ist ein großer Vorteil. Außerdem besteht die Möglichkeit die Berechnung der Klassen auf mehrere Prozesse gleichzeitig zu verteilen, da die Entscheidungsbäume ihre Berechnungen unabhängig voneinander durchführen. Neben den offensichtlich positiven Eigenschaften, haben die Autoren *Sentiment Mining of Movie Reviews using Random Forest with Tuned Hyperparameters* [Par14, S. 6] bewiesen, dass RFs gute Klassifikatoren im Bereich des Textminings sind.

C.2.2 Support Vector Machines

1995 entwickelten Corinna Cortes et al. das Klassifikationsverfahren Support Vector Machines (SVM) [Cor95]. SVM repräsentiert Objekte in einem Vektorraum, der gewöhnlich mehrdimensional ist. Im nächsten Schritt versucht SVM die Objekte mittels einer Hyperebene zu trennen. Die Herausforderung ist nicht nur ein Modell zum Trennen der Objekte zu finden, sondern diese so gut wie möglich zu trennen. Abbildung C.3 zeigt das Dilemma der Hyperebene. Die gestrichelten Hyperebenen trennen die Menge schlechter als die durchgezogene. Der maximale Abstand zu den Punkten, hier die zwei grauen orthogonalen Linien der Hyperebene, werden als Margin bezeichnet. Zu beachten gilt, dass nicht die gesamte Menge der Objekte für die Bildung der Hyperebene notwendig ist, dies ergäbe einen extremen Rechenaufwand. In unserem Beispiel werden jeweils nur ein Punkt zur Bestimmung der Hyperebene verwendet. Diese werden als Support Vektoren bezeichnet.

SVM verwendet einen sogenannten Kernel-Trick [Hof06, S. 10], der im Gegensatz zu RF aus dem Abschnitt C.2.1 sehr rechenintensiv ist. Dafür ist SVM in der Lage mit wenigen Objekten eine solide Klassifizierung zu erreichen. Neethu und Rajasree vergleichen in ihrem Paper *Sentiment Analysis in Twitter using Machine Learning Techniques* [Nee13] verschiedene maschinelle Methoden, auch SVM. Dabei stellt sich heraus, dass alle maschinellen Methoden ähnlich gut abschneiden.

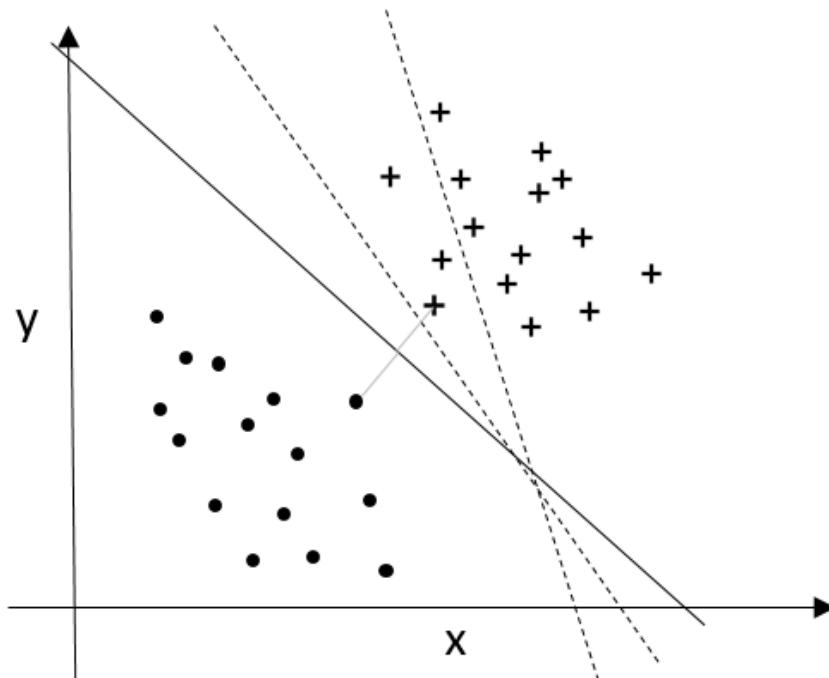


Abbildung C.3: Beispiel für ein Support Vector Machine Klassifikation. Die durchgezogene Hyperebene trennt die zwei Klassen mit einem maximalen Margin, wohingegen die gestrichelten Hyperebenen sie trennen, aber nicht perfekt. Die Objekte, die die Hyperebene bilden, werden Support Vektoren genannt.

C.2.3 Ergebnisse

In diesem Teil diskutieren wir die Ergebnisse der maschinellen Verfahren RF und SVM und wie wir den Status erreichten.

Eine Herausforderung war die optimalen Parameter für unseren Trainingsdatensatz zu finden, wozu wir verschiedene Methoden untersuchten. Dabei stellt sich *GridSearchCV* [Ped11] als solide raus.

Nachdem wir *GridSearchCV* mit RF laufen ließen, erhalten wir die folgenden Parameter für die optimalste Einstellung.

- bootstrap: True
- max_features: Auto
- min_samples_leaf: 2
- min_samples_split: 2
- n_estimators: 100

Das selbe Vorgehen führen wir für die SVMs.

- kernel: rbf
- gamma: 0,0001
- C: 1000

Den wichtigen Teil des Codes für die Generierung des Modells von RF beschreiben wir im Anhang C.2.4. Der Code von SVM ist äquivalent.

In Abbildung C.4 fällt auf, dass sowohl RF und SVM ähnlich gut in der Klassifizierung sind. RF klassifiziert die Klasse -1 schlechter dafür die Klasse 1 besser als SVM. Im Schnitt kommen die beiden Verfahren auf einen ähnlichen Durchschnitts-AUC.

Tabelle C.2: *Maschinelle Lernverfahren:* Die Zeilen 1 und 2 repräsentieren die Messergebnisse der Verfahren RF und SVM. *Lexikalische Verfahren:* Die Zeilen 3 und 4 (grau) stellen die Messergebnisse für vaderSentiment und sentimentr dar.

Methoden	Klasse -1			Klasse 1			Insgesamt	
	Prec	Rec	F1	Prec	Rec	F1	F1	Acc
RandomForest	0,80	0,68	0,73	0,88	0,85	0,86	0,76	0,73
SVM	0,74	0,84	0,79	0,83	0,83	0,83	0,77	0,79
vaderSentiment	0,77	0,96	0,85	0,95	0,85	0,90	0,83	0,85
sentimentr	0,79	0,82	0,80	0,91	0,85	0,87	0,80	0,79

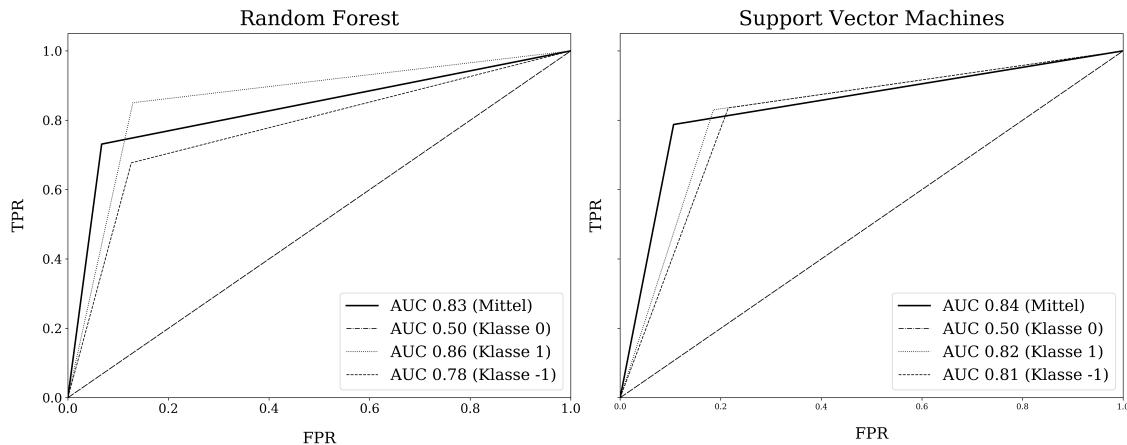


Abbildung C.4: ROC-Charts der maschinellen Lernmethoden. Der linke ROC-Chart zeigt das Ergebnis von Random Forest und rechts von den Support Vector Machines.

C.2.4 Code

In diesem Anhang verdeutlichen wir den Ablauf des Skriptes der maschinellen Lernverfahren. Es gilt zu beachten, dass wir nur den RF als Beispiel heranziehen, SVM jedoch ein ähnliches Prinzip verfolgt.

Im Code C.1 setzen wir in den Zeilen 1-18 die Parameter für die Funktionen. In den Zeilen 21-26 wandeln wir die Daten in eine One-Hot-Kodierung um, danach wird die Trainings- und Testmenge generiert. Die Zeilen 36-45 illustrieren die Methodik, wie wir die optimalsten Parameter generieren.

```

1 # Parameter for GridSearch
2 cross_validation = 10
3
4 # Parameters for Vec
5 min_df = 1
6 ngram_range = (1, 2)
7 lowercase=True
8 stop_words= 'english' # None or 'english'
9
10 # Parameters for RF Testing
11 param_grid = {
12     'bootstrap': [True],
13     'max_features': ['auto'],
14     'min_samples_leaf': [2, 3, 4],
15     'min_samples_split': [2, 3, 4],
16     'n_estimators': [95, 100, 105],
17     'random_state': [42]
18 }
19
20 # One-Hot Codierung Input
21 tweet_list = preprocessing_own(df['tweet'].tolist(), preprocessing_config)

```

```

22 vectorizer = CountVectorizer(min_df=min_df, ngram_range=ngram_range, lowercase=
23     lowercase, stop_words=stop_words)
24 X = vectorizer.fit_transform(tweet_list).toarray()
25
26 # One-Hot Codierung Output
27 y = label_binarize(df['sentiment_simple'], classes=[0, 1, -1])
28
29 # Number of classes for output
30 n_classes = y.shape[1]
31
32 # Split
33 X_train, X_test, y_train, y_test = train_test_split(
34     X, y, test_size=0.20, random_state=0)
35
36 # Create a based model
37 rf = RandomForestClassifier()
38 # Instantiate the grid search model
39 grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
40     cv = cross_validation, n_jobs = -1, verbose = 1)
41
42 # Fitting
43 grid_search.fit(X_train, y_train)
44
45 # Best parameters
46 grid_search.best_params_

```

Listing C.1: Das vollständige Skript für das Training des RFs. Nach der Vorbereitung findet das eigentlich Training in der Zeile 42 statt.

Der vollständig ausführbare Code liegt im Anhang *Code/RandomForest.py* und *Code/SVM.py*

D Deep Learning

In diesem Anhang reichen wir die Thesis um weitere Informationen des Bereiches Deep Learning an. Zu Beginn beschreiben wir die Modelle aus den Abschnitten KNN [4.7.2](#) und CNN [4.7.2](#) mit Code und Abbildungsexporten aus dem Tensorboard genauer. Weiterhin liefern wir im Abschnitt Ergebnisse [D.3](#) detailliertere Abbildung.

D.1 HDLTex

In diesem Anhang liefern wir weitere Informationen zum HDLTex-Modell aus dem Kapitel [4.7.1](#).

Wir zeigen im Listing [D.1](#) den Code für das originale HDLTex-Modell. Zu finden sind der Code inklusive das Erstellen der 1.000 Modelle im Anhang unter *Code/Building Models/KNN - HDLTex - Original.html* (.html, .py und .ipynb).

```
1 # Learning rate
2 learning_rate = 0.001
3 # Number of outputs variants
4 output_number = 3
5 # Number of neurons for the hidden layers
6 number_neurons = 1028
7 # number of epochs
8 number_epochs = 25
9 # 50 \% Dropout
10 drop_out = 0.5
11 # Activation function
12 activation_function = 'relu'
13 # Loss function
14 loss_function = 'categorical_crossentropy'
15
16 # (23704,) size of the input
17 shape = (X.shape[1],)
18
19 # Modell initialisieren
20 model = Sequential()
21 # 32 number of neurons for the input layer
22 model.add(Dense(32, input_shape = shape, activation=activation_function))
23 # Our first hidden layer with 1024 neurons
24 model.add(Dense(number_neurons, activation=activation_function))
25 model.add(Dropout(drop_out))
26 model.add(Dense(number_neurons, activation=activation_function))
27 model.add(Dropout(drop_out))
28 model.add(Dense(number_neurons, activation=activation_function))
```

```

29 model.add(Dropout(drop_out))
30 model.add(Dense(number_neurons, activation=activation_function))
31 model.add(Dropout(drop_out))
32 model.add(Dense(number_neurons, activation=activation_function))
33 model.add(Dropout(drop_out))
34 model.add(Dense(number_neurons, activation=activation_function))
35 model.add(Dropout(drop_out))
36 model.add(Dense(number_neurons, activation=activation_function))
37 model.add(Dropout(drop_out))
38 model.add(Dense(number_neurons, activation=activation_function))
39 model.add(Dropout(drop_out))
40 model.add(Dense(output_number, activation=activation_function))
41 model.compile(loss=loss_function, optimizer=RMSprop(lr=learning_rate), metrics=['
  acc'])
42
43 # Training and getting the training history (Accuracy)
44 history = model.fit(X_train, y_train, epochs=number_epochs, verbose=0,
  validation_data=(X_test, y_test), batch_size=128)

```

Listing D.1: Der Code zeigt detailliert das originale HDLTex-Modell.

Das Listing D.2 zeigt unser angepasstes HDLTex-Modell aus dem Abschnitt 4.7.1. Weiterhin ist der Code inklusive das Erstellen der 1.000 Modelle im Anhang unter *Code/Building Models/KNN - HDLTex - Edit.html* (.html, .py und .ipynb) zu finden.

```

1 # Learning rate
2 learning_rate = 0.001
3 # Number of outputs variants
4 output_number = 3
5 # number of epochs
6 number_epochs = 15
7 # 50 % Dropout
8 drop_out = 0.65
9 # Activation function
10 activation_function = 'relu'
11 # Loss function
12 loss_function = 'categorical_crossentropy'
13
14 # (23704,) size of the input
15 shape = (X.shape[1],)
16
17 model = Sequential()
18 model.add(Dense(32, input_shape = shape, activation=activation_function))
19 model.add(Dense(128, activation=activation_function))
20 model.add(Dropout(drop_out))
21 model.add(Dense(256, activation=activation_function))
22 model.add(Dropout(drop_out))
23 model.add(Dense(512, kernel_regularizer=regularizers.l2(0.01), activation=
  activation_function))
24 model.add(Dropout(drop_out))

```

```
25 model.add(Dense(1024, kernel_regularizer=regularizers.l2(0.015), activation=
activation_function))
26 model.add(Dropout(drop_out))
27 model.add(Dense(512, kernel_regularizer=regularizers.l2(0.01), activation=
activation_function))
28 model.add(Dropout(drop_out))
29 model.add(Dense(256, activation=activation_function))
30 model.add(Dropout(drop_out))
31 model.add(Dense(128, activation=activation_function))
32 model.add(Dropout(drop_out))
33 model.add(Dense(output_number, activation=activation_function))
34 model.compile(loss=loss_function, optimizer=RMSprop(lr=learning_rate), metrics=['
acc'])
35
36 # Training
37 history = model.fit(X_train, y_train, epochs=number_epochs, verbose=0,
validation_data=(X_test, y_test), batch_size=128)
```

Listing D.2: Der Code zeigt detailliert das angepasste HDLTex-Modell.

Die Abbildungen [D.1](#) und [D.2](#) zeigen die zwei Modelle durch das Tensorboard [[Ten15](#)] visualisiert.

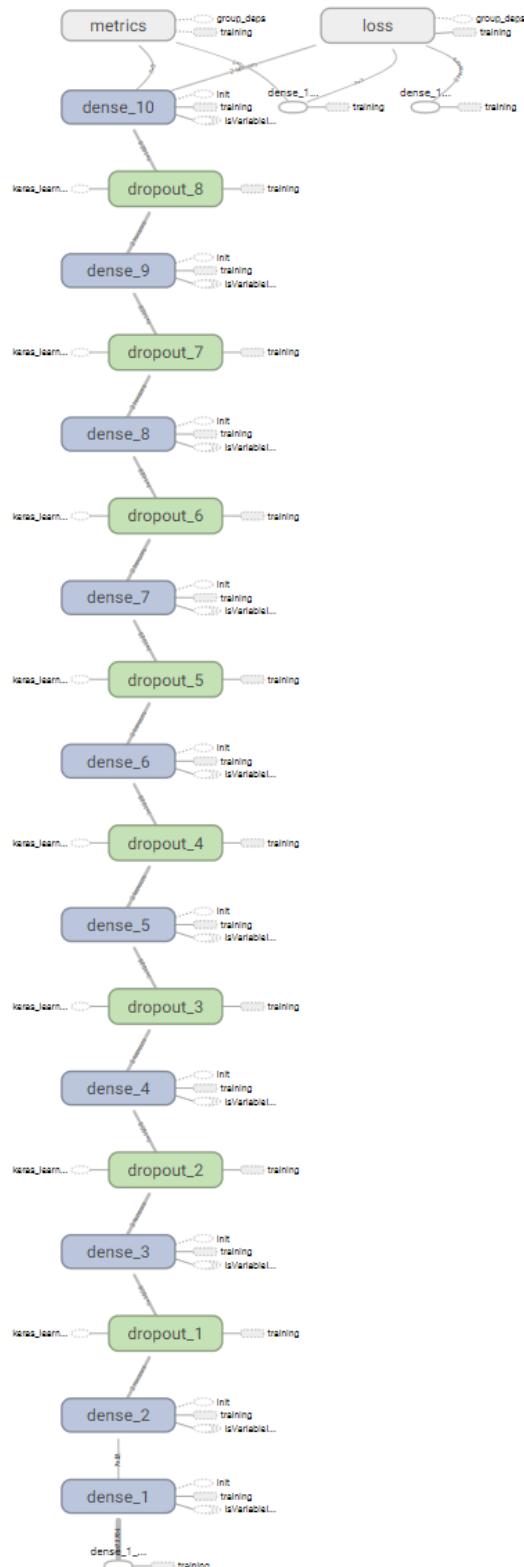


Abbildung D.1: Modellvisualisierung vom HDLTex mittels Tensorboard. *Dense 1* repräsentiert den Input-Layer und danach werden die Daten durch 9 KNNs und 8 Dropouts propagiert. *Dense 10* stellt den Output-Layer dar. Die genaue Konfiguration ist in Listing D.1 zu sehen.

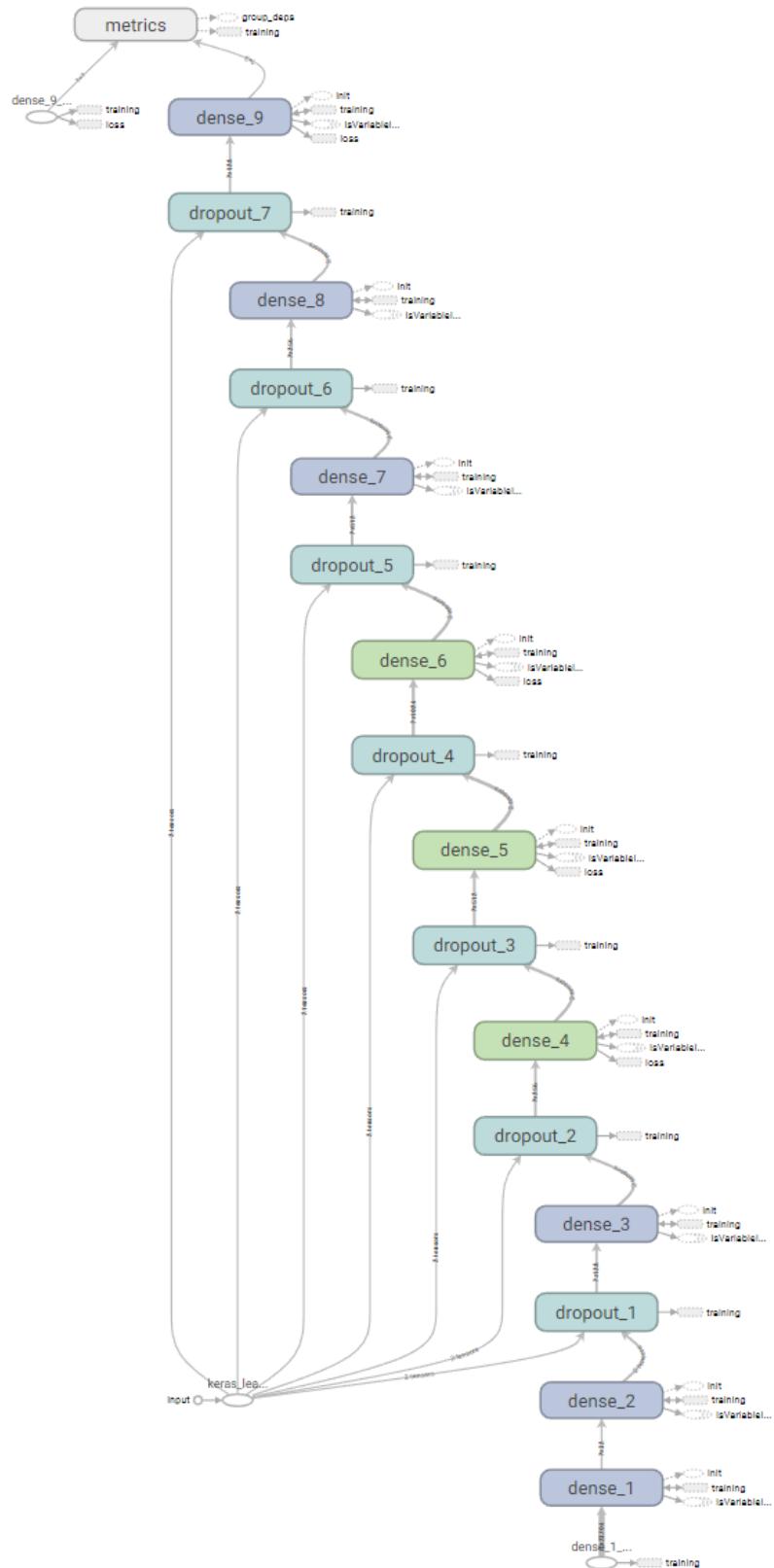


Abbildung D.2: Modellvisualisierung vom angepassten HDLTex mittels Tensorboard. *Dense 1* repräsentiert den Input-Layer und danach werden die Daten durch 9 KNNs und 8 Dropouts propagiert. *Dense 10* stellt den Output-Layer dar. Die genaue Konfiguration ist in Listing D.2 zu sehen.

D.2 CNNSC

In diesem Anhang liefern wir weitere Informationen zum originalen und angepassten CNNSC-Modell.

Im Listing D.3 ist der Code für das Erstellen des CNNSC-Modells zu sehen. Weiterhin ist der Code inklusive das Erstellen der 1000 Modelle im Anhang unter *Code/Building Models/CNN - CNNSC - Original.html* (.html, .py und .ipynb) zu finden.

```

1 # Learning rate
2 learning_rate = 0.001
3 # Number of outputs variants
4 output_number = 3
5 # Number of kernels
6 number_filter = 100
7 # Number of epochs
8 number_epochs = 25
9 # 50 % dropout
10 drop_out = 0.50
11 # Activation function
12 activation_function = 'relu'
13 # Loss function
14 loss_function = 'categorical_crossentropy'
15 # Regularizer
16 regularizer = 3.00
17
18 # Create the empty model
19 model = Sequential()
20 # Embedding layer
21 model.add(Embedding(vocab_size, 300, weights=[embedding_matrix], input_length=
   max_length))
22 # CNN
23 model.add(Conv1D(number_filter, 3, border_mode='same', kernel_regularizer=
   regularizers.l2(regularizer), activation=activation_function))
24 model.add(Dropout(drop_out))
25 model.add(MaxPooling1D(pool_size=2))
26 model.add(Flatten())
27 model.add(Dense(output_number, activation='softmax'))
28 # Compile
29 model.compile(loss=loss_function, optimizer=Adam(lr=learning_rate), metrics=['acc
   '])
30
31 # Training
32 history = model.fit(X_train, y_train, epochs=number_epochs, verbose=0,
   validation_data=(X_test, y_test), batch_size=128)

```

Listing D.3: Der Code zeigt detailliert das originale CNNSC-Modell.

Im Listing D.4 zeigt unser angepasstes CNNSC-Modell aus dem Abschnitt 4.7.2. Weiterhin ist der Code inklusive das Erstellen der 1000 Modelle im Anhang unter *Code/Building Models/CNN - CNNSC - Edit.html* (.html, .py und .ipynb) zu finden.

```

1 # Learning rate
2 learning_rate = 0.001
3 # Number of outputs variants
4 output_number = 3
5 # Number of kernels
6 number_filter = 128
7 # Number of epochs
8 number_epochs = 20
9 # 75 % dropout
10 drop_out = 0.75
11 # Activation function
12 activation_function = 'relu'
13 # Loss function
14 loss_function = 'categorical_crossentropy'
15 # Regularizer
16 regularizer = 0.00
17
18 # Create the empty model
19 model = Sequential()
20 # Embedding layer
21 model.add(Embedding(vocab_size, vector_size, weights=[embedding_matrix],
   input_length=max_length))
22 # CNN
23 model.add(Conv1D(number_filter, 1, border_mode='same', kernel_regularizer=
   regularizers.l2(regularizer), activation=activation_function))
24 model.add(Dropout(drop_out))
25 model.add(MaxPooling1D(pool_size=3))
26 model.add(Flatten())
27 model.add(Dense(output_number, activation='softmax'))
28 # Compile
29 model.compile(loss=loss_function, optimizer=Adam(lr=learning_rate), metrics=['acc'])
30
31 # Training
32 history = model.fit(X_train, y_train, epochs=number_epochs, verbose=0,
   validation_data=(X_test, y_test), batch_size=128)

```

Listing D.4: Der Code zeigt detailliert das angepasste CNNSC-Modell.

Die Abbildungen D.3 und D.4 zeigen die zwei Modelle durch das Tensorboard [Ten15] visualisiert.

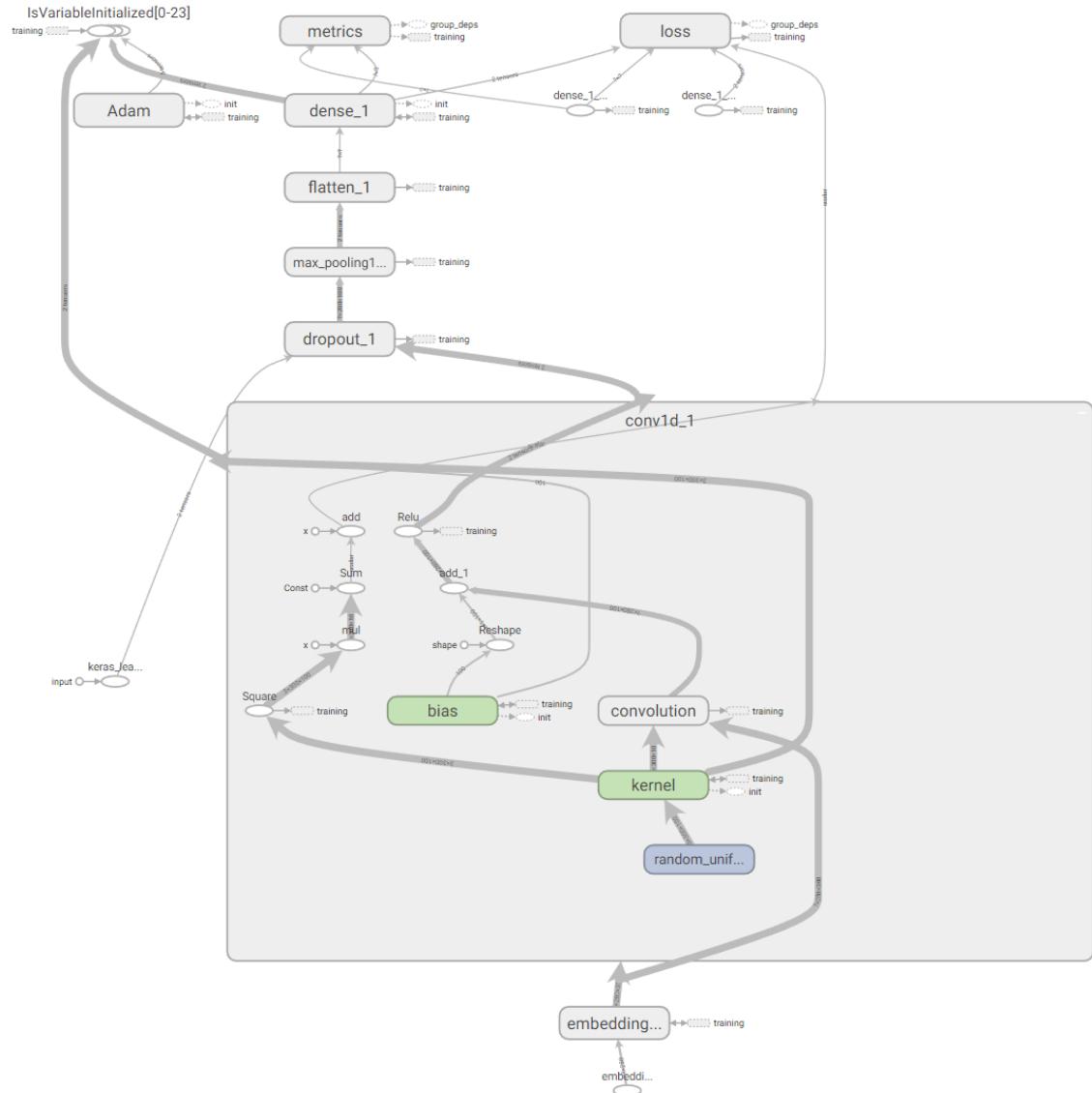


Abbildung D.3: Modellvisualisierung vom CNNSC mittels Tensorboard. Der Input-Layer ist mittels dem *Embedding-Layer* realisiert. Die graue Box stellt eine detaillierte Ansicht des ersten CNN-Layers dar. Mit *Dense 1* ist der Output-Layer gemeint. Die genaue Konfiguration ist im Listing D.3 zu finden.

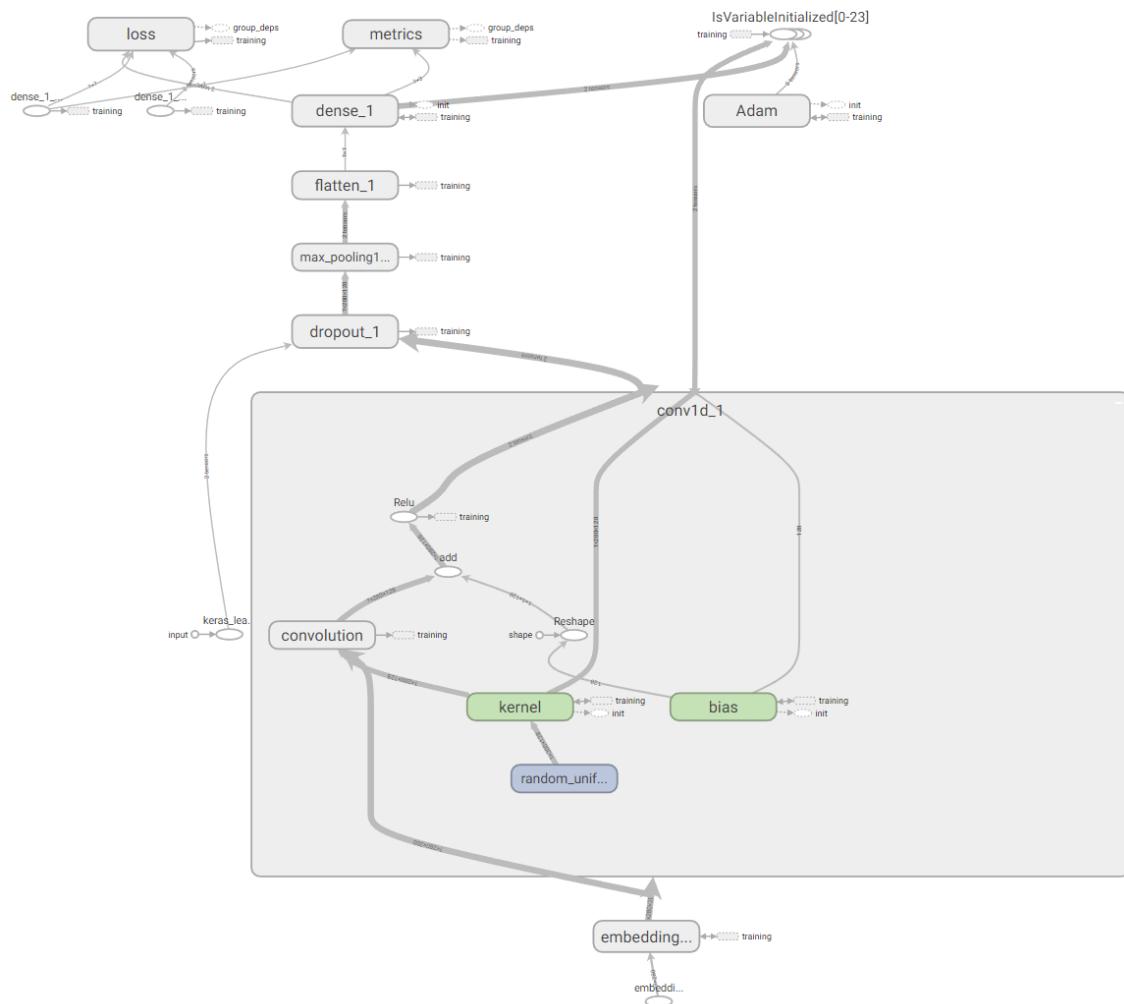


Abbildung D.4: Modellvisualisierung vom angepassten CNNSC mittels Tensorboard. Der Input-Layer ist mittels des *Embedding-Layers* realisiert. Die graue Box stellt eine detaillierte Ansicht des ersten CNN-Layers dar. Mit *Dense 1* ist der Output-Layer gemeint. Die genaue Konfiguration ist im Listing D.4 zu finden.

D.3 Ergebnisse

Dieser Teil des Anhangs beschreibt die Ergebnisse der originalen und angepassten HDLTex- und CNNSC-Modelle aus dem Abschnitt 4.7.

D.3.1 HDLTex

In diesem Anhang liefern wir zusätzliche Informationen zu den Ergebnissen aus dem Abschnitt 4.7.1.

Beginnend plotten wir eine detaillierte und größere Version des Genauigkeitsverlaufs, siehe Abbildungen D.5 und D.6. Die y-Achse steht für die Genauigkeit und die x-Achse die Anzahl der Epochen während dem Trainingsprozess. Außerdem sehen wir die Konfidenzbänder von 95 %. Wie schon im Hauptkapitel beschrieben fallen uns zwei Auffälligkeiten beim Vergleich beider Plots auf. Erstens erreichen wir eine leicht höhere Genauigkeit mit unserem Modell und zweitens sind unsere Konfidenzbänder schmäler.

Der dritte Plot D.7 zeigt die Genauigkeit der Modelle als Boxplots. Links das HDLTex- und rechts das angepasste Modell. Wie im Ergebnisabschnitt 4.7.1 beschreiben, können wir eine signifikante Verbesserung erkennen.

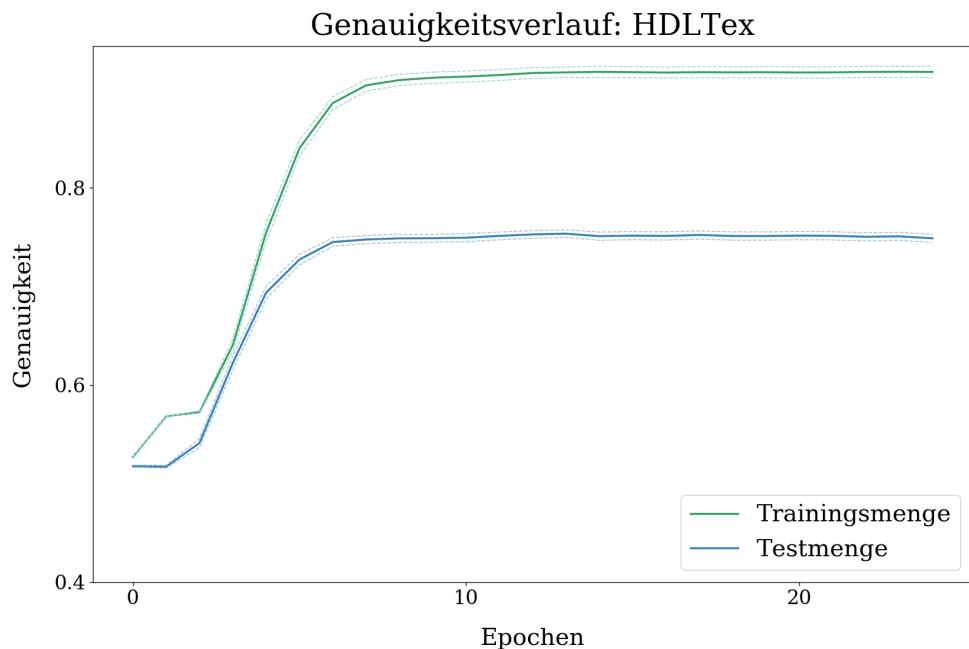


Abbildung D.5: HDLTex: Genauigkeitsverlauf. Die y-Achse repräsentiert die Genauigkeit gegen die Anzahl der Epochen des Trainings (x-Achse). In grün ist die Trainings- und blau die Testmenge zu sehen. Der Plot zeigt eine Vergrößerung der Ansicht des Ergebnisabschnittes 4.12.

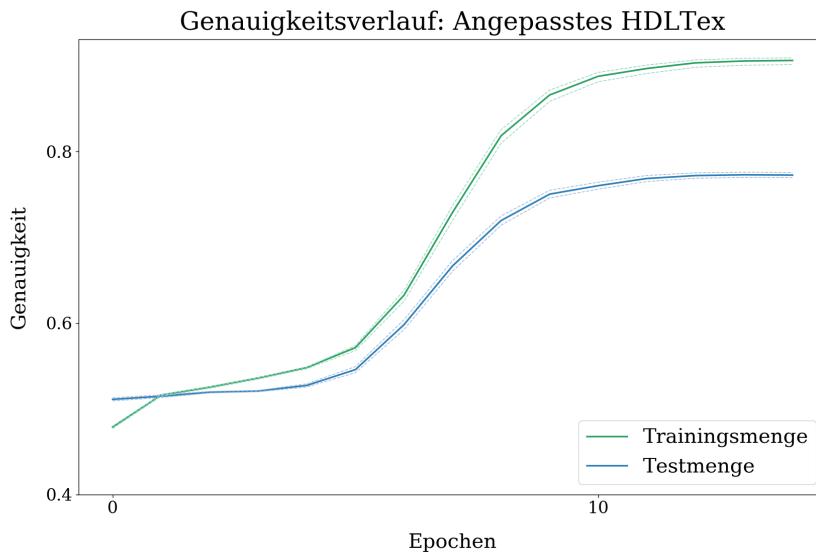


Abbildung D.6: Angepasstes HDLTex: Genauigkeitsverlauf. Die y-Achse repräsentiert die Genauigkeit die Anzahl der Epochen des Trainings (x-Achse). In grün ist die Trainings- und blau die Testmenge zu sehen. Der Plot zeigt eine Vergrößerung der Ansicht des Ergebniskapitels 4.12.

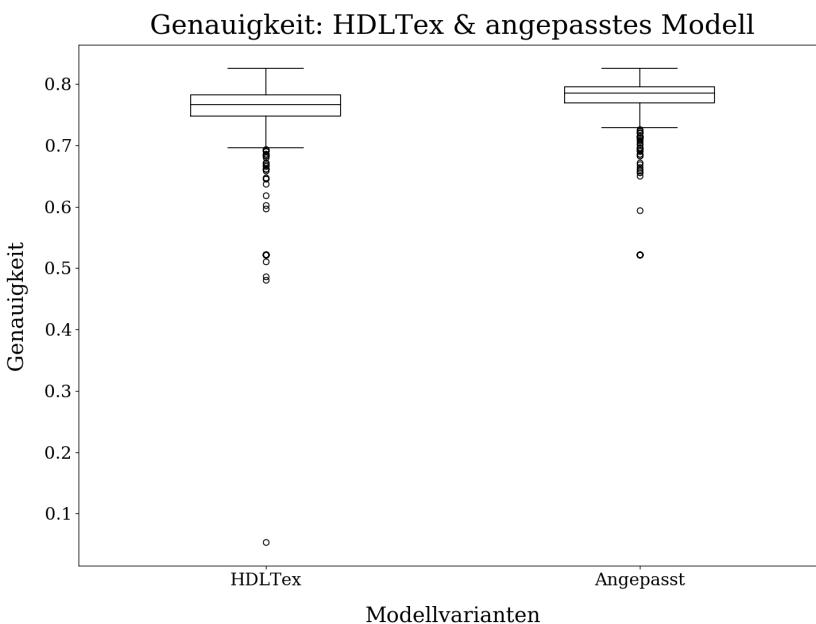


Abbildung D.7: Diese Abbildung stellt die Ergebnisse, hier die Genauigkeit, des HDLTex-Modells dem angepassten Modell gegenüber. Der linke Boxplot zeigt die Verteilung für das HDLTex-Modell und rechts unser angepasstes Modell über 1.000 Modelle an.

D.3.2 CNNSC

In diesem Kapitel liefern wir zusätzliche Informationen zu den Ergebnissen aus dem Abschnitt [4.7.2](#).

Beginnend plotten wir eine detaillierte und größere Version des Genauigkeitsverlaufs, siehe Abbildungen [D.8](#) und [D.9](#). Die y-Achse steht für die Genauigkeit und die x-Achse die Anzahl der Epochen während dem Trainingsprozess. Außerdem sehen wir Konfidenzbänder von 95 %. Wie schon im Hauptkapitel beschrieben fallen uns drei Auffälligkeiten beim Vergleich beider Plots auf. Erstens erreichen wir eine leicht höhere Genauigkeit mit unserem Modell, zweitens weisen unsere Konfidenzbänder eine niedrigere Varianz auf und drittens unsere Trainingsphase ist kürzer.

Der dritte Plot [D.10](#) zeigt die Genauigkeit der Modelle als Boxplots. Links das CNNSC- und rechts das angepasste Modell. Wie im Ergebnisabschnitt [4.7.2](#) beschreiben, können wir eine signifikante Verbesserung erkennen. Im Gegensatz zu den Boxplots aus den HDLTex-Modellen aus dem Abschnitt [D.7](#), können wir kein extrem schwaches Modell beobachten. Deswegen gilt in Abbildung [D.10](#) die y-Achse wegen dem kleinen Bereich zu beachten, den wir zwecks der besseren Unterteilung nicht ändern.

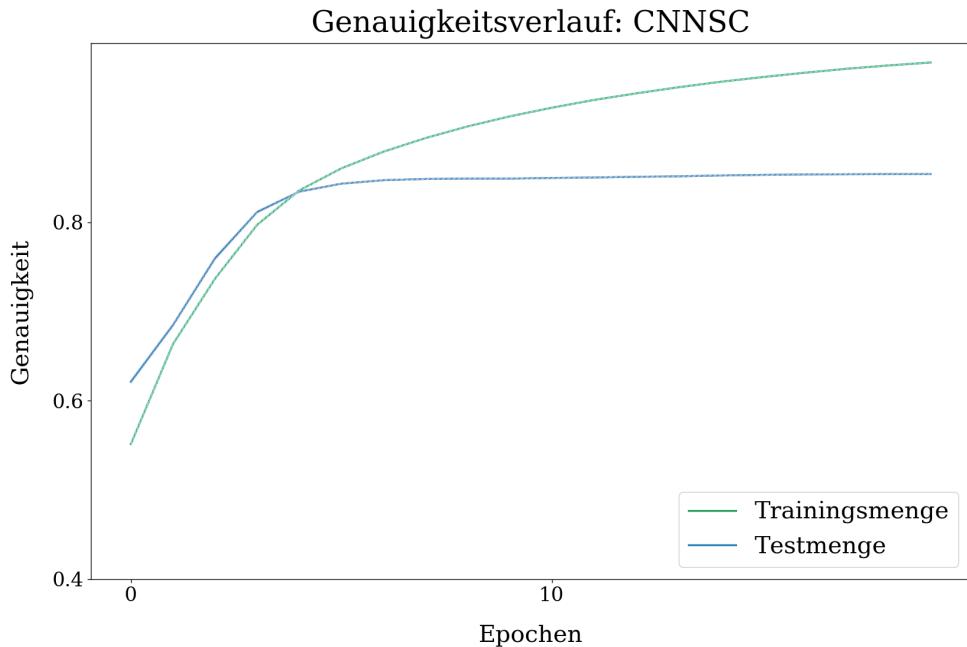


Abbildung D.8: CNNSC: Genauigkeitsverlauf. Die y-Achse repräsentiert die Genauigkeit gegen die Anzahl der Epochen des Trainings (x-Achse). In grün ist die Trainings- und blau die Testmenge zu sehen. Der Plot zeigt eine Vergrößerung der Ansicht des Ergebnisabschnittes [4.16](#).

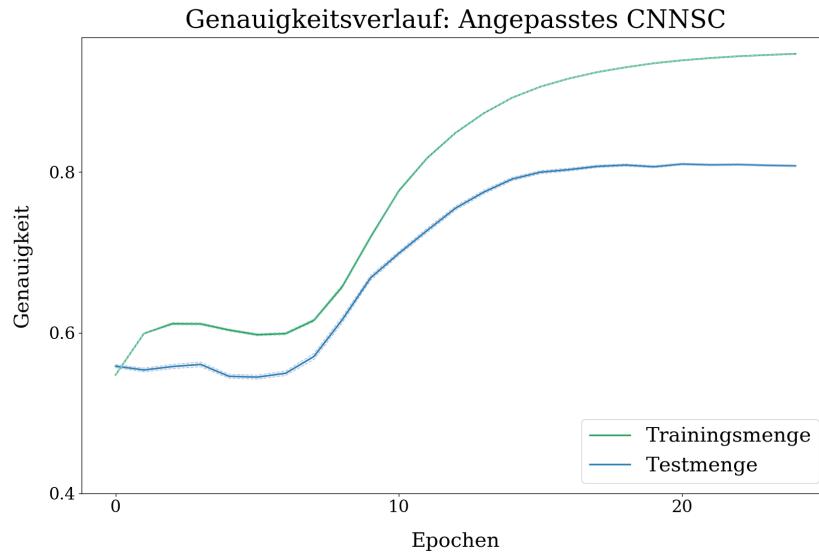


Abbildung D.9: Angepasstes CNNSC: Genauigkeitsverlauf. Die y-Achse repräsentiert die Genauigkeit gegen die Anzahl der Epochen des Trainings (x-Achse). In grün ist die Trainings- und blau die Testmenge zu sehen. Der Plot zeigt eine Vergrößerung der Ansicht des Ergebnisabschnittes 4.16.

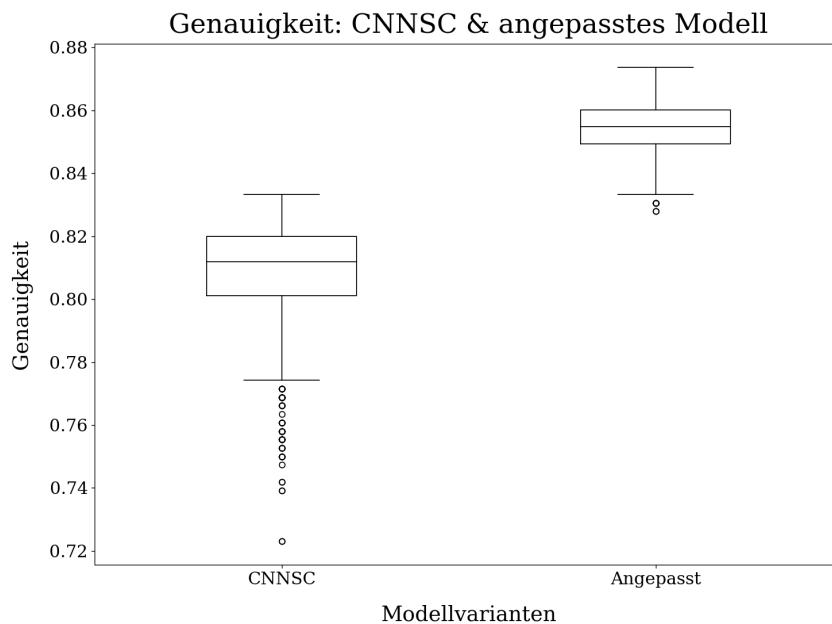


Abbildung D.10: Diese Abbildung stellt die Ergebnisse, hier die Genauigkeit, des CNNSC-Modells gegenüber des angepassten Models. Der linke Boxplot zeigt die Verteilung für das CNNSC-Modell und rechts unser angepasstes Modell über 1.000 Modelle an.

E Bitcoin und Sentiment

In diesem Anhang präsentieren wir weitere Informationen über die Zeitreihe. Zu Beginn betrachten wir die Korrelationen der Modelle genauer. Danach bieten wir weitere Einblicke in die Ergebnisse der linearen Regression mittels des originalen CNNSC-Sentiments.

Abbildung E.1 zeigt die Korrelation zwischen den Modellen und dem Bitcoin-Niveau. vaderSentiment, sentimentr, RF und das angepasste CNNSC-Sentiment zeigen eine hohe positive Korrelation. Wir vermuten, dass RF und das angepasste CNNSC-Sentiment die Regeln gut in ihren Modellen spiegeln. Außerdem zeigen alle vier Sentiments eine positive Korrelation zum Bitcoin-Niveau. Weiterhin besteht eine hohe Korrelation zwischen SVM, HDLTex- und dem angepassten HDLTex-Sentiment. Die drei Sentiments sind negativ zum Bitcoin-Niveau korreliert. Das einzige Sentiment, das heraussticht, ist das CNNSC-Sentiment. Es zeigt, trotz den guten Ergebnissen aus Kapitel 5, eine geringe Korrelation zum Bitcoin-Niveau.

Abbildung E.2 zeigt ebenfalls eine Korrelationsmatrix allerdings mit der log-Rendite, dem Sentiment und dem Bitcoin-Niveau. Bei den log-Renditen ergibt sich ein ähnliches Bild, nur mit niedrigeren Korrelationen, wie beim Bitcoin-Niveau.

Tabelle E.1 gibt die Ergebnisse von 5 Modellen mittels linearer Regression wieder. Im Gegensatz zum Zeitreihenanalysekapitel 5 verwenden wir acht Prädiktoren, siehe Formel (E.1).

$$\Delta B = \Delta S_0 + \Delta S_1 + \Delta S_2 + \dots + \Delta S_7 + \Delta x \quad (\text{E.1})$$

Δ beschreibt die Variable als log-Rendite, beispielsweise berechnet sich unser Ziel $\Delta B = b_t - b_{t-1}$, wobei ΔB für die Bitcoin-Rendite und ΔS_t für die CNNSC-Sentiment-Rendite steht. Weiterhin können wir verzeichnen, dass die meisten Prädiktoren älter als ΔS_2 nicht signifikant sind.

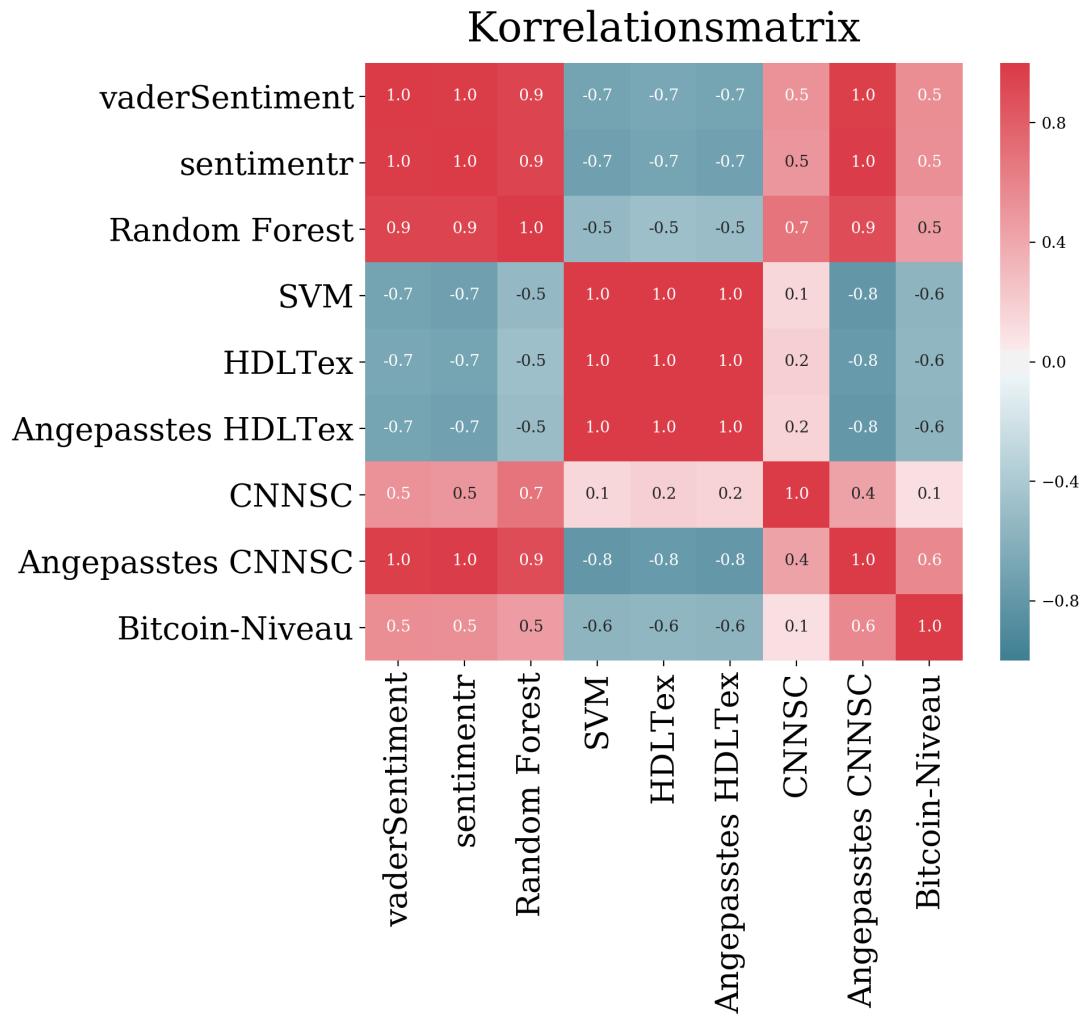


Abbildung E.1: Dieser Plot zeigt die Korrelation aller Modelle und dem Bitcoin-Niveau.

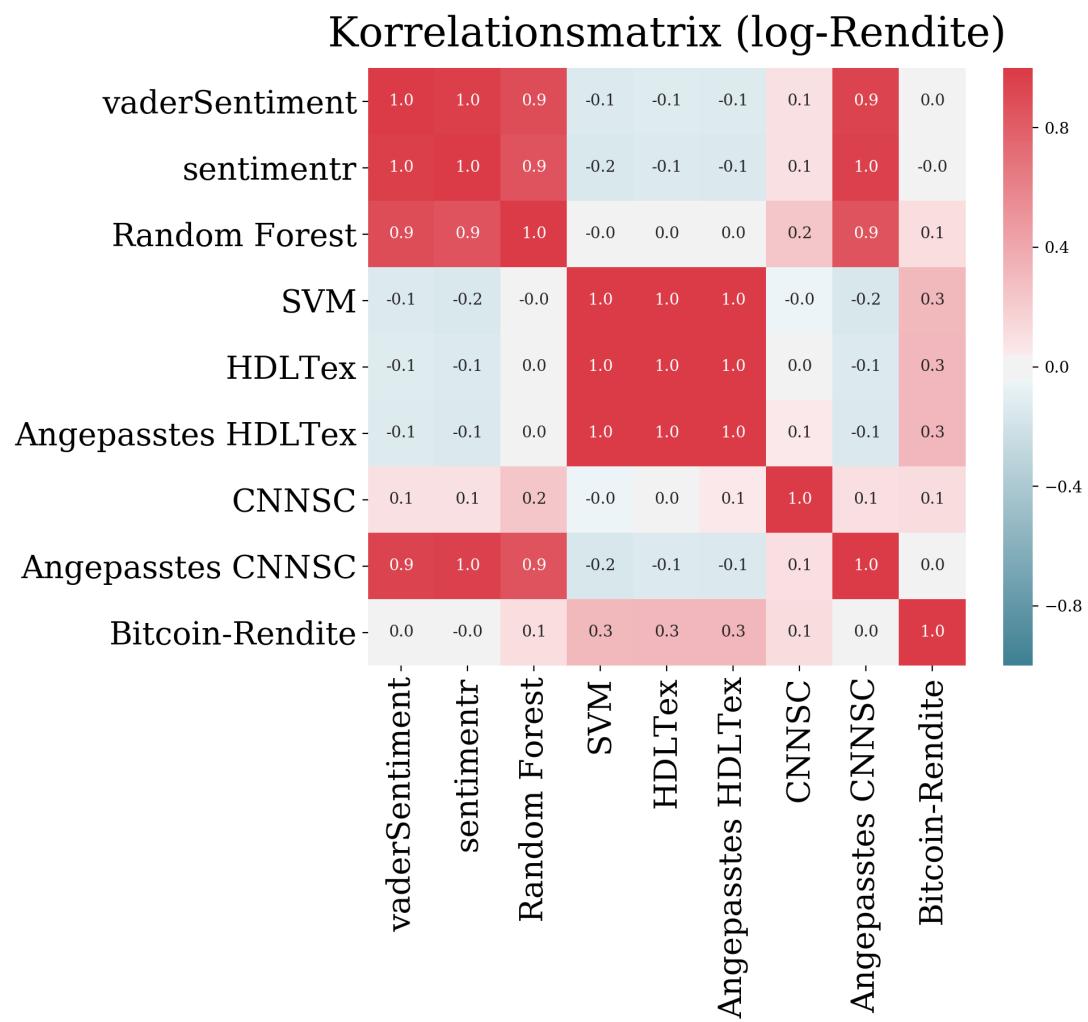


Abbildung E.2: Dieser Plot zeigt die Korrelation aller Modelle und der Bitcoin-Rendite.

Tabelle E.1: Sieben Tage Sentiment-Prognose für die Bitcoin-Rendite. ΔS_0 steht für die heutige Rendite, ΔS_1 die gestrige Rendite, ΔS_2 für die Rendite von vor zwei Tagen usw. des CNNSC-Sentiments. Die Koeffizienten berechnen wir mit NeweyWest und der zugehörige Standardfehler steht jeweils in der unteren Zeile. Der *Lineare Zeittrend* eliminiert den Trend, *#Tweets* repräsentiert die Anzahl der Tweets pro Tag, *Handelsvolumen* ist die Betragsmenge pro Tag, die gehandelt wird, und *Marktkapitalisierung* der Gesamtwert des Bitcoins.

	log-Rendite Bitcoin				
	(1)	(2)	(3)	(4)	(5)
ΔS_0	0,139*	0,148**	0,139**	0,153**	0,153**
	(0,081)	(0,068)	(0,067)	(0,072)	(0,069)
ΔS_1	0,156*	0,174***	0,163**	0,167**	0,199***
	(0,080)	(0,067)	(0,066)	(0,066)	(0,074)
ΔS_2	-0,274***	-0,235***	-0,232***	-0,244***	-0,217**
	(0,093)	(0,082)	(0,087)	(0,091)	(0,095)
ΔS_3	-0,138	-0,076	-0,063	-0,076	-0,037
	(0,105)	(0,123)	(0,116)	(0,118)	(0,130)
ΔS_4	-0,250**	-0,166	-0,157	-0,167	-0,132
	(0,099)	(0,121)	(0,117)	(0,119)	(0,126)
ΔS_5	-0,116	-0,009	-0,003	-0,021	0,012
	(0,105)	(0,142)	(0,136)	(0,138)	(0,146)
ΔS_6	-0,062	0,083	0,089	0,079	0,131
	(0,111)	(0,171)	(0,167)	(0,165)	(0,180)
ΔS_7	-0,220**	-0,063	-0,057	-0,066	-0,026
	(0,090)	(0,165)	(0,161)	(0,160)	(0,170)
Linearer Zeittrend		✓	✓	✓	✓
#Tweets			✓	✓	✓
Volumen				✓	✓
Marktkapitalisierung					✓
Beobachtungen	226	226	226	226	226
Angepasstes R^2	0,133	0,136	0,139	0,144	0,146

Notiz:

* p<0,1; ** p<0,05; *** p<0,01

F Beschreibung des Anhangs

Während einer Arbeit fallen Daten an, die zum Zweck der Nachvollziehbarkeit benötigt werden. Wir geben in diesem Teil einen Überblick in welchen Ordnern sich welche Ergebnisse befinden, siehe Tabelle F.1.

Tabelle F.1: Übersicht des physikalischen Anhangs.

Pfad	Beschreibung
<i>Masterthesis_Bernhard-Preisler.pdf</i>	Die vollständige digitale Thesis als PDF-Format.
<i>AMT/Rohdaten</i>	Hier befinden sich alle Ergebnisse aus AMT des Abschnittes 3.2.1. Die vier Dateien entsprechen unseren vier Annotationsvergängen.
<i>AMT/init_data.py</i>	Diese Datei setzt alle AMT-Ergebnisse so zusammen, dass die als tabellenartiges Objekt, hier Pandas, verwendet werden kann.
<i>Anforderungen/</i>	In Anforderungen aus dem Kapitel A sind alle Informationen über das Downloadsyste und die Versionen der Python-Pakete zu finden. Die Python-Pakete können mit pip ohne große Umstände installiert werden.
<i>Code/Building Models/</i>	In diesem Ordner sind alle Python-Skripte zu finden, die unsere unterschiedlichen Sentiments berechnen, siehe Abschnitt 4.7. Sie sind in .py als Skript, .ipynb als Jupyter Notebook und als .html für die direkte Ansicht verfügbar.
<i>Code/Cleaning/</i>	Diese Skripte dienen zur Datenbereinigung, siehe Absatz 3.2.2.
<i>Code/Others/</i>	Hier sind die Skripte für Krippendorf Alpha aus dem Abschnitt 3.2.1 und weitere zu finden.
<i>Code/Zeitreihe</i>	Enthält die Python-Skripte, um die einzelnen Sentiments der Zeitreihe aus dem Absatz 3.2.3 zu generieren, inkl. der Zusammensetzung der Zeitreihe. Sie sind in .py als Skript, .ipynb als Jupyter Notebook und als .html für die direkte Ansicht verfügbar.

<i>Data/Trainingsdatensatz/</i>	In diesem Ordner befindet sich der Trainingsdatensatz und die Aufteilung in Trainings- und Validierungsset, vgl. Abschnitt 3.2.1. Weiterhin enthält der Ordner die Twitter-IDs der annotierten Tweets.
<i>Data/Twitterdaten/</i>	Enthält den original heruntergeladen Twitterdatensatz (raw/) und die aufbereiteten (cleaned), vgl. die Abschnitte 3.2 und 3.2.2.
<i>Feature Importance</i>	Dieser Anhang enthält die Abbildungen der Feature Importance aus dem Anhang B.3.2.
<i>Data/Zeitreihe</i>	Enthält die generierte Zeitreihe aus dem Abschnitt 3.2.3.
<i>Deep Learning Ergebnisse/</i>	Hier sind alle Ergebnisse der Deep Learning-Modelle aus dem Abschnitt 4.7 zu finden. KNN steht für das HDLTex- und CNN für das CNNSC-Modell aus den Abschnitten 4.7.1 und 4.7.2. Diese Ordner unterteilen sich nochmals für jede Metrik in Unterordner. Außerdem sind in den Unterordnern die Modelle und deren Gewichte zu finden.
<i>PHP/</i>	Enthält alle Skripte, die wir zum Download der Tweets von Twitter verwenden, vgl. Absatz 3.2.
<i>Pre-Trained Vectors/</i>	In diesem Ordner liegen alle getesteten word2vec-Modelle aus dem Abschnitt 3.2.2.
<i>Webseiten/</i>	Enthält alle benötigten Webseiten, wie Python-Pakete, die Wortvektoren, die MongoDB, von Twitter und die Quelle von dem Volumen und der Marktkapitalisierung von Bitcoin.

Index

- Aktivierungsfunktion, 26
 - Leaky Relu, 26
 - Relu, 26
 - Sigmoid, 26
 - Swish, 26
 - Tanh, 26
- Amazon Mechanical Turk, 7
 - Requester, 7
 - Turks, 7
- Anforderungen
 - Linux, 59
 - PHP, 59
 - Python, 59
- Annotierte Tweets, 7
- Bias-Varianz-Tradeoff, 19
 - Bias, 19
 - Overfitting, 19
 - Underfitting, 19
 - Varianz, 19
- Bitcoin, 5
 - Blockchain, 5
 - Dezentralisierung, 5
 - Double-Spending, 5
 - Mining, 5
- Convolutional Neural Network, 31
 - AlexNet, 32
 - GoogLeNet, 32
 - Kernel, 31
 - Max-Pooling, 34
 - Padding, 32
 - Pooling, 32
 - Rand, 32
 - ResNet, 32
 - Stride, 32
 - VGG, 32
- Cronjob, 6
- Cross-Entropy-Verlustfunktion, 28
- Deep Learning
 - Adam, 30
 - Batch Gradient Descent, 30
 - Dropout, 29
 - Exponentially Weighted Moving Averages, 30
 - Frobenius Norm, 29
 - Gradient Descent with Momentum, 30
 - Hidden Layer, 20
 - Hyperparameter, 23
 - Input Layer, 25
 - Kostenfunktion, 28
 - Label, 15
 - Layer, 20
 - Lernrate, 22
 - Mini-Batch, 30
 - Output Layer, 25
 - Parameter, 23
 - Reinforcement Learning, 15
 - RMSProp, 30
 - Stochastischer Gradientenabstieg, 30
 - Supervised Learning, 15
 - Unsupervised Learning, 15
- Deep Learning-Modelle
 - CNNSC, 38
 - CNNSC (angepasst), 39
 - HDLTex, 35
 - HDLTex (angepasst), 37
- Entscheidungsbaum, 73
- Evaluationsdatensatz, 17
- Faltendes Neuronales Netzwerk, 31
- Feature Engineering

- Bag-of-N-Grams, 12
- Bag-of-Words, 12
- Frequency-based filtering, 13
- Rechtschreibkorrektur, 13
- Stemming, 13
- Stopwords, 13
- Token, 12
- Word Embedding, 12
- Word2Vec, 12
- Fit, 17
- Fitting, 17
- Graphentheorie, 18
 - Gerichteter Graph, 18
 - Kanten, 18
 - Knoten, 18
 - Ungerichteter Graph, 18
- Künstliches Neuronales Netz, 20
- Krippendorf Alpha, 9
- Large Scale Visual Recognition Challenge, 32
- Lernen, 17
- Parameterinitialisierung, 27
- PHP, 6
- Recurrent Neural Networks, 33
 - Forget Gate, 34
 - Gate Gate, 34
 - Input Gate, 34
 - LSTM, 33
 - Output Gate, 34
 - Self-connected Hidden Layer, 33
- Regularisierung, 29
- Sentiment, 17
- Sentimentanalyse, 16
- Support Vector Machines
 - Hyperebene, 75
 - Margin, 75
 - Support Vektor, 75
- Testdatensatz, 17
- Trainieren, 17
- Trainingsdatensatz, 7, 17
- Twitter
 - API, 6
 - Hashtag, 6
 - Retweet, 6
 - Tweet, 6
- Valence Aware Dictionary for sEntiment Reasoning, 71
- Vergleichsverfahren
 - Support Vector Machines, 75
- Vergleichverfahren
 - Random Forest, 73
 - sentimentr, 72
 - vaderSentiment, 71
- Verlustfunktionen, 28
- Wörterbuch, 71
- Word2Vec, 13
- Zeitreihenanalyse, 43
 - Komponenten, 44
 - Saisonale Naive Methode, 44
 - Trend, 43
 - Zyklisch, 43