Auswirkungen von Meldungen in sozialen Medien auf Aktienmärkte

Impact of social media announcements on stock markets

Bachelorarbeit

Vorgelegt von:

Andreas Credé

Jahnallee 2

04109 Leipzig

Matrikelnummer: 4704513

Studiengang: Finanz- und Wirtschaftsmathematik

Prüfer: Prof. Dr. Marc Gürtler

Abgabetermin: 05.03.2020

Technische Universität Braunschweig | Institut für Finanzwirtschaft
Abt-Jerusalem-Str. 7 | 38106 Braunschweig | Deutschland

Univ.-Prof. Dr. Marc Gürtler
Tel. +49 (0) 531 391-2896
Fax +49 (0) 531 391-2899
marc.guertler@tu-bs.de
http://www.fiwi.tu-bs.de

Aufgabenstellung zu einer Bachelorarbeit

Name, Vorname: Credé, Andreas
Matrikelnummer: 4704513
Studiengang: Finanz- und Wirtschaftsmathematik

Thema der Bachelorarbeit:
Auswirkungen von Meldungen in sozialen Medien auf Aktienmärkte
Impact of social media announcements on stock markets

Beginn: 28.11.2019
Abgabe: 28.02.2020

Als Zweitprüfer wird Herr Prof. Dr. Jens-Peter Kreiß benannt.

Offensichtlich haben soziale Medien heutzutage erheblichen Einfluss auf verschiedene Bereiche des Lebens. Daraus resultiert aus finanzwirtschaftlicher Sicht unmittelbar die Frage, ob soziale Medien auch Einfluss auf den Kapitalmarkt haben. In diesem Kontext erörtert inzwischen bereits eine Vielzahl wissenschaftlicher Beiträge die Auswirkungen von (insbesondere international bedeutenden) Berichterstattungen in sozialen Medien auf Aktienmärkte. Das Ziel der Berücksichtigung solcher Einflussfaktoren liegt in einer Verbesserung bestehender Regressionsmodelle und daraus resultierend einer verbesserten Aktienkursprognose.

Im Rahmen seiner Bachelorarbeit erhält Herr Credé die Aufgabe, Auswirkungen von Meldungen in sozialen Medien auf die Aktienmarktentwicklungen zu untersuchen und für ein Prognosemodell zu verwenden. Dabei soll zunächst ein Überblick zu bereits bestehenden Untersuchungen in der Literatur gegeben werden. Hierbei ist insbesondere die im jeweiligen Literaturbeitrag verwendete Methode zur Identifikation von kursrelevanten Informationen vorzustellen und auf die Prognosegüte des resultierenden Modells einzugehen. Anschließend soll Herr Credé auf Basis der Literaturergebnisse (in Abstimmung mit dem Institut für Finanzwirtschaft) eigenständig eine Variable konstruieren, die als Repräsentant für die Stimmungslage am Kapitalmarkt dienen soll. Diese Variable soll zusammen mit weiteren marktbeeinflussenden Variablen im Rahmen eines Regressionsmodells genutzt werden, Aktienmarktentwicklungen zu prognostizieren. Abschließend ist die Prognosegüte des resultierenden Modells zu beurteilen.

Prof. Dr. Marc Gürtler

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die genehmigten oder angegebenen Quellen und Hilfsmittel benutzt habe.

Ferner versichere ich, dass es sich hier um eine Originalarbeit handelt, die noch nicht in einer anderen Prüfung vorgelegen hat.

_____

Braunschweig, 4. März 2020

Inhaltsverzeichnis

Abkürzungsverzeichnis

ADF Augmented Dickey-Fuller Test

AIC Akaike-Informationskriterium (*Akaike information criterion*)

API Application Programming Interface

AR Autoregression

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

MAPE Mean average percentage error

MSE mean squared error

NLP Natural Language Processing

NLTK Natural Language Toolkit

OAuth Open Authorization

POS Part-of-speech

REST Representational State Transfer

RMSE Root-mean-square error

SQL Structured Query Language

STTS Stuttgart-Tübingen Tagset

VADER Valence Aware Dictionary and Sentiment Reasoner

VAR Vektorautoregression

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Über den Einfluss der sozialen Medien auf unseren Alltag wird sehr oft spekuliert. Facebook, YouTube und Instagram beeinflussen schon lange unser Konsumverhalten, sowohl durch zielgerichtete Werbung als auch durch unterbewusste Beeinflussung. Die Vermutung liegt nahe, dass auch andere Teile unseres alltäglichen Lebens von den sozialen Medien beeinflusst werden. Bereits 2013 zeigte eine Umfrage unter 360 britischen Finanzexperten[1], dass deren Mehrheit an den Einfluss der Inhalte sozialer Medien auf die Bewertung von Aktien glaubte. Trotz des Konsenses verwendeten damals nur 7% der befragten Finanzexperten diese Erkenntnis selbst in ihren Analysen als Indikator. Die Dissonanz zwischen dem empfundenem Einfluss der sozialen Medien auf den Aktienmarkt und der tatsächlichen Untersuchung durch die Erforschung dieses Gebietes zeigt einen sehr großen Handlungsbedarf auf, weswegen die folgende Arbeit sich mit dem Einfluss der sozialen Medien auf die Aktienkurse beschäftigen wird.

Die technische Analyse[2] verwendet bereits seit Jahren Indikatoren, um die Stimmung an den Märkten abzubilden und daraus Handelsentscheidungen abzuleiten. So lässt sich zum Beispiel aus der Put-Call-Ratio, d.h. dem Verhältnis von gehandelten Verkaufs- zu Kaufoptionen, der Verkaufsdruck in einem Markt direkt aus den Umsatzzahlen ermitteln[3]. Andere Indikatoren werden aufgrund von Umfragen unter Marktteilnehmern ermittelt. So erfragt beispielsweise die *American Association of Individual Investors* wöchentlich unter Privatanlegern die erwartete Entwicklung der Aktienmärkte in den nächsten sechs Monaten. Eine ähnliche Umfrage unter Anlageberatern wird seit 1963 regelmäßig vom US-amerikanischen Forschungsinstitut *Investors Intelligence* durchgeführt. Bei all diesen Indikatoren werden jedoch nur Meinungen spezieller Gruppen von Marktteilnehmern analysiert – einen (öffentlich zugänglichen) Indikator für eine „allgemeine“ Stimmung gibt es nicht, obwohl die Daten in Form von Facebook-Statusmeldungen, Instagram-Posts und Tweets etc. öffentlich zugänglich sind. Das folgende Beispiel zeigt, wie stark selbst der Einfluss einzelner Tweets auf den Aktienkurs eines Unternehmens sein kann.

---

[1] Vgl. Services (2013).

[2] Mit der technischen Analyse wird ein Form der Finanzanalyse bezeichnet, die sich allein auf die Kurs- und Umsatzhistorie eines Wertes bezieht.

[3] Vgl. Murphy (2014), S. 186.

Beispiel 1.1: Einführendes Beispiel

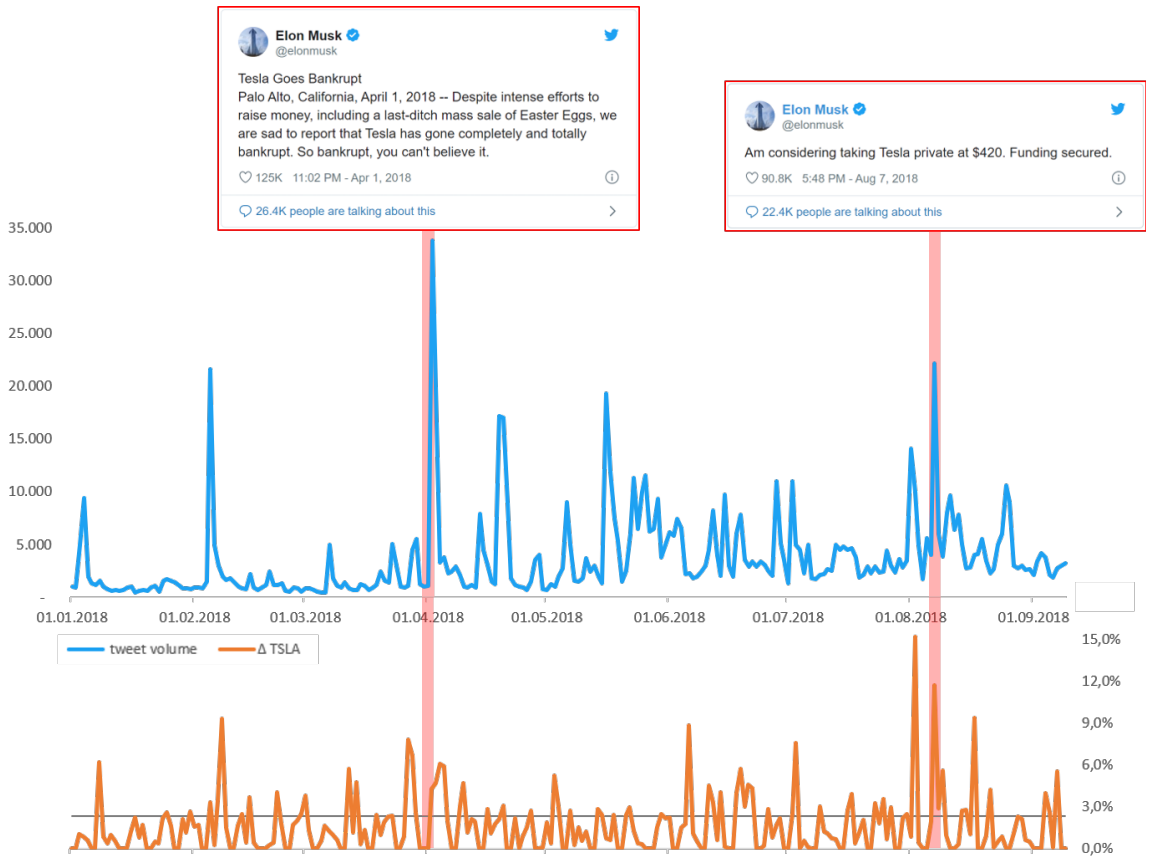

Abbildung 1.: Anzahl Tweets pro Tag (hellblau) über Tesla und dessen Vorstandsvorsitzenden, Elon Musk (01.2018 - 09.2018), Aktienkurs von Tesla (orange), Mittlere absolute Veränderung des Kurses (grau)

Abbildung 1 zeigt das Aufkommen an Tweets über Tesla (unter dem Hashtag #tesla) und über den Gründer und Vorstandsvorsitzenden Elon Musk (Twitter-Nutzer @elonmusk). Deutlich erkennbar sind die beiden Spitzen in der Anzahl der Tweets im April und August, die in Zusammenhang mit den kontrovers diskutierten Tweets von Elon Musk stehen. Abbildung 1 zeigt in der unteren Zeitreihe die Schwankungen[4]in Teslas Aktienkurs im selben Zeitraum und die deutlich erkennbare höhere Schwankung (4,26% bzw. 11,72% vs. durchschnittlichen 2,33% täglicher Schwankung) des Aktienkurses an Tagen mit kontroversen Tweets über das Unternehmen.

Ausschläge wie dieser legen nahe, dass es einen Zusammenhang zwischen der Stimmung in sozialen Netzwerken und dem Aktienkurs eines Unternehmens geben könnte. Sie zeigen aber auch, dass der Inhalt der Nachrichten über die Richtung des Kurses entscheidend

sein kann. So gab der Aktienkurs nach dem „Aprilscherz-Tweet“ vom 1. April 2018 nach, während er nach dem "Privatisierungs-Tweet"vom 7. August 2018 deutlich zunahm.

Die folgende Arbeit gliedert sich in drei Teile. Im ersten Teil wird zunächst ein Überblick über bisherige Forschungsarbeiten mit ähnlichen Fragestellungen gegeben. Dabei werden die verwendeten Modelle vorgestellt und hinsichtlich ihrer Prognosegüte verglichen. Der zweite Teil behandelt die Erfassung und Verarbeitung von Texten, die zur Entwicklung einer Stimmungsvariablen notwendig sind. Dabei werden zunächst die Grundlagen der Computerlinguistik erklärt und ein Modell zur lexikonbasierten Sentiment-Analyse von Texten vorgestellt. Anschließend wird der im Rahmen dieser Arbeit erhobene Textkorpus[5] mit den vorgestellten Methoden analysiert und daraus eine Variable für die Stimmung abgeleitet. Im dritten Teil wird eine Einführung in vektorautoregressive Modelle für Zeitreihen gegeben und wie sich daraus Prognosen erzeugen lassen. Auf Basis der in Teil 2 entwickelten Variable wird schließlich ein Modell trainiert und die Güte der daraus erzeugten Prognose analysiert.

[4]Gemessen wurde die Veränderung zum Vortag in Prozent.

[5]Als Textkorpus wird eine Sammlung schriftlicher Texte bezeichnet, die meist einer gemeinsamen Sprache oder Textgattung angehören.

2. Verwandte Arbeiten

In diesem Kapitel werden verwandte Arbeiten mit ähnlichen Fragestellungen vorgestellt und hinsichtlich ihrer verwendeten Methoden und der Güte der verwendeten Modelle analysiert und verglichen.

2.1. Überblick

Zahlreiche Arbeiten wurden bereits dem Thema der Prognose von Aktienkursen gewidmet. Frühe Forschungsansätze gehen dabei oft von der Annahme der Markteffizienzhypothese[6] und der Random-Walk-Theorie[7] aus, nach denen Aktienkurse nur von neuen Informationen beeinflusst werden und sich nicht aufgrund von Vergangenheitswerten vorhersagen lassen. Demzufolge dürfte kein Marktteilnehmer in der Lage sein, überdurchschnittliche Gewinne zu erwirtschaften und eine Vorhersage eines Aktienkurses dürfte eine Genauigkeit von 50% langfristig nicht überschreiten.

Obwohl Eugene Fama 2013 für seine Forschungen auf diesem Gebiet der Alfred-Nobel-Gedächtnispreis für Wirtschaftswissenschaften verliehen wurde, konnte in jüngeren Forschungen die Annahme widerlegt werden, dass Aktienkurse einem Random-Walk folgen[8]. Andere Untersuchungen versuchten die These zu widerlegen, dass Ereignisse und die daraus resultierenden Nachrichten re in zufällig und damit unvorhersehbar sind. Die zunehmende Verfügbarkeit und Schnelligkeit von Informationen, insbesondere durch technische Innovationen wie soziale Netzwerke, ermöglicht zwar nicht eine Vorhersage von Ereignissen bevor diese eintreten, es konnte jedoch gezeigt werden, dass gerade aus sozialen Netzwerken gewonnene Informationen als frühe Indikatoren für eine Vorhersage dienen können. Die Anwendungsgebiete gehen dabei weit über die Vorhersage von Aktienkursen hinaus. So konnten zum Beispiel aus den Google Trends[9]

[6] Vgl. Fama (1965).

[7] Vgl. Fama u. a. (1969).

[8] Vgl. Qian und Rasheed (2007).

[9] Bei *Google Trends* handelt es sich um einen Dienst von Google, der den relativen Anteil von Suchbegriffen der Google-Suchmaschine bereitstellt.

Vorhersagen für Reiseziele und Arbeitslosigkeitszahlen[10] oder Ausbrüche von Infektionskrankheiten[11] getroffen werden.

Die Annahme, dass Informationen aus verschiedensten Quellen im Internet eine Vorhersage für einen Aktienkurs verbessern können, erscheint also begründet. Der (explizite) Inhalt der Nachrichten stellt aber nicht den einzigen Einflussfaktor auf Preise an Aktienmärkten dar. Aus der Verhaltenspsychologie ist bekannt, dass auch die mit den Nachrichten verbundenen Emotionen einen nicht zu vernachlässigenden Einfluss auf die Entscheidungen von Investoren haben und damit auch direkten Einfluss auf Aktienmarktpreise haben[12]. Das Teilgebiet der *Behavioral Finance* beschäftigt sich mit Erklärungsansätzen, warum Marktteilnehmer z. B. durch Unter- oder Überraktionen auf Informationen irrationale Entscheidungen treffen, die zu Ineffizienzen am Markt führen.

Die Quantifizierung der öffentlichen Meinung zu bestimmen und ihren Einfluss auf die Aktienmärkte zu messen ist daher Ziel zahlreicher Forschungsarbeiten geworden. Einen ersten Ansatz für eine strukturierte Analyse der Stimmung von Nachrichten in Microblogging-Diensten liefert [Pak2010]. Dabei wurden gezielt Nachrichten von Twitter (kurz *Tweets*) mit eindeutigen positiven oder negativen Stimmungen gesucht, indem nach Tweets mit eindeutig „traurigen“ oder „fröhlichen“ Emoticons gesucht wurde. Mit dem dadurch gewonnenen Datensatz von etwa 300.000 Tweets konnte ein multinomialer naiver Bayes-Klassifikator trainiert werden, mit dem schließlich weitere Tweets analysiert werden konnten. Dabei konnte gezeigt werden, dass basierend auf Tweets ein Klassifikator gebaut und ein Indikator für eine öffentliche Stimmung generiert werden kann[13].

Dass diese Daten nun auch für die Vorhersage von Aktienkursen genutzt werden können, konnte in [Bollen2011] gezeigt werden. Dabei wurde ein deutlich größerer Korpus von etwa 9,85 Mio. Tweets mittels zweier proprietärer Lösungen (*Google-Profile of Mood States, GPOMS* und *Opinion Finder*) analysiert und die Stimmung in einem täglichen Stimmungsindikator zusammengefasst. Berücksichtigt wurden dabei nur Tweets mit expliziten subjektiven Gefühlsäußerungen

[10] Vgl. Varian und Choi (2011).

[11] Vgl. Pelat u. a. (2009).

[12] Vgl. Nofsinger (2003), S. 148f.

[13] Pak und Paroubek (2010).

(z. B. beginnend mit „I am feeling“). Mithilfe der gewonnenen Stimmungsdaten wurde schließlich ein Modell für die Vorhersage des Dow Jones Industrial Average trainiert. Dabei wurde ein selbstorganisierendes Fuzzy-Neuronales Netz verwendet. Bei der Vorhersage der Richtung des Dow Jones, also einer positiven oder negativen Entwicklung am vorhergesagten Tag, erreichte das trainierte neuronale Netz eine Präzision von 87,6%[14].

Vergleichbare Ergebnisse finden sich auch in [Mao2012]. Dabei wurde über einen Zeitraum von 2 Monaten die Anzahl der Tweets, die Unternehmen aus dem S&P 500 erwähnen, verwendet, um die Vorhersage eines linearen Regressionsmodells zu verbessern. Die Analyse wurde dabei jeweils für den gesamten Index, den einzelnen Sektoren des Index und auf Ebene einer einzelnen Aktie (in diesem Fall der Aktie der Apple Inc.) durchgeführt. Auf Ebene des gesamten Index konnte die Genauigkeit des Prognosemodells für die Änderungsrichtung des Aktienkurses mit 68% die erwarteten 50% eines „zufälligen Ratens“ bei weitem übertreffen. Für die einzelnen Sektoren und die einzelne Aktie konnte keine verlässliche Prognose erzeugt werden. Es konnte jedoch eine (positive) Korrelation zwischen der Anzahl der Tweets und dem Handelsvolumen gezeigt werden[15].

Eine ähnliche Untersuchung findet sich in [Zhang2011]. Anhand einer Menge als „emotionsgeladen“ klassifizierter Wörter (z. B. „hope“, „fear“, „happy“) wurde eine Variable entwickelt, die die Anzahl der Tweets dieser Stimmung widerspiegelt. Die Variable wurde anschließend mit verschiedenen Indizes verglichen, wobei sich eine besonders starke Korrelation zum S&P 500 und eine (negative) Korrelation zum VIX[16] zeigte. Ein Prognosemodell mit vergleichbaren Präzisionswerten wurde jedoch nicht aufgestellt[17].

2.2. Fazit

Die bisherigen Untersuchungen legen die Vermutung nahe, dass sich auf Basis von Meldungen in sozialen Medien ein verbessertes Prognosemodell für Aktienkurse erzeugen lassen sollte. Die bisherigen Untersuchungen behandeln jedoch bisher meist nur die Prognose ganzer Aktienindi-

[14] Bollen, H. Mao und Zeng (2011).

[15] Vgl. Y. Mao u. a. (2012).

[16] Beim *VIX* handelt es sich um einen Volatilitätsindex. Er drückt die erwartete Volatilität des S&P 500 aus.

[17] Vgl. Zhang, Fuehres und Gloor (2011).

zes. Eine stabile Vorhersage für einzelne Aktienkurse findet sich lediglich in [Mao2012], jedoch mit dem Ergebnis, dass sich keine verbesserte lineare Regression erzeugen ließ. Eine Korrelation ließ sich nur zwischen der Anzahl der Meldungen und dem Handelsvolumen zeigen[18].

Die vorgestellten Untersuchungen betrachten zudem teilweise nur sehr kurze Zeiträume (z. B. nur etwa $2,5$ Monate bei [Mao2012]), wodurch sich kein repräsentativer Textkorpus für die Analyse der Stimmung erzeugen lässt. Einige der Modelle basieren außerdem auf proprietärer Software für die Analyse der Stimmung (z. B. GPOMS / OpinionFinder bei [Bollen2011]), wodurch eine Analyse und Reproduktion der Ergebnisse erschwert wird. Andere Untersuchungen verwenden hingegen nur einfache, nicht an Texte aus sozialen Netzwerken angepasste Modelle zur Sentiment-Analyse, wodurch sich ein ungenaues Stimmungsbild ergeben kann.

[18] Vgl. Y. Mao u. a. (2012).

3. Entwicklung einer Sentiment-Variablen

Das Ziel dieses Kapitels ist die Entwicklung einer Variablen, mit der das Sentiment über ein bestimmtes Thema quantifiziert werden kann. Dafür müssen die Daten aus den sozialen Netzwerken zunächst analysiert werden. Dieses Kapitel gibt daher eine Einführung in Methoden des Natural Language Processing, bevor die verarbeiteten Texte einer Sentiment-Analyse unterzogen und daraus eine Sentiment-Variable abgeleitet wird. Anschließend wird die Implementierung der erläuterten Methoden in Python vorgestellt.

3.1. Grundlagen des Natural Language Processing

Die Verarbeitung und Analyse strukturierter Daten, z. B. in Tabellenform, stellt für Computer dank gut entwickelter Programmiersprachen kaum ein Problem dar. Menschliche Sprache liegt hingegen selten in einem strukturierten Datenformat vor, welches von Computern verarbeitet werden kann. Das Gebiet des **Natural Language Processing**[19] (NLP) beschäftigt sich mit Methoden, wie natürliche Sprache in ein für Computer verständliches Format überführt werden kann. Diese Verarbeitung erfolgt in mehreren Teilschritten, die in der Praxis meist sequentiell durchgeführt werden. Man spricht daher auch von einem *Pipeline-Modell* spricht. Jeder Prozessschritt baut dabei auf der Ausgabe des vorherigen Schrittes auf. Je nach Anwendungsfall können die einzelnen Verarbeitungsschritte jedoch abweichen. Für den vorliegenden Fall wird das in Abbildung 2 dargestellte *Saarbrücker Pipeline-Modell* verwendet.

Abbildung 2.: Darstellung des Saarbrücker Pipeline-Modells. Die grauen Prozesschritte sind für die Analyse von Tweets nicht notwendig und entfallen daher.

Die **Spracherkennung** dient dazu, gesprochene Sprache in Textform umzuwandeln. Da die Daten bereits in Textform vorliegen und nicht als Audio-Signal, entfällt dieser Schritt. Bei der **Dialog- und Diskursanalyse** werden aufeinanderfolgende Sätze und ihre Beziehungen untereinander analysiert. Bei den zu verarbeitenden Daten handelt es sich in der Regel nur um einzelne (oder sehr wenige) Sätze, die nicht in komplexen Beziehungen zueinander stehen. Daher wird auf diesen Schritt verzichtet.

[19] Im Deutschen werden für NLP meistens die Begriffe *Computerlinguistik* und *Linguistische Datenverarbeitung* synonym verwendet.

Tokenisierung

Bei der Tokenisierung wird ein als Zeichenkette vorliegender Text auf Wortebene segmentiert. Ein *Token* ist dabei eine Instanz einer Zeichenfolge, die als semantische Einheit für die Weiterverarbeitung zusammengefasst wurde[20]. Bei der Verarbeitung menschlicher Sprache werden die Begriffe *Token* und *Wort* oft synonym verwendet. Eine einfache Form der Tokenisierung ist die sog. White-Space-Tokenisierung, bei der der Text an Satz- und Leerzeichen getrennt wird. Es existieren allerdings auch komplexere Tokenizer, die beispielsweise mit regulären Ausdrücken[21] arbeiten. Solche komplexeren Tokenizer sind notwendig, da nicht in allen Schriftsprachen eine White-Space-Tokenisierung möglich ist. Die japanische und chinesische Schrift verwenden beispielsweise keine Leerzeichen zwischen einzelnen Wörtern.

Die in der Praxis verwendeten Tokenizer arbeiten oft in zwei Stufen. In der ersten Stufe wird ein vorliegender Text in einzelne Sätze geteilt. Dabei werden verschiedene Methoden verwendet, deren Effizienz stark vom Anwendungsfall abhängt. So nutzt der in der Implementierung verwendete *Punkt-sentence-Tokenizer*[22] beispielsweise unüberwachtes Lernen, um aus einem großen Textkorpus einer Sprache eine Sammlung von Abkürzungen, Redewendungen und Satzanfängen zu erstellen, mit der er schließlich Sätze trennen kann. Der Tokenizer ist damit unabhängig von der verwendeten Sprache einsetzbar, sofern ein ausreichender Textkorpus für das Training vorhanden ist. Die zweite Stufe des Tokenizers unterteilt den Satz schließlich in einzelne Tokens. Die Implementierung verwendet dabei eine Menge regulärer Ausdrücke, um zum Beispiel Satzzeichen zu erkennen und Kontraktionen zu trennen ([„We'll“] → [„We“], [„'ll“]).

In einem weiteren Schritt wird anschließend jedem Token des Textes eine Wortart, der *part-of-speech* (POS), basierend auf der Definition des Wortes und den angrenzenden Wörtern hinzugefügt. Dafür benötigt der POS-Tagger ein sog. *Tagset*, also eine Menge von Abkürzungen (den *Tags*) für die verschiedenen Wortarten. Für die englische Sprache ist das *Penn Treebank Tagset*[23] weit verbreitet – für die deutsche Sprache hat sich das Stuttgart-Tübingen Tagset[24]

[20] Vgl. Manning, Raghavan und Schütze (2008), S. 22.

[21] Reguläre Ausdrücke sind eine formale Sprache, die zur Beschreibung von Mengen von Zeichenketten mit syntaktischen Regeln dienen.

[22] Vgl. Kiss und Strunk (2006), S. 488 ff.

[23] Vgl. Marcus, Santorini und Marcinkiewicz (1993), S. 317.

[24] Vgl. Schiller, Teufel und Thielen (1995).

(STTS) als Standard etabliert.

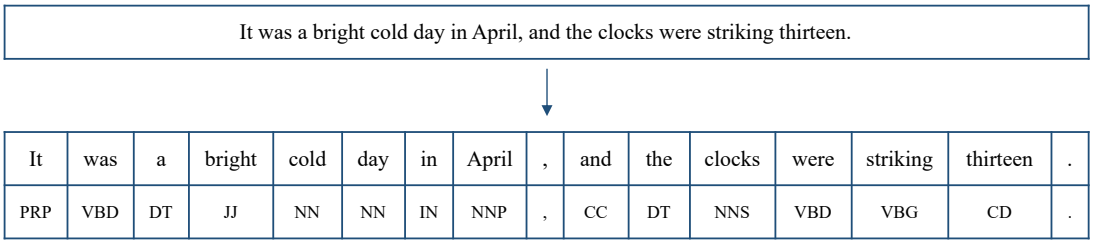

Abbildung 3.: Beispiel einer Tokenisierung mit POS-Tagging (PRP = Personalpronomen, JJ = Adjektiv, ...)[25]

Der in der Implementierung verwendete POS-Tagger basiert auf dem Perzeptron-Algorithmus[26], einer vereinfachten Form eines neuronalen Netzes. Die Ausgabe des Perzeptrons ist dabei der gesuchte POS aus dem Penn Treebank Tagset. Das Perzeptron wurde mit verschiedenen Textkorpora trainiert (u.a. Artikel des *Wall Street Journals*, Nachrichten verschiedener Nachrichtenagenturen sowie verschiedene Internet-Quellen wie Blogs, Webseiten und Social Media.).

Morphologische Analyse

In der morphologischen Analyse wird versucht, Flexionen von Wörtern zu identifizieren und sie auf ihre Grundform zurückzuführen - das *Lemma*.

Definition 3.1: Lemma

Ein **Lemma** ist die Grundform oder kanonische Form eines Wortes, unter der ein Begriff in einem Nachschlagewerk vorzufinden ist.

Ein Lemma ist also die Grundform eines Wortes ohne Deklinationen, Konjugationen oder andere Flexionen, also z. B. Verben im Infinitiv, Substantive im Singular etc. Aufgabe der morphologischen Analyse ist die Rückführen jedes Wortes auf das entsprechende Lemma. In der Praxis kommen dabei zwei verschiedene Verfahren zur Anwendung: Die *Stammformreduktion* und die *Lemmatisierung*. Das heuristische Verfahren der Stammformreduktion versucht Wortendungen zu entfernen und dadurch Wörter auf ihre Grundformen zurückzuführen. Die Lemmatisierung hingegen nutzt Wörterbücher und morphologische Analysemethoden für die Rückführung auf

[25] Eine vollständige Auflistung der POS-Tags findet sich in Anhang A.2.

[26] Vgl. Rosenblatt (1958).

ein gemeinsames Lemma, indem flexierte Wortteile verändert oder entfernt werden. Die Lemmatisierung ist der Stammformreduktion dabei hinsichtlich der Genauigkeit überlegen, was insbesondere darauf zurückzuführen ist, dass die Lemmatisierung bei der Analyse umfangreichere Daten wie zum Beispiel Synonyme einbeziehen kann[27].

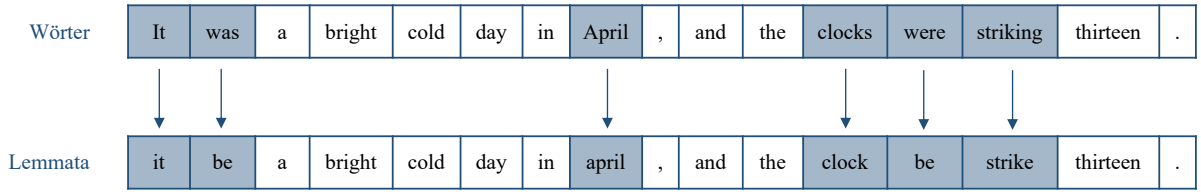

Abbildung 4.: Beispiel einer Morphologischen Analyse mittels Lemmatisierung. Die durch Lemmatisierung veränderten Worte sind farblich markiert. Alle Worte liegen nach der Lemmatisierung in ihrer Grundform vor: Verben im Infinitiv, Substantive im Nominativ Singular etc.

Die Implementierung verwendet den *WordNet Lemmatizer*, einen lexikonbasierten Lemmatisierer auf Basis von *WordNet*, einer umfangreichen lexikalischen Datenbank englischer Wörter[28]. Der Lemmatisierer versucht, ein Lemma in WordNet zu finden. Wenn kein direkter Treffer möglich ist, werden solange Umformungsregeln angewendet, bis ein entsprechendes Lemma in WordNet gefunden wurde, auf das das Wort zurückgeführt werden kann. Dabei sind sowohl das Lexikon (WordNet) als auch das Regelwerk sprachabhängig. Für die deutsche Sprache existiert mit *GermaNet*[29] ein ähnliches Projekt.

Syntaktische Analyse

Die Ausgabe aus der Lemmatisierung behandelt bisher nur einzelne Wörter bzw. Lemmata und setzt diese nicht in Relation zueinander. Dies ist die Aufgabe eines *Parsers* in der syntaktischen Analyse. Dieser zerlegt einen Satz in seine grammatikalischen Bestandteile und weist jedem Wort eine Funktion im Satz (Subjekt, Prädikat, Objekt, ...) zu. Die einzelnen Worte werden somit hinsichtlich ihrer grammatikalischen Beziehungen zueinander analysiert.

Für die syntaktische Analyse eines Satzes wird eine *formale Grammatik* benötigt. Darunter versteht man eine vollständige Liste eindeutiger, formalisierter Regeln zur Bildung von Sät-

[27] Vgl. Balakrishnan und Ethel (2014), S. 178.

[28] Vgl. Soergel (1998).

[29] Vgl. Kunze und Lemnitzer (2002).

zen[30]. Mit einer formalen Grammatik lassen sich ausgehend von einem Startsymbol S die (Produktions-)Regeln anwenden, um einen Satz zu erzeugen. Die Anwendung einer Regel wird als *Ableitung* bezeichnet. Die Symbole links der Ableitung unterscheidet man in *Terminalsymbole* T und *Nichtterminalsymbole* N. Dabei werden solange Regeln angewendet, bis keine Nichtterminalsymbole mehr vorliegen. Beispiel 3.1 zeigt eine *kontextfreie Grammatik*[31] und Erläuterungen zu den einzelnen Regeln.

Definition 3.2: Grammatik[32]

Eine **formale Grammatik** ist ein 4-Tupel $G = (V,\ T,\ P,\ S)$ mit

- einem Vokabular V
- einer Teilmenge $T \subset V$ von Terminalsymbolen
- einer Menge P von Produktionsregeln
- einem Startsymbol $S \in V \setminus T$

Die Menge der Nichtterminalsymbole ergibt sich damit durch $N = V \setminus T$.

Beispiel 3.1 zeigt, wie man unter Anwendung der Produktionsregeln einen Satz erzeugen kann. Selbst mithilfe der gegebenen Grammatik kann jedoch eine Vielzahl von Sätzen erzeugt werden, wobei jeder Satz grammatikalisch korrekt ist, aber nicht unbedingt inhaltlich richtig oder sinnvoll sein muss. So kann zum Beispiel der Satz „The cat sat the dog.“ erzeugt werden, der nur wenig Sinn ergibt. Die „Ableitungsgeschichte“[33] des Satzes kann (wie in Abbildung 5) als Baumdiagramm (sog. „parse tree“) dargestellt werden.

[30] Die Regeln dienen eigentlich der Bildung von Wörtern (im Sinne der theoretischen Informatik als Folge von Symbolen eines Alphabets), wobei ein Wort in der theoretischen Informatik einem Satz in der deutschen Sprache entspricht. Im Folgenden wird statt dem informatischen „Wort“ der „Satz“ verwendet.

[31] Dabei handelt es sich um einen Spezialfall einer Grammatik, bei der jede Regel genau ein Nichtterminalsymbol auf beliebige andere Symbole ableitet.

[32] Vgl. Hromkovic (2011), S. 353.

[33] Als „Ableitungsgeschichte“ wird die Folge von Anwendungen der Produktionsregeln (Ableitungen) bezeichnet.

Beispiel 3.1: Anwendung einer kontextfreien Grammatik

(1)	$S \rightarrow NP\ VP$
(2)	$PP \rightarrow P\ NP$
(3)	$NP \rightarrow Det\ N \mid NP\ PP$
(4)	$VP \rightarrow V\ NP \mid NP\ PP$
(5)	$Det \rightarrow$ 'a' | 'the'
(6)	$N \rightarrow$ 'dog' | 'cat'
(7)	$V \rightarrow$ 'chased' | 'sat'
(8)	$P \rightarrow$ 'on' | 'in'

NP = *noun phrase*, VP = *verb phrase*, PP = *adposition phrase*, | ist als logisches „oder“ zu verstehen. Durch Anwendung der Produktionsregeln lassen sich (beginnend beim Startsymbol *S*) Sätze erzeugen:

$S \xrightarrow{(1)}$	NP VP
NP VP $\xrightarrow{(3),(4)}$	Det N V NP
Det N V NP $\xrightarrow{(3)}$	Det N V Det N
Det N V Det N $\xrightarrow{(5),(6),(7),(5),(6)}$	„The dog chased a cat“

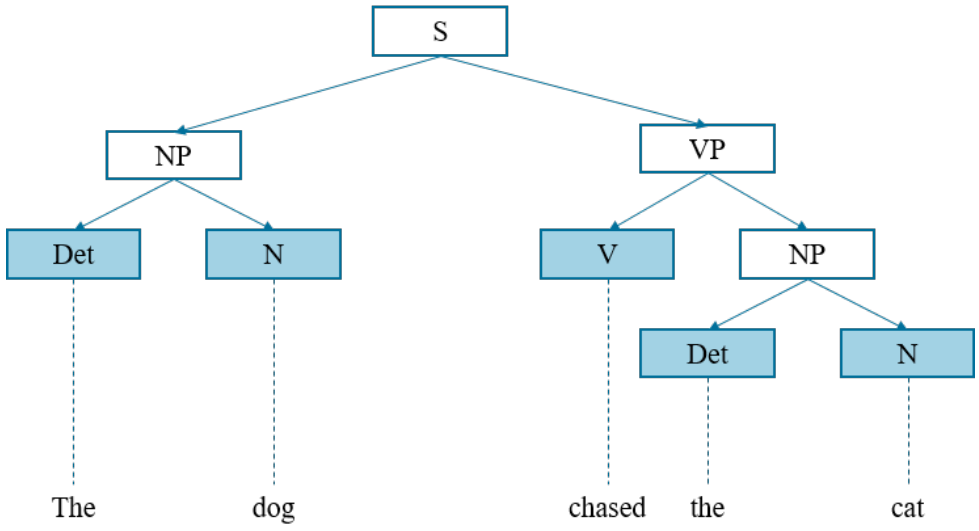

Abbildung 5.: Parse Tree des Satzes aus Beispiel 3.1. Die Anwendung einer Produktionsregel (Ableitung) wird durch einen Pfeil nach unten dargestellt.Die letzte Ableitung auf das Terminalsymbol ist als gestrichelte Linie dargestellt und bildet den für Menschen verständlichen Satz[34].

Man erkennt an Abbildung 5 auch die sukzessive Zerlegung des Satzes in seine syntaktischen Bestandteile. Die weitere Analyse des Textes kann nun auf unterschiedlichen Ebenen dieser Zerlegung stattfinden. Dazu betrachten wir in der weiteren Analyse die *Phrase* als syntaktische Einheit.

Definition 3.3: Phrase

Eine **Phrase** ist eine abgeschlossene (d.h. syntaktisch gesättigte) syntaktische Einheit.

Mit „syntaktisch gesättigt“ wird in diesem Zusammenhang die Eigenschaft einer Phrase bezeichnet, dass sie alle notwendigen Ergänzungen enthält. Der Begriff ist verwandt mit dem geläufigeren Wort des *Satzglieds*, der einen Spezialfall einer Phrase darstellt. Die semantische bzw. Sentiment-Analyse, wie sie in den nächsten Abschnitten vorgestellt wird, bezieht sich stets auf die Zerlegung eines Satzes in Phrasen, um die vollständige inhaltliche Bedeutung zu erfassen.

Semantische Analyse

Unter dem Begriff der semantischen Analyse wird eine Vielzahl von Analysemethoden zusammengefasst, deren gemeinsames Ziel es ist, die Bedeutung eines Textes zu erfassen. Dazu gehört auch, die mit einem Text verbundene Stimmung im Rahmen einer Sentiment-Analyse zu ermitteln, die im folgenden Abschnitt beschrieben wird.

3.2. Sentiment-Analyse

Die Sentiment-Analyse ermittelt die in einem Text vermittelte Stimmung des Autors. Anders als bei der *Emotionsanalyse*, bei der spezielle Emotionen wie Freude, Gelassenheit oder Wut identifiziert werden, beschränkt sich die Sentiment-Analyse auf die Analyse der Polarität („positiv“ oder „negativ“) und ggf. der Subjektivität eines Textes. Einen Text als eindeutig positiv oder eindeutig negativ einzustufen ist (insbesondere für Computer) keine einfache Aufgabe. Es bietet sich daher an, statt einer solchen binären Klassifikation eine stetige Variable auf einem Intervall $[-\omega, \omega]$ einzuführen, um die Polarität abzubilden, wie in Abbildung 6 dargestellt.

[34] Eigene Darstellung.

[35] Eigene Darstellung.

Abbildung 6.: Skala für eine stetige Polaritätsvariable[35].

Die Sentiment-Analyse kann damit formal definiert werden:

Definition 3.4: Sentiment-Analyse

Eine Sentiment-Analyse A für einen Text T eines Textkorpus Σ^* ist eine Abbildung

$$A : T \to [-\omega,\ \omega]\ ,\ \ T \in \Sigma^*,$$

welche die Polarität des Textes T widerspiegelt.

Aus der Definition erkennt man, dass A nur die Polarität der Texte analysiert. Eine komplexere Analyse unter Einbeziehung der Subjektivität eines Textes ist jedoch ebenso denkbar, wenn man die Zielmenge der Abbildung entsprechend definiert, also:

$$A : T \to ([-\omega,\ \omega] \times [-s,\ s]) \tag{3.1}$$

mit einer Skala $[-s,s]$ für die Subjektivität. Im Folgenden wird ω als Variable für einen Polaritätswert verwendet.

Um eine solche Abbildung A zu finden gibt es zahlreiche Ansätze. In jüngerer Zeit werden vermehrt Methoden des maschinellen Lernens verwendet, um Texte zu analysieren[36]. „Klassische“ Methoden verwenden hingegen meist ein Lexikon mit besonderen „emotionalen Wörtern“, also Wörtern, die eindeutig einem Polaritätswert ω zugeordnet werden können. Beide Verfahren erreichen in Vergleichsstudien[37] ähnliche durchschnittliche Genauigkeitswerte, jedoch hängt die Genauigkeit stark vom speziellen Anwendungsfall ab. Im folgenden Abschnitt wird ein Ansatz für ein solches *lexikonbasiertes Verfahren* vorgestellt.

[36] Vgl. Boiy und Moens (2009).

[37] Vgl. Dhaoui, Webster und Tan (2017), S. 14 und Kolchyna u. a. (2015), S. 11 ff.

Lexikonbasierte Verfahren

Lexikonbasierte Verfahren ermitteln das Sentiment eines Textes unter Verwendung eines **Lexikons** mit Lemmata und deren zugehörigen Polaritätswerten, die unter Verwendung eines **Regelwerkes** angewendet werden. Schließlich wird für den Text ein **Gesamtsentiment** aus den einzelnen Sentiments ermittelt.

Das Lexikon

Lexikonbasierte Verfahren berechnen ein Gesamtsentiment eines Textes auf Basis der einzelnen Lemmata der Wörter im Text[38]. Das dafür verwendete Lexikon kann manuell oder automatisch erzeugt werden[39]. Bei der manuellen Erzeugung werden in Laborexperimenten von zuvor geschulten Teilnehmern Wörter auf einer Skala (z. B. von -5 bis +5) bewertet[40]. Die so ermittelten Polaritätswerte jedes Wortes werden unter dem entsprechendem Lemma zusammengeführt. Dabei werden neben den Mittelwerten oft auch die Standardabweichungen der Bewertungen im Lexikon gespeichert. Da die Genauigkeit der Analyse sehr stark von der Qualität des verwendeten Lexikons abhängt, findet oft noch eine Überprüfung des gesamten Lexikons durch eine Gruppe von Experten statt, um die Subjektivität in den Daten zu minimieren[41]. Abbildung 7 zeigt einen Auszug aus dem Lexikon des *VADER*-SentimentAnalyzers, der in der Implementierung verwendet wurde.

Automatisch erzeugte Lexika verwenden oft einen Kern sog. „seed words“, um daraus ein Lexikon mit positiven bzw. negativen Wörtern aufzubauen. Dafür werden große Textkorpora analysiert und Worte aufgrund der Häufigkeit des gemeinsamen Auftretens (der sog. „Kookurrenz“) mit den bekannten seed words kategorisiert und dem Lexikon hinzugefügt[42].

[38] Vgl. Turney (2002).

[39] Vgl. Stone (1966).

[40] Vgl. Dodds und Danforth (2010), S. 442f.

[41] Vgl. Taboada u. a. (2011).

[42] Vgl. Kanayama und Nasukawa (2006).

[43] In Anlehnung an C. J. Hutto und Gilbert (2014).

lemma	avg	st.dev	values
⋮	⋮	⋮	⋮
hell	-3.6	0.66332	[-4, -4, -4, -4, -4, -2, -3, -4, -3, -4]
help	1.7	0.78102	[3, 2, 1, 2, 1, 2, 3, 1, 1, 1]
hero	2.6	0.8	[2, 3, 2, 2, 4, 4, 2, 3, 2, 2]
heroic	2.6	0.8	[3, 3, 1, 4, 2, 3, 2, 3, 2, 3]
⋮	⋮	⋮	⋮

Abbildung 7.: Auszug aus einem typischen Lexikon, wie es in lexikonbasierten Sentiment-Analyzern verwendet wird. Die Spalte *values* enthält die Beurteilungen der Testpersonen auf einer Skale von $+5$ bis -5[43].

Das Regelwerk

Der zweite Baustein eines lexikonbasierten Verfahrens ist das Regelwerk. Es besteht aus verschiedenen Regeln, die z. B. **Verstärkungen** oder **Negationen** bei der Bewertung eines Textes berücksichtigen.

Ein **Verstärker** verändert die Polarität des Lemmas, auf das er sich bezieht. Meist wird dies durch eine multiplikative Konstante umgesetzt. Das gleiche Schema kann auch auf Adjektiv-Substantiv-Kombinationen angewendet werden. So ist z. B. „Problem“ weniger negativ konnotiert als „riesiges Problem“, was ebenfalls mit einer multiplikativen Konstante dargestellt wird. Es gibt jedoch auch Modelle, in denen statt einer multiplikativen eine additive Konstante verwendet wird.

Während Verstärker oft in unmittelbarer Nähe ihres Bezugswortes stehen, können **Negationen** auch weiter entfernt stehen (z. B. „Nobody gave a good performance in this movie.“). Die richtige computerlinguistische Verarbeitung des Textes ist daher entscheidend, um eine Phrase zu identifizieren und analysieren zu können[44]. Wenn eine Phrase mit Negation erfolgreich erkannt wurde, muss der Polaritätswert des negierten Lemmas entsprechend angepasst werden. Ähnlich wie bei der Verarbeitung der Modifikationen stehen auch hier verschiedene Optionen zur Verfügung:

- **switch negation**: Der Polaritätswert des Lemmas wird umgekehrt, indem er mit -1 multipliziert wird

[44] Vgl. Taboada u. a. (2011), S. 276.

- **shift negation**: Der Polaritätswert wird in Richtung der entgegengesetzten Polarität verschoben, indem z. B. $+4$ (bei einem Lemma mit negativer Polarität) addiert wird

Das Beispiel 3.2 zeigt einige Anwendungen von verschiedenen Verstärkungen und Negationen.

Beispiel 3.2: Verstärkungen und Negationen

Verstärkungen:

„good“	$+2$
„very good“	$+2 \cdot 1,5 = +3$
„somewhat good“	$+2 \cdot 0,8 = +1,6$

Durch die Verstärkung „very“ wird das Lemma „good“ verstärkt, indem die für den Verstärker „very“ festgelegte multiplikative Konstante angewendet wird. Der Verstärker (bzw. in diesem Fall „Abschwächer“) „somewhat“ wird analog angewendet mit einer multiplikativen Konstante, die kleiner als 1 ist.

Negationen:

„good“	$+2$	
„not good“	$+2 \cdot (-1) = -2$	*switch negation*
„not good“	$+2 - 3 = -1$	*shift negation*

Die switch negation invertiert die Polarität des Lemmas, während bei der shift negation in entgegengesetzter Richtung der Polarität eine Konstante addiert wird. Vorteilhaft bei der shift negation ist z. B. die Stabilität bei mehrfacher Anwendung einer Negation. „Not not good“ erhält unter der switch negation den gleichen Polaritätswert wie „good“, was der menschlichen Wahrnehmung dieser Phrase entspricht.

Neben einem Lexikon mit Polaritätswerten für die einzelnen Lemmata müssen also auch Lexika für die Verstärker und die Negationswörter und deren zugehörigen Anpassungsregeln, d.h. additive oder multiplikative Kontanten, erstellt werden. Die Wahl des richtigen Regelwerkes kann dabei sehr starke Auswirkungen auf die Genauigkeit der Sentiment-Analyse haben und die richtige Auswahl hängt vom spezifischen Anwendungsfall ab[45].

[45] Vgl. Liu und Seneff (2009), S. 162.

Das Gesamtsentiment

Um mithilfe des vorgestellten Lexikons und des Regelwerks ein Gesamtsentiment für einen Text ermitteln zu können, muss dieser zunächst wie in Abschnitt 3.1 beschrieben vorverarbeitet werden. Anschließend kann die in Definition 3.4 eingeführte Sentiment-Analyse A auf dem Text durchgeführt werden:

$$A(T) = \sum_{p \in P} A(p) \tag{3.2}$$

wobei P die Menge aller Phrasen des Textes T ist. Die Sentiment-Analyse einer Phrase $A(p)$ kann unter Verwendung von Lexikon und Regelwerk für den Fall multiplikativer Konstanten für Negationen und Verstärker ausgedrückt werden durch:

$$A(p) = \begin{cases} \sum_{i=1}^{n} \omega_i \cdot \prod_j n_{i,j} \prod_k m_{i,k} & , n \geq 0 \\ 0 & , n = 0 \end{cases} \tag{3.3}$$

mit

- den Polaritätswerten ω_i aller Lemmata
- den zu ω_i zugehörigen Negationswörtern $n_{i,j}$
- den zu ω_i zugehörigen Verstärkern $m_{i,k}$ und
- der Anzahl der in p enthaltenen Phrasen n

Bei der Analyse einer Phrase ist es entscheidend, die richtige Ebene der Phrase für die Analyse zu verwenden. Im Baumdiagramm in Abbildung 5 sind beispielsweise mehrere Ebenen von Phrasen für die Analyse verfügbar. Da jedes Lemma nur einfach in das Gesamtsentiment einfließen soll, empfiehlt sich ein einfacher rekursiver Algorithmus für die Sentiment-Analyse einer Phrase:
Mithilfe des Algorithmus in Quellcode 3.1 wird die relevante Phrase ermittelt. Wenn die „Eltern-Phrase“ p', d.h. die Phrase, die im Baumdiagramm über dem Element p steht, einen Verstärker, eine Negation oder ein Lemma enthält, dann wird für die Sentiment-Analyse die Eltern-Phrase verwendet. Durch die Rekursion wird der Baum solange nach oben durchlaufen bis durch ein weiteres Aufsteigen keine Phrasen mit zusätzlichem Informationsgehalt mehr gefunden werden können.

```
1  def recursive(phrase):
2  For alle Phrasen $p$:
3    Ermittle parent p' zu p
4    If p'\p enthaelt Negation, Verstaerker oder Lemmata
5    Then:
6      phrase = recursive(phrase)
7      return(phrase)
8    Else:
9      return(phrase)
```

Quellcode 3.1: Rekursiver Algorithmus zum Ermitteln der relevanten Phrasenebene

3.3. Datenerhebung

In den vergangenen Kapiteln wurden die Grundlagen für die Textverarbeitung und Sentiment-Analyse (beliebiger) Texte vorgestellt. Für die praktische Anwendung müssen nun zunächst Daten beschafft und analysiert werden.

Wahl eines sozialen Mediums

Die Kommunikation im Internet hat in den letzten Jahren stetig zugenommen. Dabei greifen immer mehr Nutzer auf sogenannte Microblogging-Dienste wie Twitter, Tumblr oder Facebook zurück, um ihre Meinung zu äußern, statt traditionelle internetbasierte Kommunikationsmittel wie E-Mails oder klassische Blogs zu verwenden. Twitter macht dabei mit 330 Millionen aktiven Nutzern[46] und durchschnittlich 500 Mio. Tweets pro Tag einen bedeutenden Anteil dieser Kommunikation aus. Die Wahl von Twitter als Datenquelle für die Analyse ist somit naheliegend. Obwohl die Anzahl der Zeichen je Tweet auf 140[47] Zeichen beschränkt ist, liefert Twitter zu nahezu jedem Thema eine große Anzahl an Tweets mit persönlichen Meinungen und ermöglicht daher eine gute Repräsentation der Stimmungslage. Die Verwendung von *hashtags* ermöglicht darüber hinaus eine einfache Gruppierung von Tweets zu einem bestimmten Thema.

Twitter-API

Daten können von Twitter durch die Twitter-Programmierschnittstelle (engl. *application programming interface, API*) abgerufen werden. Twitter stellt dabei neben einer Streaming-API

[46] Vgl. Statistisches Bundesamt (2019).

[47] Im November 2017 wurde die maximale Anzahl der Zeichen von 140 auf 280 Zeichen erhöht.

für Echtzeitdaten auch die hier verwendete REST[48]-API für historische Daten zur Verfügung[49]. Teil dieser REST-API ist die *Search API*, mit der die historischen Tweets abgerufen werden können. An diese API kann ein HTTP[50]-Request gesendet werden. Dieser muss neben den Zugangsdaten - Twitter nutzt das Open Authorization (OAuth)-Protokoll für die Autorisierung - eine gültige Query enthalten. Diese besteht aus den verschiedenen Suchparametern (Datum bzw. Datumsbereich, Suchbegriff, Hashtag, Nutzername etc.), nach denen die Tweets gefiltert werden sollen. Die HTTP-Response enthält schließlich alle der Query entsprechenden Tweets im JSON-Format[51].

Speicherung der Daten

Die Twitter-API stellt eine Sammlung von Tweets im JSON-Format zur Verfügung. Die Struktur dieses Dateityps erlaubt eine einfach lesbare Textform bei gleichzeitiger starker Komplexität der Daten, indem mittels Verschachtelungen Werte zu Gruppen zusammengefasst oder durch Verwendung von Arrays auch umfangreiche Datenmengen gespeichert werden können. Ein einzelner Tweet, wie er von der API zurückgegeben wird, enthält viele Informationen, die für die weitere Analyse nicht notwendig sind. Um den Speicherbedarf zu reduzieren werden überflüssige Informationen daher zunächst entfernt. Dadurch kann der Speicherbedarf eines einzelnen Tweets um mehr als 90% reduziert werden (von durchschnittlich $4,2$kB auf ca. 400kb).

```
{
  'tweets': [
    {
      'screen_name' : 'Dr. Malte Kaufmann',
      'user_name' : '@MalteKaufmann',
      'date':  '2019-06-24',
      'text': 'Ein schwarzer Tag für die #Bundeswehr. Ein schwarzer Tag für DE. Wir trauern um den verunglückten
               Kameraden. Herzliches Beileid an die Hinterbliebenen ! 😢😢😢\n#Eurofighter @bundeswehrInfo
               @Team_Luftwaffe https://t.co/06oZSS0V0Q',
      'reactions': 'reactions:12/39/163'
    },
    {
      [...]
    },
}
```

Abbildung 8.: Beispiel eines Tweets im reduzierten Format. Die überflüssigen Meta-Informationen (z. B. Betriebssystem des Nutzers oder sein Profilbild) wurden entfernt[52].

[48] Representational State Transfer (REST)

[49] Vgl. Pfaffenberger (2016), S. 54f.

[50] Hypertext Transfer Protocol

[51] JavaScript Object Notation (JSON)

Für die Speicherung der Daten wird eine SQLite-Datenbank verwendet. In Abbildung 9 ist das Datenmodell für die Speicherung der Tweets dargestellt. Die Abbildung zeigt auch die übrigen Tabellen der Datenbank für die Speicherung der einzelnen Suchanfragen, die an die API gesendet wurden, und für die Speicherung der Aktienkurse für die spätere Regressionsanalyse. Die Datenbank und die im Datenmodell beschriebenen Tabellen können mit den Funktionen aus Anhang B.1 erzeugt werden.

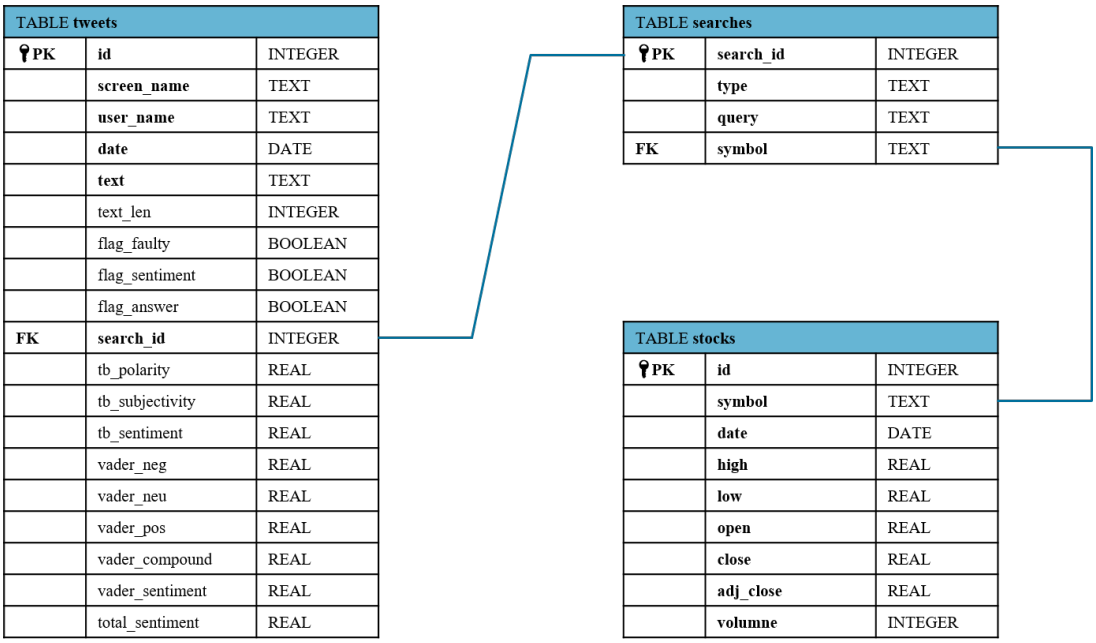

Abbildung 9.: Datenbank-Modell zur Speicherung der Daten für die Regressionsanalyse[53]

3.4. Implementierung

Die Implementierung erfolgt in der Programmiersprache *Python*. Die Schritte des NLP wurden dabei mit dem Paket *NLTK* umgesetzt. Die Sentiment-Analyse basiert auf dem Paket *VADER*.

[52]Eigene Darstellung.

[53]Eigene Darstellung.

Natural Language Toolkit (NLTK)

Tokenisierung

Der Tokenizer des NLTK-Paketes arbeitet wie in Abschnitt 3.1 beschrieben in zwei Stufen. Die Funktion *word_tokenize* verwendet zunächst die Funktion *sent_tokenize*, um einen Text in einzelne Sätze zu zerteilen. Anschließend wird jeder Satz mit *word_tokenize* in die einzelnen Tokens zerlegt. Abbildung 3.2 zeigt einen exemplarischen Aufruf der beiden Funktionen.

```
1 >>> from nltk.tokenize import sent_tokenize, word_tokenize
2 >>> text = 'But it was alright, everything was alright, the struggle was finished. He
      had won the victory over himself. He loved Big Brother.'
3 >>> sentences = sent_tokenize(text)
4 ['But it was alright, everything was alright, the struggle was finished.', 'He had won
      the victory over himself.', 'He loved Big Brother.']
5 >>> tokens = [word_tokenize(s) for s in sentences]
6 [['But', 'it', 'was', 'alright', ',', 'everything', 'was', 'alright', ',', 'the',
      'struggle', 'was', 'finished', '.'], ['He', 'had', 'won', 'the', 'victory', 'over',
      'himself', '.'], ['He', 'loved', 'Big', 'Brother', '.']]
```

Quellcode 3.2: Tokenisierungs-Funktionen des NLTK-Paketes

Der auf dem Perzeptron-Algorithmus basierende POS-Tagger kann mit der Funktion *pos_tag* aufgerufen werden. Dafür muss der Text bereits in tokenisierter Form vorliegen.

```
1 >>> from nltk.tokenize import sent_tokenize, word_tokenize
2 >>> text = 'But it was alright, everything was alright, the struggle was finished. He
      had won the victory over himself. He loved Big Brother.'
3 >>> sentences = sent_tokenize(text)
4 ['But it was alright, everything was alright, the struggle was finished.', 'He had won
      the victory over himself.', 'He loved Big Brother.']
5 >>> tokens = [word_tokenize(s) for s in sentences]
6 [['But', 'it', 'was', 'alright', ',', 'everything', 'was', 'alright', ',', 'the',
      'struggle', 'was', 'finished', '.'], ['He', 'had', 'won', 'the', 'victory', 'over',
      'himself', '.'], ['He', 'loved', 'Big', 'Brother', '.']]
```

Quellcode 3.3: POS-Tagger des NLTK-Paketes

Morphologische Analyse

Der WordNet-Lemmatizer des NLTK-Paketes verwendet das Lexikon *WordNet* und kann wie in Quellcode 3.4 dargestellt aufgerufen werden.

```
>>> from nltk.stem import WordNetLemmatizer
>>> wnl = WordNetLemmatizer()
>>> tokens = ['better', 'corpora', 'books']
>>> lemmata = [wnl.lemmatize(t) for t in tokens]
['good', 'corpus', 'book']
```

Quellcode 3.4: Lemmatisierer des NLTK-Paketes auf Basis von WordNet

Syntaktische Analyse

Das im folgenden Abschnitt verwendete Paket zur Sentiment-Analyse verwendet im Gegensatz zu anderen Modellen einen in die Sentiment-Analyse integrierten Algorithmus zur syntaktischen Analyse. Auf eine gesonderte syntaktische Analyse kann daher verzichtet werden.

VADER

Mit dem ***V**alence **A**ware **D**ictionary and s**E**ntiment **R**easoner* (VADER) existiert ein lexikonbasiertes Python-Paket für die Sentiment-Analyse, das im Vergleich zu anderen Paketen über ein sehr komplexes Regelwerk verfügt. Es wurde speziell für die Analyse von Texten aus sozialen Netzwerken entwickelt. Die Entwickler von VADER, Hutto und Gilbert, konnten zeigen, dass ihre Kombination von Lexikon und Regeln auf Texten aus sozialen Netzwerken besonders präzise arbeitet und sogar „menschliche Klassifikatoren“ bei der binären Klassifikation von Tweets übertrifft[54].

Um das Regelwerk aufzustellen, wurden zunächst von zwei Linguistik-Experten ein Korpus von 800 Tweets manuell analysiert, wobei jedem Tweet ein Sentiment von -4 bis $+4$ zugewiesen wurden. Anschließend wurden aus den Daten fünf Heuristiken abgeleitet:

1. Interpunktion (insbesondere das Ausrufezeichen) hat einen starken Einfluss auf die Intensität des Sentiment

2. Großschreibung ganzer Wörter verstärkt die Intensität eines Sentiments

[54] Vgl. C. J. Hutto und Gilbert (2014), S. 8.

3. Verstärker verstärken die Intensität des Sentiments

4. Konjunktionen wie „aber“ signalisieren einen Wechsel der Stimmung, wobei das Gesamtsentiment maßgeblich vom der Konjunktion folgenden Satz abhängt

5. Ein Großteil (ca. 90%) der negierten Phrasen kann durch das Trigramm[55] identifiziert werden, das einem Lemma vorausgeht

Die so gewonnenen Regeln wurden anschließend getestet, indem 20 Testpersonen die zuvor von Experten analysierten Tweets mit leichten Modifikationen (z. B. mit veränderter Interpunktion) vorgelegt wurden. Aufgrund der so gewonnenen Daten konnten die Regeln quantifiziert und in das Regelwerk implementiert werden.

Das von VADER verwendete Lexikon basiert auf zahlreichen bereits bestehenden Lexika. Dabei wurden ca. 7.500 Lemmata identifiziert, die besonders für den Bereich von Texten aus sozialen Netzwerken geeignet sind. Darunter befinden sich auch zahlreiche Worte aus dem „Netzjargon“, (z. B. „LOL“, „ROFL“) sowie Emoticons. Zusammen mit dem aufgestellten Regelwerk konnte so ein Modell für die Sentiment-Analyse entwickelt werden.

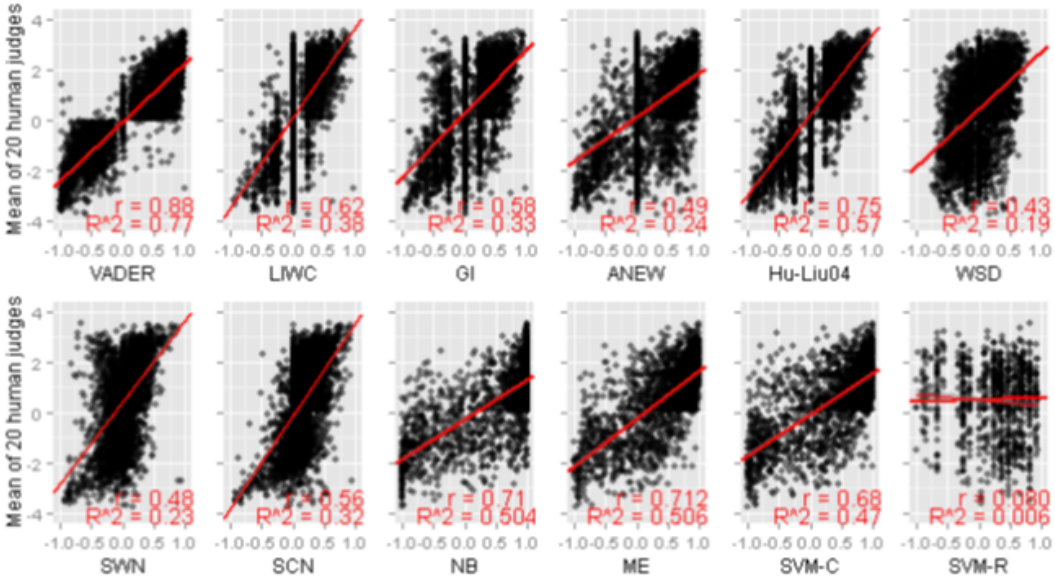

Abbildung 10.: Sentiment-Werte verschiedener Sentiment-Analyse-Tools[56]

[55] Als Trigramm wird ein Satzfragment von drei aufeinanderfolgenden Wörtern bezeichnet.

[56] Aus: C. J. Hutto und Gilbert (2014), S. 8.

Dieses Modell wurde anschließend mit 11 anderen Sentiment-Analyse-Modellen auf einen Korpus von ca. 4.000 Tweets angewendet und die ermittelten Sentiments mit denen der menschlichen Testpersonen verglichen. Abbildung 10 zeigt die hohe Genauigkeit von VADER im Vergleich zu anderen Modellen. Insbesondere der geringe Anteil von falsch negativen (linker oberer Quadrant im Diagramm) und falsch positiven (rechter unterer Quadrant im Diagramm) Klassifikationen fällt dabei auf. Zum Vergleich der Gesamtgenauigkeit der Modelle wurde das *F-Maß*[57] verwendet. Bei der Analyse von Tweets konnte VADER mit einem F_1-Maß von $F_{1,VADER} = 0,96$ alle anderen getesteten Modelle sowie individuelle menschliche Testpersonen ($F_{1,human} = 0,84$) übertreffen.

In Python ist das gesamte VADER-Modell in einem Paket implementiert[58]. Mit der *SentimentIntensityAnalyzer*-Klasse kann eine Phrase durch den Aufruf der Funktion *polarity_scores* (wie in Quellcode 3.5 dargestellt) analysiert werden. Für die Analyse größerer Mengen von Tweets findet sich in Anhang B.7 eine Funktion, die eine Menge von Tweets abruft, analysiert und an die Datenbank zurückgibt.

```
>>> from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
>>> sentence = "The book was good."
>>> analyzer = SentimentIntensityAnalyzer()
>>> analyzer.polarity_scores(sentence)
{'pos': 0.492, 'compound': 0.4404, 'neu': 0.508, 'neg': 0.0}
```

Quellcode 3.5: Python: Sentiment-Analyse des VADER-Paketes

3.5. Ergebnis

Mit den in Abschnitt 3.3 vorgestellten Methoden wurde eine Datenbank erzeugt und mit den abgerufenen Tweets gefüllt. Dabei wurden exemplarisch verschiedene Unternehmen in unterschiedlichen Zeiträumen betrachtet. Tabelle 3.1 enthält eine Übersicht der erzeugten Datensätze. Insgesamt wurden 343.674 Tweets abgerufen, von denen 212.599 analysiert und verwendet werden konnten.

Aus diesen Daten muss nun eine Zeitreihe erzeugt werden, die in der Regressionsanalyse mit

[57] Definition in Anhang A.1.

[58] Vgl. C. Hutto und Gilbert (2019).

[59] Eigene Darstellung.

Datensatz	Query	Zeitraum	Tweets
TESLA1819	#tesla,@tesla	01.01.2018 - 31.12.2019	78.489
MSFT18	#microsoft,@microsoft	01.01.2018 - 31.12.2018	29.545
SIEMENS19	#siemens,@siemens,siemens	01.01.2019 - 31.12.2019	21.987
NESTLE19	#nestle,@nestle	01.01.2019 - 31.12.2019	62.795
DBK19	#DeutscheBank,@DeutscheBank	01.01.2019 - 31.12.2019	19.783
EXXON18	#Exxon,@exxonmobil,exxon	01.01.2018 - 31.12.2018	22.215

Tabelle 3.1.: Übersicht über die erzeugten Datensätze[59]

den Aktienkursen verglichen werden kann. Da die Aktienkurse tagesweise verfügbar sind, liegt es nahe, die Sentiment-Daten ebenfalls tagesweise zu aggregieren. Für jeden Tag eines Zeitraums wird daher der Mittelwert der Sentiments aller Tweets ermittelt. Dabei werden als fehlerhaft klassifizierte Tweets nicht berücksichtigt. Ebenso werden Tweets mit einem Sentiment von 0 nicht einbezogen, da bei diesen in der Regel kein richtiges Sentiment ermittelt werden konnte (z. B. da keine Lemmata mit Polaritätswerten vorhanden waren). Die Aggregation der Daten auf Tagesebene und die Verknüpfung mit den Aktienkursen ist in Quellcode B.6 umgesetzt.

Abbildungen 11 und 12 zeigen die Zeitreihen von zwei der erzeugten Datensätze. Man erkennt die unterschiedlichen Niveaus des Sentiments ($\mu_{sent}(\text{TSLA}) = 0,28$, $\mu_{sent}(\text{XOM}) = 0,09$), d.h. die Tweets über Tesla sind durchschnittlich deutlich positiver als die Tweets über Exxon Mobil. Insbesondere bei den Ausreißern in den Zeitreihen lässt sich durch bloße Betrachtung eine Korrelation vermuten, z.B. bei Exxon Mobil im Februar 2018. Die stark gestiegene Anzahl der Tweets ging hier mit einem steigenden Sentiment und einem fallenden Aktienkurs einher.

Die Visualisierung der Zeitreihen und die Ermittlung einiger statistischer Werte kann mit der Funktion *get_dataset_statistics* erzeugt werden, die in B.10 beschrieben ist.

[60] Eigene Darstellung.

[61] Eigene Darstellung.

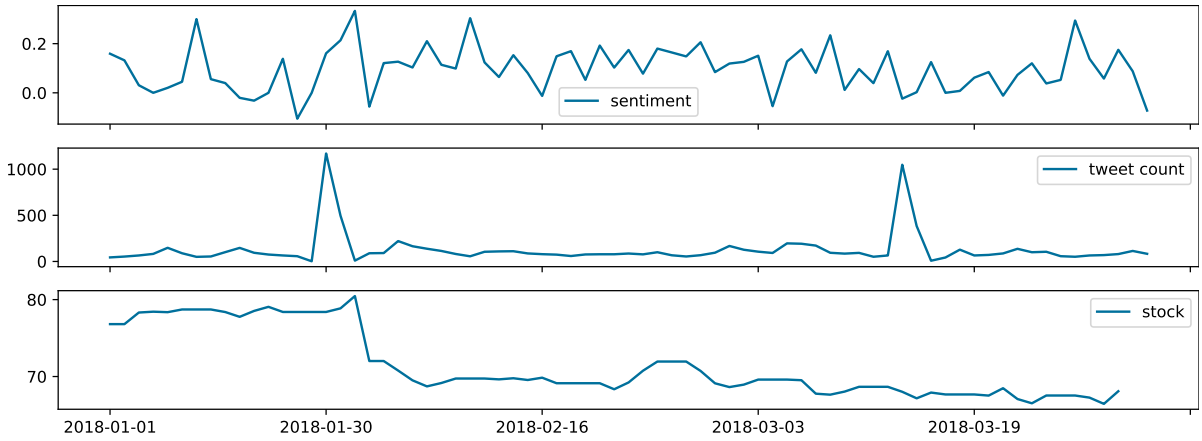

Abbildung 11.: Zeitreihen der Exxon Mobil (Sentiment-Werte, Anzahl der Tweets, Aktienkurs) über einen Zeitraum von 3 Monaten[60].

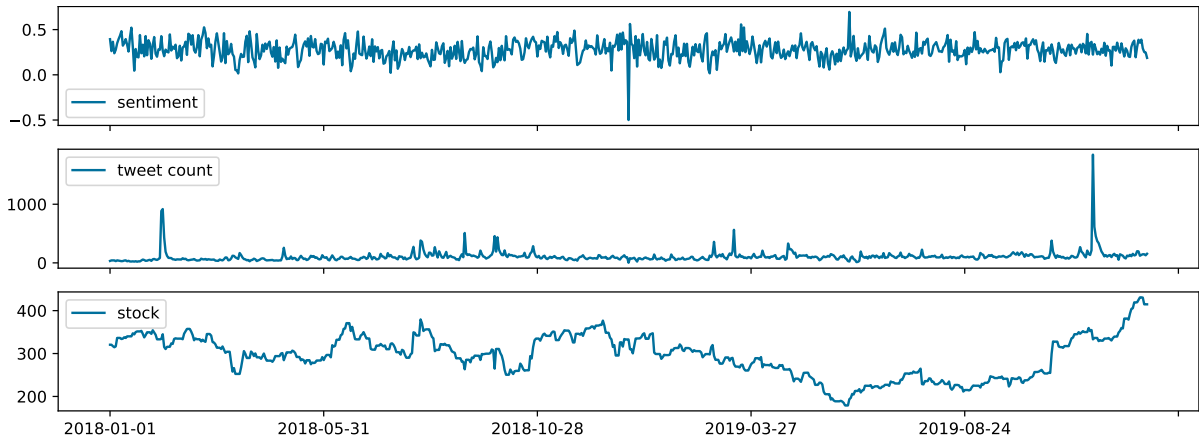

Abbildung 12.: Zeitreihen der Tesla, Inc. (Sentiment-Werte, Anzahl der Tweets, Aktienkurs) über einen Zeitraum von 2 Jahren.[61]

4. Regressionsanalyse und Prognose von Kursentwicklungen

Bei einfacheren Regressionen (z. B. linearen Regressionen) wird von vornherein eine Unterscheidung zwischen endogenen und exogenen Variablen getroffen. Ein vektorautoregressives Modell (VAR) hingegen trifft diese Unterscheidung nicht – alle Variablen werden als abhängige (endogene) Variablen gleich behandelt. Ähnlich wie bei univariaten autoregressiven Zeitreihen wird eine Variable von ihren Vergangenheitswerten beeinflusst. Es fließen jedoch auch die vergangenen Werte aller anderen Variablen (also ein Vektor von Variablen) in das System ein. Dies ermöglicht nicht nur eine gegenseitige Beeinflussung der Variablen, es können mit dem VAR-Modell auch zahlreiche strukturelle Analysen der Daten durchgeführt werden. So können z. B. Granger-Kausalitätstests zur Untersuchung zeitlicher Kausalitäten zwischen verschiedenen Variablen durchgeführt werden oder Zerlegungen der Prognosefehler-Varianz, um den Anteil einer Variable am Prognosefehler zu identifizieren. Ein weiterer großer Vorteil des VAR-Modells liegt darin, dass es „theoriefrei“ ist und sich somit alleine auf Daten stützt und keine bzw. wenige Restriktionen beinhaltet.[62].

In diesem Kapitel werden zunächst grundlegende Eigenschaften vektorautoregressiver Modelle vorgestellt. Anschließend werden Stationaritätstests, die Auswahl der richtigen Lag-Ordnung und Schätzer für die Parameter behandelt. Für das theoretische Modell wird schließlich eine Methode zum Trainieren und zur Bewertung der Prognosegüte erarbeitet. Den Abschluss bildet die Implementierung der vorgestellten Theorie in der Programmiersprache *Python*.

4.1. Vektorautoregressive Modelle

In einem einfachen autoregressiven Modell ($AR(p)$-Modell) hängen alle Werte der Zeitreihe $(y_t)_{t\in T}$ von den vergangenen Werten und einem weißen Rauschen u_t ab, d.h.

$$y_t = \nu + \alpha_1 y_{t-1} + \ldots + \alpha_p y_{t-p} + u_t \tag{4.1}$$

[62] Vgl. Sims (1980), S. 27.

Eine multivariate Erweiterung dieser Zeitreihe führt zu

$$y_{k,t} = \nu + \overbrace{\alpha_{k\ 1,1} y_{1,t-1}}^{\text{Var. 1, Lag 1}} + \overbrace{\alpha_{k\ 2,1} y_{2,t-1}}^{\text{Var. 2, Lag 1}} + \ldots + \overbrace{\alpha_{k\ K,1} y_{K,t-1}}^{\text{Var. k, Lag 1}} \tag{4.2}$$

$$+ \ldots + \underbrace{\alpha_{k\ 1,p} y_{1,t-p}}_{\text{Var 1, Lag p}} + \ldots + \underbrace{\alpha_{k\ K,p} y_{K,t-p}}_{\text{Var k, Lag p}} \tag{4.3}$$

In der Matrixschreibweise lässt sich dieser Prozess kompakt darstellen:

Definition 4.1: $VAR(p)$**-Prozess**[63]

Ein $VAR(p)$-Prozess ist ein Prozess $(y_t)_{t\in\mathbf{T}}$ mit

$$y_t = \nu + A_1 y_{t-1} + \ldots + A_p y_{t-p} + u_t, \quad t = 0, \pm 1, \pm 2 \tag{4.4}$$

mit

- einem $(K \times 1)$-dimensionalem Zufallsvektor $y_t = (y_{1t}, \ldots, y_{Kt})^T$,
- $(K \times K)$-Koeffizientenmatrizen A_i,
- einem $(K \times 1)$-Vektor mit Regressionskonstanten $\nu = (\nu_1, \ldots, \nu_K)^T$,
- und einem K-dimensionalem weißen Rauschen $u_t = (u_{1t}, \ldots, u_{Kt})^T$ mit $\mathbb{E}(u_t) = 0$, $\mathbb{E}(u_t u_t^T) = \Sigma_u$ und $\mathbb{E}(u_t u_s^T) = 0$ für $s \neq t$

Dabei ist die Kovarianzmatrix Σ_u nichtsingulär.

4.2. Stationarität

Für ein Prognosemodell ist es wichtig, dass sich gewisse Eigenschaften (wie z. B. Erwartungswert und Varianz) der beobachteten Zeitreihe im Laufe der Zeit nicht ändern. Im folgenden Abschnitt werden daher zunächst die grundlegenden Eigenschaften der **Stabilität** und **Stationarität** einer Zeitreihe untersucht.

[63] Vgl. Lütkepohl (2005), S. 13.

Definition 4.2: Stabilität[64]

Eine Zeitreihe $(y_t)_{t\in\mathbb{T}}$ ist **stabil**, wenn die zugehörige Verteilung eine α-stabile Verteilung ist.

Definition 4.3: Stationarität

Eine Zeitreihe $(y_t)_{t\in\mathbb{T}}$ ist (schwach) **stationär**, wenn

1. der Erwartungswert konstant ist: $\mathbb{E}[y_t] = \mu$
2. die Varianz endlich ist: $Var(y_t) < \infty\,, \quad \forall t \in \mathbb{T}$
3. die Autokovarianzfunktion nicht von t abhängt:
   $Cov(y_{t_1}, y_{t_2}) = Cov(y_{t_1+h}, y_{t_2+h}) = \gamma(h)\,, \quad \forall h, t_1, t_2 \in \mathbb{T}$

Betrachtet man einen $VAR(1)$-Prozess $y_t = \nu + A_1 y_{t-1} + u_t$ erkennt man, dass die Vektoren $y_1, \ldots, y_t$ nur von $y_0, u_1, \ldots, u_t$ abhängen. Geht man von einer „unendlichen Vergangenheit“ aus, dann hängen die Vektoren y_t ausschließlich vom weißen Rauschen u_t und somit von einer α-stabilen Verteilung ab. Ein einfaches Kriterium für die Stabilität des $VAR(1)$-Prozesses erhalten wir aus der Rekursionsgleichung. Durch rekursives Einsetzen[65] kann der Prozess für y_t auch dargestellt werden als

$$y_t = (I_K + A_1 + \ldots + A_1^j)\nu + A_1^{j+1} y_{t-j-1} + \sum_{i=0}^{j} A_1^i u_{t-i} \tag{4.5}$$

Wenn nun alle Eigenwerte von A_1 betraglich kleiner als 1 sind, dann ist die Folge A_1^i, $i = 0, 1, \ldots$ absolut summierbar[66] und somit existiert auch die unendliche Summe

$$\sum_{i=0}^{\infty} A_1^i u_{t-i} \tag{4.6}$$

und der $VAR(1)$-Prozess kann als

$$y_t = \mu + \sum_{i=0}^{\infty} A_1^i u_{t-i}\,, \quad t = 0, \pm 1, \pm 2, \ldots \tag{4.7}$$

[64] Vgl. Ito (2006), S. 50 ff.

[65] siehe Anhang A.4.

[66] Vgl. Lütkepohl (2005), S. 656 f.

geschrieben werden. Die ersten beiden Momente ergeben sich daraus mit[67]

$$\mathbb{E}[y_t] = \mu \ , \quad \forall t \tag{4.8}$$

und

$$\Gamma_y(h) = \mathbb{E}\left[(y_t - \mu)(y_{t-h} - \mu)^T\right] \tag{4.9}$$

$$= \sum_{i=0}^{\infty} A_1^{h+i} \Sigma_u {A_1^i}^T \tag{4.10}$$

Da dies aber nur unter der Voraussetzung, dass die Eigenwerte von A_1 betraglich kleiner als 1 sind, möglich ist, erhalten wir als Stabilitätskriterium:

Satz 4.1

Ein $VAR(1)$-Prozess ist stabil, wenn alle Eigenwerte von A_1 betraglich kleiner als 1 sind. Dies ist äquivalent zu

$$det(I_K - A_1 z - \ldots A_p z^p) \neq 0, \quad \forall |z| \leq 1$$

Für $VAR(p)$-Prozesse kann dieses Kriterium einfach erweitert werden, da jeder $VAR(p)$-Prozess auch als $VAR(1)$-Prozess ausgedrückt werden kann[68].

Außerdem besitzt jeder stabile $VAR(p)$-Prozess eine Darstellung als $MA(\infty)$-Prozess in der Form

$$Y_t = \mu + \sum_{i=0}^{\infty} A^i U_{t-i} \tag{4.11}$$

Aus der $MA(\infty)$-Darstellung des Prozesses lassen sich leicht Erwartungswert und Autokovarianz ablesen:

$$\mathbb{E}[y_t] = \mu \ , \tag{4.12}$$

$$\Gamma_y(h) = \mathbb{E}\left[(y_t - \mu)(y_{t-h} - \mu)^T\right] \tag{4.13}$$

$$= \sum_{i=0}^{\infty} \phi_{h+i} \Sigma_u \Phi_i^T \tag{4.14}$$

mit den Koeffizientenmatrizen ϕ_i. Der Prozess hat also einen konstanten Erwartungswert sowie eine von t unabhängige Autokovarianzfunktion. Zusammen mit der endlichen Varianz, die aus der α-stabilen Verteilung folgt, erfüllt ein stabiler Prozess also die Bedingungen der schwachen Stationarität. Damit folgt als Stationaritätsbedingung:

[67] Vgl. Lütkepohl (2005), S. 15, S. 688.

[68] Siehe Anhang A.5.

Satz 4.2: Stationaritätsbedingung

Ein stabiler $VAR(p)$-Prozess y_t, $t = 0, \pm 1, \pm 2, \ldots$ ist (schwach) stationär.

Testen auf Stationarität und Stationarisierung

In der Praxis sind die echten Modellparameter meist unbekannt. Ein Test durch Überprüfen der Stationaritätsbedingung ist daher nicht möglich. Zum Testen einer empirischen Zeitreihe auf Stationarität verwendet man daher in der Regel Tests wie z. B. den **erweiterten Dickey-Fuller-Test** (ADF). Der ADF gehört zur Klasse der Einheitswurzeltests und testet die Hypothese H_0 (der stochastische Prozess hat eine Einheitswurzel und ist somit nicht stationär) gegen die Alternative H_1 (der Prozess hat keine Einheitswurzel)[69].

Sollte der ADF eine Zeitreihe als nichtstationär erkennen, dann kann mittels **Differenzenbildung** ein in der Zeitreihe enthaltener linearer oder polynomialer Trend entfernt werden. Dabei wird der Differenzenfilter p-ter Ordnung $\Delta^p := (1-L)^p$ mit $(1-L)X_t = X_t - X_{t-1}$ auf eine Zeitreihe angewendet, wodurch der Polynomgrad des Trends um 1 reduziert wird. Beispiel 4.1 verdeutlicht die Grundidee an einem Prozess mit linearem Trend.

Beispiel 4.1

Sei $X_t = a_0 + a_1 t + Y_t$ ein Prozess mit linearem Trend. und Y_t ein stationärer Prozess. Anwenden des Differenzenoperators liefert dann

$$\begin{aligned} \Delta X_t &= X_t - X_{t-1} \\ &= (a_0 + a_1 t + Y_t) - (a_0 + a_1(t-1) + Y_{t-1}) \\ &= a_1 + Y_t - Y_{t-1} =: a_1 Y_t' \,, \end{aligned}$$

wobei der Prozess $a_1 + Y_t'$ stationär ist.

Für Prozesse mit polynomialem Trend höherer Ordnung kann durch mehrfaches Differenzieren (Δ^p) eine stationäre Zeitreihe erzeugt werden. Der Vorteil der Trendelimination durch Differenzenbildung liegt darin, dass keine Annahmen an die tatsächliche Struktur des Trends gemacht werden müssen. Für die meisten ökonomischen Zeitreihen reicht eine Differenzenbildung mit

[69] Vgl. Said und Dickey (1984).

$p = 1$ aus, um die Zeitreihe zu stationarisieren.

4.3. Informationskriterien

Da die Ordnung p des $VAR(p)$-Prozesses unbekannt ist, muss diese zunächst ermittelt werden. Für ein Vorhersagemodell ist jedoch die exakte Ordnung des datenerzeugenden Prozesses weniger interessant. Ein gutes Modell für die Vorhersage zu bestimmen reicht vollkommen aus. Dabei muss ein Kompromiss zwischen guter Datenanpassung und geringer Modellkomplexität gefunden werden. Das Risiko eines *Overfittings*, also einer Überanpassung an die Daten und einer damit verbundenen schlechten Prognosefähigkeit, kann reduziert werden, indem man eine höhere Modellkomplexität bestraft. Informationskriterien dürfen jedoch nicht als absolutes Maß für die Güte eines Modells verwendet werden – sie geben nur das Modell an, das unter den Alternativen am besten geeignet ist.

Das Ziel ist eine möglichst präzise Vorhersage des Prozesses, daher ist eine Betrachtung der mittleren quadratischen Abweichung (MSE) der Vorhersage sinnvoll. Akaike entwickelte das erste dieser sog. Informationskriterien, indem er den 1-Schritt-Vorhersagefehler verwendete[70]:

$$\Sigma_{\hat{y}}(1) = \frac{T + Km + 1}{T} \tag{4.15}$$

Dabei ist m die Ordnung des an die Daten angepassten VAR-Prozesses, T die Stichprobengröße und K die Dimension der Zeitreihe, d.h. die Anzahl der Variablen. In der Praxis muss die Kovarianzmatrix Σ_u durch einen entsprechenden Schätzer ersetzt werden. Nach Akaike[71] wird dafür der Kleinste-Quadrate-Schätzer verwendet. Damit erhält man das **Akaike-Informationskriterium** (kurz: AIC):

[70] Vgl. Akaike (1969), S. 245 f.

[71] Vgl. Akaike (1969), S. 246.

Definition 4.4: Akaike-Informationskriterium

Für einen $VAR(m)$-Prozess ist das **Akaike-Informationskriterium**[72] definiert als

$$AIC(m) = ln\left|\tilde{\Sigma}_u(m)\right| + \frac{2mK^2}{T} \tag{4.16}$$

mit

- mK^2 Anzahl der zu schätzenden Parameter
  (m: Lag-Ordnung des $VAR(m)$-Prozesses, K: Anzahl der Variablen)
- Stichprobengröße T

Der Summand $\frac{2mK^2}{T}$ wirkt dabei als Strafterm, der mit steigendem m wächst und für $T \to \infty$ gegen 0 konvergiert. Somit kann nun ein Schätzer $\hat{p}(AIC)$ gewählt werden, der $AIC(m)$ minimiert und eine optimale Ordnung für den VAR-Prozess bestimmt.

Neben dem Akaike-Informationskriterium gibt es auch zahlreiche weitere Informationskriterien, mit denen die Lag-Ordnung ermittelt werden kann, z. B. **final prediction error** (FPE), **Bayessches Informationskriterium** (BIC), **Hannan-Quinn-Informationskriterium** (HQIC). Welches Informationskriterium die genaueste Lag-Ordnung liefert, hängt stark vom Anwendungsfall ab. Bei kleinen Stichprobengrößen liefert BIC in der Regel genauere Ergebnisse, während HQIC bei sehr großen Stichproben das beste Ergebnis liefert. AIC ist den meisten Informationskriterien bei einzelnen Stichprobengrößen unterlegen, ermittelt allerdings durchschnittlich die geringsten Abweichungen der ermittelten Lag-Ordnung zur echten Lag-Ordnung[73].

4.4. Schätzung und Vorhersage von VAR-Modellen

In der allgemeinen Prozessgleichung aus Abschnitt 4.1 sind die Vektoren ν und die Matrizen $A_1, \ldots, A_p \in \mathbb{R}^{n \times n}$ unbekannt und müssen geschätzt werden. Nach dem Satz von Gauß-Markow kann dieses lineare Modell optimal mit der Methode der kleinsten Quadrate geschätzt werden. In der kompakten Matrixschreibweise ist das $VAR(p)$-Modell gegeben durch $Y = BZ + U$ mit

$$Y := (y_1, \ldots, y_T) \qquad B := (\nu, A_1, \ldots, A_p)$$

[72] Vgl. Akaike (1973), S. 199 ff.

[73] Vgl. Shittu (2009), S. 415.

$$Z := (Z_0, \ldots, Z_{T-1}) \qquad Z_t := \begin{bmatrix} 1 \\ y_t \\ \vdots \\ y_{t-p+1} \end{bmatrix} \qquad U := (u_1, \ldots, u_T) \tag{4.17}$$

Mithilfe der Matrixvektorisierung (wie in A.3 beschrieben) erhalten wir:

$$vec(Y) = vec(BZ) + vec(U) = (Z^T \otimes I_K)vec(B) + vec(U) \tag{4.18}$$

und mit $y := vec(Y)$, $\beta := vec(B^T)$, $\upsilon := vec(U)$:

$$y = (Z^T \otimes I_K)\beta + \upsilon \tag{4.19}$$

Die Kovarianzmatrix von υ ist dabei gegeben durc $\Sigma_\upsilon = I_T \otimes \Sigma_u$. Mit dem multivariaten Kleinste-Quadrate-Schätzer kann nun β geschätzt werden:

$$\hat{\beta} = \arg\min_{\beta} (Y - BZ)^T \cdot (I_T \otimes \Sigma_u^{-1}) vec(Y - BZ), \tag{4.20}$$

wobei Σ_u die Kovarianzmatrix von u ist. Der Kleinste-Quadrate-Schätzer für die Regressionskoeffizienten kann dann mit

$$\hat{\beta} = ((ZZ^T)^{-1} Z \otimes I_K) y \tag{4.21}$$

angegeben werden[74]. Da $\beta = vec(B) = (\nu, A_1, \ldots, A_p)$ haben wir damit einen Schätzer für die Koeffizientenmatrizen A sowie für ν. Aufgrund der positiven Definitheit der Hesse-Matrix von $S(\beta)$, $\frac{\delta^2 S}{\delta\beta\delta\beta^T}$ kann nachgewiesen werden, dass der Schätzer $\hat{\beta}$ tatsächlich ein minimaler Vektor ist[75].

Mithilfe des geschätzten VAR-Modells kann eine beste h-Schritt-Vorhersage erzeugt werden durch

$$\hat{y}_t(h) = \hat{\nu} + \hat{A_1} y_t(h-1) + \ldots + \hat{A_p} y_t(h-p) \tag{4.22}$$

mit den entsprechend ermittelten Schätzern $\hat{B} = (\hat{\nu}, \hat{A_1}, \ldots, \hat{A_p})$ [76].

[74] Vgl. Lütkepohl (2005), S. 70 ff.

[75] Für eine vollständige Herleitung: Vgl. Lütkepohl, 2005, S. 35 ff.

[76] Vgl. Lütkepohl (2005), S. 94.

4.5. Implementierung

Die Implementierung der gesamten Regressionsanalyse wurde in der Programmiersprache *Python* ausgeführt. Die Vektorautoregression ist im Paket *statsmodels*[77] implementiert. Der Programmablauf lässt sich grob in fünf Teilschritte einteilen:

1. Datenabruf
2. Datenaufbereitung
3. Modellanpassung & -validierung
4. Vorhersage
5. Bewertung der Prognosegüte

Datenabruf

Die ermittelten Daten (analysierte Tweets sowie Kursdaten) werden, wie in Kapitel 3.3 erläutert, in einer SQLite-Datenbank gespeichert. Die Aggregation der Tweets und das Zusammenführen mit den Kursdaten erfolgt in einer SQLite-Query[78]. Die Daten werden anschließend in einen pandas-Dataframe eingelesen.

Datenaufbereitung

Die eingelesenen Daten werden in die richtigen Datenformate (z. B. das richtige Datumsformat) für die Analyse überführt. Im vorliegenden Fall wird eine tägliche, lückenlose Zeitreihe mit Sentiment-Daten von Twitter mit der von Wochenenden und Feiertagen unterbrochenen Zeitreihe von Aktienkursen verglichen. Die durch die fehlenden Kurse entstehenden Lücken werden dazu mit dem Schlusskurs des letzten Handelstages aufgefüllt. Bei der Prognose von Kursveränderungen wird somit erst eine Veränderung des Kurses indiziert, wenn auch wirklich ein neuer Kurs vorliegt. Anschließend werden die Zeitreihen, wie in Abschnitt 4.2 beschrieben, mittels ADF auf Stationarität geprüft und ggf. durch Differenzenbildung stationarisiert. Die Datenaufbereitung ist in der Funktion *preproc_dataset*[79] implementiert.

[77] Vgl. DevTeam (2019).

[78] siehe Anhang B.6.

[79] siehe Anhang B.11.

Modellanpassung & -validierung

Zunächst wird die optimale Lag-Ordnung mit dem in Abschnitt 4.3 vorgestellten Informationskriterium AIC ermittelt. Das Informationskriterium ist in der Funktion *get_lag_order* implementiert, wie sie im Anhang B.13 dargestellt ist. Mit der ermittelten Ordnung kann nun mithilfe des Pakets *statsmodels* ein $VAR(p)$-Modell aufgestellt und an die Daten angepasst werden. Um die Prognosegüte zu beurteilen und insbesondere um eine Verbesserung des Prognosemodells durch Hinzunahme der Sentiment-Daten identifizieren zu können, wird auch ein $AR(p)$-Modell, welches nur die Aktienkurse enthält, aufgestellt und trainiert.

Vorhersage

Mit beiden Modellen ($VAR(p)$ und $AR(p)$) wird nun eine Vorhersage getroffen. Das Aufstellen der Modelle und das Erzeugen einer Vorhersage wird in einer Funktion gemeinsam ausgeführt. Für das $VAR(p)$-Modell wird die Vorhersage durch die Funktion *var_prediction*[80] erzeugt. Das $AR(p)$-Modell wird durch *ar_prediction*[81] erzeugt.

Bewertung der Prognosegüte

Die Bewertung der Prognosegüte basiert auf einer Menge von Gütemaßen, die alle in der Funktion *forecast_accuary*[82] berechnet werden. Dazu werden die Zeitreihen mit den Prognosewerten des *AR*- und des *VAR*-Modells mit den echten Werten des Aktienkurses verglichen und die Abweichungen gemessen. Die Beurteilung der Prognosegüte und die Diskussion der Ergebnisse findet im folgenden Abschnitt statt.

[80] Siehe Anhang B.14.

[81] Siehe Anhang B.15.

[82] Siehe Anhang B.16.

4.6. Ergebnisse

Um die Prognosegüte zu beurteilen wird das Maß der **Trefferquote**[83] verwendet. Um die Trefferquote zu ermitteln wird ein binärer Klassifikator verwendet. Dafür wurde die Richtung der Bewegung des Aktienkurses im Vergleich zum Vortag ermittelt und mit $+1$ (Aktienkurs ist gestiegen), -1 (Aktienkurs ist gefallen) ausgedrückt. Der Fall, dass es keine Veränderung zum Vortag gab, kommt ausschließlich bei den Datensätzen vor, die auf keinen Handelstag gefallen sind. Die Daten mit Nullwerten (keine Veränderung) wurden daher aus der Bewertung entfernt und es liegt ein binärer Klassifikator vor.

In Abbildungen 13 und 14 sind die Trefferquoten und der Mean Absolute Percentage Error (MAPE)[84] der beiden Modelle im Vergleich dargestellt. Dabei wurden Teildatensätze mit verschiedenen Längen und Zeiträumen getestet.

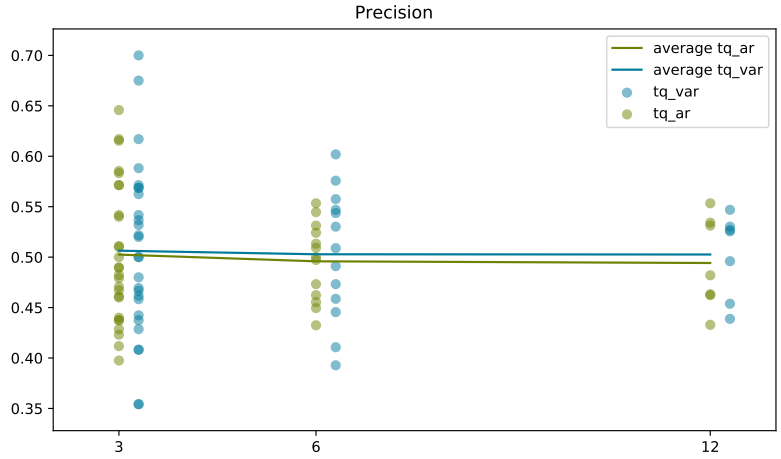

Abbildung 13.: Präzision der beiden Modelle im Vergleich[85].

[83] Vgl. Definition A.2 in Anhang A.1.

[84] Vgl. Definition A.4 in Anhang A.1.

[85] Eigene Darstellung.

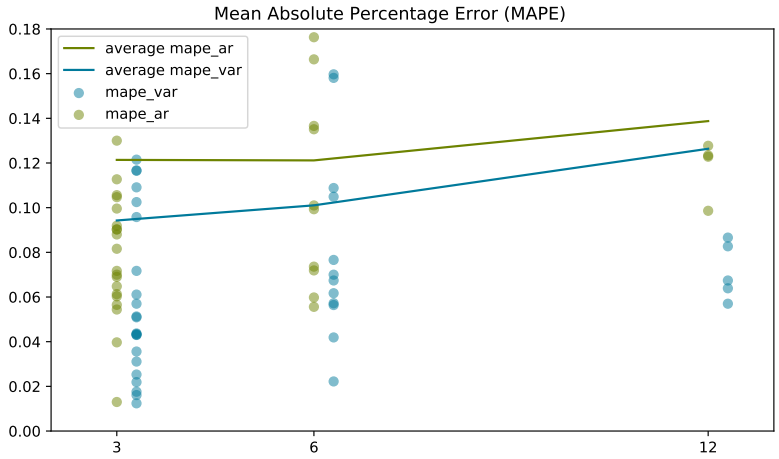

Abbildung 14.: MAPE der beiden Modelle im Vergleich[86].

Man erkennt anhand der Trefferquoten in Abbildung 13, dass durch Hinzunahme der Sentiment-Variablen eine Verbesserung der Prognose möglich ist. Die Verbesserung liegt allerdings nur im Bereich von wenigen Prozentpunkten. Der MAPE in Abbildung 14 zeigt ein ähnliches Bild. Im VAR-Modell unter Einbeziehung der Sentiment-Variablen konnte der MAPE reduziert werden. Die Verbesserung der Prognose ist dabei unabhängig von der Länge des Zeitraums.

[86]Eigene Darstellung.

5. Kritische Betrachtung und Ausblick

In diesem Kapitel wird eine Zusammenfassung der Ergebnisse gegeben sowie Stärken und Schwächen des Modells diskutiert. Außerdem wird ein Ausblick auf Anpassungen des Modells für eine Weiterführung gegeben.

Für die Erzeugung eines Textkorpus muss zunächst ein Unternehmen ausgewählt werden, dessen zugehörige Tweets analysiert werden sollen. In dieser Arbeit wurden ausschließlich „Global Player“ ausgewählt, also Unternehmen, die weltweit agieren und deren Handeln ausreichend in den sozialen Netzwerken diskutiert wird. Kleinere Unternehmen eignen sich aufgrund einer geringen Anzahl von Tweets nicht für die Analyse. Die Analyse beschränkte sich außerdem nur auf Tweets in englischer Sprache, da für verschieden Sprachen verschiedene Lexika für die Sentiment-Analyse benötigt werden. Der analysierbare Textkorpus könnte dahingehend erweitert werden, indem mehrere Sprachen zur Analyse hinzugezogen werden.

In der Datenerhebung wurden ca. 350.000 Tweets erfasst und analysiert, von denen jedoch nur ca. 61% (ca. 212.000 Tweets) erfolgreich analysiert und verwertet werden konnten. Durch den relativ großen Anteil nicht analysierbarer Tweets geht ein erheblicher Anteil an Informationen verloren. Durch eine Erweiterung der Methoden der Sentiment-Analyse, vorallem durch Verwendung umfangreicherer Lexika, könnte der Anteil der analysierbaren Tweets noch deutlich gesteigert werden. Auf maschinellem Lernen basierende Methoden der Sentiment-Analyse erzielen derzeit zwar noch schlechtere Ergebnisse bei der Genauigkeit der Analyse, erreichen jedoch meistens einen höheren Anteil analysierter Tweets[87]. Durch die Kombination der beiden Methoden und die daraus resultierenden Synergieeffekte könnten sich bessere Ergebnisse in der Analyse erzielen lassen.

Für die Entwicklung einer Sentiment-Variablen wurde außerdem ausschließlich die Polarität eines Textes verwendet. Menschliche Emotionen sind jedoch wesentlich komplexer und lassen sich nicht auf einer eindimensionalen Skala abbilden. Mit dieser Reduktion ist daher auch ein Informationsverlust verbunden. Durch komplexere Sentiment-Analyse-Modelle könnten auch diese in den Texten enthaltenen Informationen analysiert und verwendet werden. Es gibt bereits Ansätze, die neben der Polarität auch die Subjektivität eines Textes ermitteln können, sowie

[87] Siehe Abschnitt 3.2.

komplexere Ansätze wie z. B. das „Profile of Mood States“, mit dem Emotionen in sechs verschiedenen Dimensionen ausgedrückt werden. Eine Verwendung komplexere Sentiment-Analyse-Modelle könnte also ein genaueres Stimmungsbild liefern und eventuell bessere Prognosen liefern.

Die Ergebnisse des vorigen Kapitels zeigen, dass sich mithilfe der Daten aus sozialen Netzwerken ein Prognosemodell für Aktienkurse verbessern lässt. Dabei werden jedoch schnell die Grenzen des Modells sichtbar. Mit nur wenigen Prozentpunkten fällt die Verbesserung des Modells durch Hinzunahme der Sentiment-Variablen allgemein sehr schwach aus. Dennoch konnte das Modell die erwartete Genauigkeit eines „einfachen Ratens“ von 50% übertreffen.

Weitere Verbesserungsmöglichkeiten gibt es bei der Modellbildung. Durch die zahlreichen Methoden zur Fehleranalyse, die das VAR-Modell ermöglicht, könnte ein besser angepasstes Modell entwickelt und damit eine präzisere Vorhersage erzeugt werden. Dabei könnten speziell für Finanzzeitreihen entwickelte Modelle wie z. B. ARCH-Zeitreihen ebenfalls verwendet werden, um die Prognose zu verbessern.

6. Fazit

Diese Arbeit wurde durch die Fragestellung motiviert, ob sich die Stimmung in sozialen Netzwerken auf den Aktienkurs eines Unternehmens auswirkt und ob sich mithilfe dieser Informationen eine Prognose für einen Aktienkurs ermitteln lässt. Als Einführung wurde dafür zunächst ein Überblick über Forschungsarbeiten mit ähnlichen Fragestellungen gegeben und deren Methoden verglichen. Da für eine eigene Vorhersage zunächst eine Variable benötigt wird, die die Stimmung in sozialen Netzwerken beschreibt, wurden zunächst die Grundlagen der Textverarbeitung in der Computerlinguistik dargestellt und darauf aufbauend ein Modell zur Textverarbeitung entwickelt. Anschließend wurden zu verschiedenen Unternehmen Tweets abgerufen und zu einem Textkorpus zusammengefasst. Zur Analyse der Stimmung eines Textkorpus ist eine Sentiment-Analyse notwendig. Die verschiedenen Modelle zur Ermittlung des Sentiments wurden dazu zunächst vorgestellt und schließlich die Textkorpora analysiert, woraus eine Sentiment-Variable abgeleitet werden konnte.

Um die Frage zu beantworten, ob ein Prognosemodell durch Hinzunahme einer Sentiment-Variablen verbessert werden kann, muss zunächst ein Referenzmodell aufgestellt werden. Hierzu wurde das Modell der Vektorautoregression vorgestellt, die Schätzung der Parameter und der Ordnung erörtert und aufgezeigt, wie sich anhand des Modells Prognosen erzeugen lassen. Mit dem Referenzmodell wurde zunächst eine Vorhersage des Aktienkurses ohne Einbeziehung der Sentiment-Variablen getroffen. Anschließend wurde dem Prognosemodell die Sentiment-Variable sowie die Anzahl täglicher Tweets hinzugefügt und die daraus resultierenden Prognosen wurden mit dem Referenzmodell verglichen. Beim Vergleich der Trefferquoten der beiden Modelle konnte gezeigt werden, dass sich durch Sentiment-Variablen eine Prognose von Aktienkursen verbessern lässt.

In einer kritischen Betrachtung wurden abschließend einige Schwächen aufgezeigt und es konnten Impulse für eine Fortführung der Arbeit gegeben werden.

A. Mathematischer Anhang

A.1. Definitionen

Definition A.1: Präzision

Die **Präzision** P eines binären Klassifikators ist der Anteil der korrekt als positiv klassifizierten Ergebnisse an der Gesamtheit der als positiv klassifizierten Ergebnisse:

$$P(\text{tatsächlich positiv} \mid \text{positive Klassifikation}) = \frac{r_p}{r_p + f_p}, \tag{A.1}$$

wobei f_p der Anteil der falschen positiven Klassifizierten und r_p der Anteil der richtigen positiven Klassifikationen ist.

Auch: *Genauigkeit, positiver Vorhersagewert, precision*

Definition A.2: Trefferquote

Die **Trefferquote** R eines binären Klassifikators ist der Anteil der korrekt als positiv klassifizierten Ergebnisse an der Gesamtheit der tatsächlich positiven Ergebnisse:

$$P(\text{positive Klassifikation} \mid \text{tatsächlich positiv}) = \frac{r_p}{r_p + f_n}, \tag{A.2}$$

wobei f_n der Anteil der falschen negativ Klassifizierten und r_p der Anteil der richtigen positiven Klassifikationen ist.

Auch: *Sensitivität, Empfindlichkeit, recall, hit rate*

Definition A.3: F-Maß

Das F_α**-Maß** ist ein kombiniertes Maß für die Beurteilung der Güte eines binären Klassifikators. Es ist definiert durch

$$F_\alpha = (1+\alpha^2) \cdot \frac{P \cdot R}{\alpha^2 \cdot P + R} \tag{A.3}$$

mit der Präzision P und der Trefferquote R.

Das F_1**-Maß** (oft auch nur F-Maß genannt) ergibt sich als Spezialfall bei einer Gleichgewichtung von Präzision und Trefferquote:

$$F_1 = (1+\alpha^2) \cdot \frac{P \cdot R}{\alpha^2 \cdot P + R} = 2 \cdot \frac{P \cdot R}{P + R} \tag{A.4}$$

Definition A.4: Mean absolute percentage error

Der **mittlere absolute prozentuale Fehler** (mean absolute percentage error, MAPE) ist ein Maß für die Prognosegüte:

$$MAPE = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{a_t - f_t}{a_t} \right| , \tag{A.5}$$

wobei a_t der tatsächliche Wert und f_t der Vorhersagewert ist.

A.2. POS Tagsets

The Penn Treebank POS tagset[88]			
CC	Coordinating conjunction	TO	*to*
CD	Cardinal number	UH	Interjection
DT	Determiner	VB	Verb, base form
EX	Existential *there*	VDB	Verb, past tense
FW	Foreign Word	VBG	Verb, gerund/present participle
IN	Prepositionsubordinating conjunction	VBN	Verb, past participle
JJ	Adjective	VBP	Verb, non-3rd ps. sing. present
JJR	Adjective, comparative	VBZ	Verb, 3rd ps. sing. present
JJS	Adjective, superlative	WDT	*wh*-determiner
LS	List item marker	WP	*wh*-pronoun
MD	Modal	WP$	Possessive *wh*-pronoun
NN	Noun, singular or mass	WRB	*wh*-adverb
NNS	Noun, plural	#	Pound sign
NNP	Proper noun, singular	$	Dollar sign
NNPS	Proper noun, plural	.	Sentence-final punctuation
PDT	Predeterminer	,	Comma
POS	Possessice ending	:	Colon, semi-colon
PRP	Personal pronoun	(	Left bracket character
PP$	Possessive pronoun	)	Right bracket character
RB	Adverb	"	Straight double quote
RBR	Adverb, comparative	‘	Left open single quote
RBS	Adverb, superlative	“	Left open double quote
RP	Particle	’	Right close single quote
SYM	Symbol (mathematical or scientific)	”	Right close double quote

Tabelle A.1.: Abkürzungen des Penn-Treebank-Tagsets. Das Tagset beinhaltet alle Abkürzungen für die verschiedenen parts-of-speech, die im Zuge der Tokenisierung einem Satzteil zugewiesen werden können.

A.3. Vektorisierung von Matrizen

Einige Sätze und Beweise verwenden die Vektorisierung von Matrizen. Dabei handelt es sich um eine lineare Abbildung, die Matrizen in einen Vektor umwandelt. Sie wird meist durch $vec(A)$ notiert. Die Abbildung wandelt eine Matrix $A \in \mathbb{R}^{m \times n}$ in einen Vektor $vec(A) \in \mathbb{R}^{mn \times 1}$ um, indem die Spalten der Matrix „aufeinander gestapelt“ werden, also

$$vec(A) = vec\left(\begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix}\right) = [a_{1,1}, \ldots, a_{m,1}, a_{1,2}, \ldots, a_{m,2}, \ldots, a_{m,n}]^T$$

Oft wird diese Schreibweise in Zusammenhang mit dem Kronecker-Produkt verwendet, da man so Matrixmultiplikationen als Lineare Abbildung von Matrizen schreiben kann. Dabei gilt für vektorisierte Matrizen (passender Dimension):

$$vec(AB) = (I \otimes A)vec(B) = (B^T \otimes I)vec(A)$$

$$vec(ABC) = (I \otimes AB)vec(C) = (C^T B^T \otimes I)vec(A)$$

mit der Einheitsmatrix I[89].

[88] Vgl. Marcus, Santorini und Marcinkiewicz (1993), S. 317.

[89] Vgl. Lütkepohl, 2005, S. 661 ff.

A.4. Notationen des VAR(p)-Modells

Matrixschreibweise

Ein VAR(p)-Modell mit k Variablen für $T+1$ Beobachtungen $(y_p, ..., y_T)$ lässt sich in unterschiedlichen Notationen darstellen. In Abschnitt 4.4 wird diese kurze Matrixschreibweise verwendet[90]:

Korollar A.1: Matrixschreibweise des VAR(p)-Modells

$$Y = BZ + U$$

mit

$$Y = [y_p \; y_{p+1} \; \cdots \; y_T] = \begin{bmatrix} y_{1,p} & y_{1,p+1} & \cdots & y_{1,T} \\ y_{2,p} & y_{2,p+1} & \cdots & y_{2,T} \\ \vdots & \vdots & \vdots & \vdots \\ y_{k,p} & y_{k,p+1} & \cdots & y_{k,T} \end{bmatrix}$$

$$B = [c \; A_1 \; A_2 \; \cdots \; A_p] = \begin{bmatrix} c_1 & a^1_{1,1} & a^1_{1,2} & \cdots & a^1_{1,k} & \cdots & a^p_{1,1} & a^p_{1,2} & \cdots & a^p_{1,k} \\ c_2 & a^1_{2,1} & a^1_{2,2} & \cdots & a^1_{2,k} & \cdots & a^p_{2,1} & a^p_{2,2} & \cdots & a^p_{2,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \vdots & \ddots & \vdots \\ c_k & a^1_{k,1} & a^1_{k,2} & \cdots & a^1_{k,k} & \cdots & a^p_{k,1} & a^p_{k,2} & \cdots & a^p_{k,k} \end{bmatrix}$$

[90] Vgl. Lütkepohl (2005), S. 70.

$$Z = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ y_{p-1} & y_p & \cdots & y_{T-1} \\ y_{p-2} & y_{p-1} & \cdots & y_{T-2} \\ \vdots & \vdots & \ddots & \vdots \\ y_0 & y_1 & \cdots & y_{T-p} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ y_{1,p-1} & y_{1,p} & \cdots & y_{1,T-1} \\ y_{2,p-1} & y_{2,p} & \cdots & y_{2,T-1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{k,p-1} & y_{k,p} & \cdots & y_{k,T-1} \\ y_{1,p-2} & y_{1,p-1} & \cdots & y_{1,T-2} \\ y_{2,p-2} & y_{2,p-1} & \cdots & y_{2,T-2} \\ \vdots & \vdots & \ddots & \vdots \\ y_{k,p-2} & y_{k,p-1} & \cdots & y_{k,T-2} \\ \vdots & \vdots & \ddots & \vdots \\ y_{1,0} & y_{1,1} & \cdots & y_{1,T-p} \\ y_{2,0} & y_{2,1} & \cdots & y_{2,T-p} \\ \vdots & \vdots & \ddots & \vdots \\ y_{k,0} & y_{k,1} & \cdots & y_{k,T-p} \end{bmatrix}$$

$$U = [e_p \; e_{p+1} \; \cdots \; e_T] = \begin{bmatrix} e_{1,p} & e_{1,p+1} & \cdots & e_{1,T} \\ e_{2,p} & e_{2,p+1} & \cdots & e_{2,T} \\ \vdots & \vdots & \ddots & \vdots \\ e_{k,p} & e_{k,p+1} & \cdots & e_{k,T} \end{bmatrix}$$

Rekursive Darstellung

Ein $VAR(1)$-Prozess kann durch rekursives Einsetzen dargestellt werden als:

$$
\begin{aligned}
y_1 &= \nu + A_1 y_0 + u_1 \\
y_2 &= \nu + A_1 y_1 + u_2 = \nu + A_1(\nu + A_1 y_0 + u_1) + u_2 \\
&= (I_K + A_1)\nu + A_1^2 y_0 + A_1 u_1 + u_2 \\
&\vdots \\
y_t &= (I_K + A_1 + \ldots + A_1^{t-1})\nu + A_1^t y_0 + \sum_{i=0}^{t-1} A_1^i u_{t-i}
\end{aligned}
$$

A.5. Rückführung von VAR(p) auf VAR(1)

Korollar A.2: Rückführung von VAR(p) auf VAR(1)[91]

Für höhere Ordnungen p kann der Prozess $VAR(p)$ auf einen $VAR(1)$-Prozess zurückgeführt werden. Dazu sei $y_t = \nu + A_1 y_{t-1} + \ldots + A_p y_{t-p} + u_t$, $t = 0, \pm 1, \pm 2, \ldots$ ein $VAR(p)$-Prozess.

Zu diesem $VAR(p)$-Prozess kann nun ein entsprechender Kp-dimensionaler $VAR(1)$-Prozess

$$Y_t = \nu + AY_{t-1} + U_t$$

definiert werden, indem

$$Y_t := \begin{bmatrix} y_t \\ y_{t-1} \\ \vdots \\ y_{t-p+1} \end{bmatrix}, \ \nu := \begin{bmatrix} \nu \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

$$A := \begin{bmatrix} A_1 & A_2 & \cdots & A_{p-1} & A_p \\ I_K & 0 & \cdots & 0 & 0 \\ 0 & I_K & & 0 & 0 \\ \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & I_K & 0 \end{bmatrix}, \ \begin{bmatrix} u_t \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

gewählt werden.

[91] Vgl. Lütkepohl, 2005, S. 15 f.

B. Quellcode-Anhang

B.1. SQL

Die in dieser Arbeit verwendete Datenbank arbeitet mit dem gemeinfreien, relationalen Datenbanksystem SQLite. SQLite unterstützt den SQL-92-Standard. Alle Abfragen auf der SQLite-Datenbank werden demnach mit der Structured Query Language **SQL** durchgeführt.

Quellc Nr.	Funktion	Beschreibung
B.1	CREATE_TABLE_TWEETS.sql	Erstellen der Datenbank-Tabelle 'tweets'
B.2	CREATE_TABLE_SEARCHES.sql	Erstellen der Datenbank-Tabelle 'searches'
B.3	CREATE_TABLE_STOCKS.sql	Erstellen der Datenbank-Tabelle 'stocks'
B.4	CREATE_TABLE_TWEETS.sql	Prozedur zum Entfernen von Duplikaten in der Datenbank.
B.6	GET_FULL_TIME_SERIES.sql	Prozedur zum Aggregieren der Daten und zum Erzeugen der Zeitreihe.

Tabelle B.1.: Übersicht der SQL-Queries

Erstellen der Datenbank-Tabellen

Die folgenden Funktionen werden verwendet, um die Datenbank-Tabellen zu erstellen.

Funktion	CREATE_TABLE_TWEETS.sql
Beschreibung	Erzeugt die Tabelle 'tweets' zum Speichern der abgerufenen Tweets.
Aufruf	Der Aufruf kann mittels Python (package *sqlite3*) oder mit einem Datenbankmanagementsystem (z. B. SQLiteStudio) erfolgen.

```
CREATE TABLE tweets (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    screen_name TEXT,
    user_name TEXT,
    date DATE,
    text TEXT,
    text_len INT,
    flag_faulty BOOLEAN,
    flag_sentiment BOOLEAN,
    flag_answer BOOLEAN,
    search_id INTEGER,
    tb_polarity REAL,
    tb_subjectivity REAL,
    tb_sentiment REAL,
    vader_neg REAL,
    vader_neu REAL,
    vader_pos REAL,
    vader_compound REAL,
    vader_sentiment REAL,
    total_sentiment REAL
);
```

Quellcode B.1: SQL: Erstellen der Tabelle 'tweets'

Funktion	CREATE_TABLE_SEARCHES.sql
Beschreibung	Erzeugt die Tabelle 'searches', in der Informationen zu einer Suchabfrage gespeichert werden. Die Tabelle dient der Verknüpfung der Tweets aus der Tabelle 'tweets' mit den Aktienkursen aus der Tabelle 'stocks'.
Aufruf	Der Aufruf kann mittels Python (package *sqlite3*) oder mit einem Datenbankmanagementsystem (z. B. SQLiteStudio) erfolgen.

```
CREATE table searches (
    search_id INTEGER,
```

```
    type TEXT,
    query TEXT,
    symbol TEXT
);
```

Quellcode B.2: SQL: Erstellen der Tabelle 'searches'

Funktion	CREATE_TABLE_STOCKS.sql
Beschreibung	Erezugt die Tabelle 'stocks' zum Speichern der Aktienkurse, die für die Prognose benötigt werden. Gespeichert werden alle Preisinformationen: high (höchster Kurs des Tages), low (niedrigster Kurs des Tages), open (Eröffnungskurs), close (Schlusskurs), adj_close (angepasster Schlusskurs, der Kapitalmaßnahmen und Ausschüttungen berücksichtigt), volume (Handelsvolumen).
Aufruf	Der Aufruf kann mittels Python (package *sqlite3*) oder mit einem Datenbankmanagementsystem (z. B. SQLiteStudio) erfolgen.

```
CREATE table stocks (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    symbol TEXT,
    date DATE,
    source TEXT,
    high REAL,
    low REAL,
    open REAL,
    close REAL,
    adj_close REAL,
    volume INTEGER,
    UNIQUE(symbol, date) ON CONFLICT ROLLBACK
);
```

Quellcode B.3: SQL: Erstellen der Tabelle 'stocks'

Stored Procedures

Funktion	SP_REMOVE_DUPLICATES.sql
Beschreibung	Entfernt doppelt vorhandene Datensätze (Duplikate) aus der Datenbank.
Aufruf	Der Aufruf kann mittels Python (package *sqlite3*) oder mit einem Datenbankmanagementsystem (z. B. SQLiteStudio) erfolgen.

```sql
DELETE FROM tweets
    WHERE rowid NOT IN (
        SELECT MIN(rowid)
        FROM tweets
        GROUP BY text
        );
```

Quellcode B.4: SQL: Entfernen von Duplikaten

Funktion	SP_FLAG_FAULTY.sql
Beschreibung	Entfernt fehlerhafte Datensätze aus der Datenbank. Hier wurde lediglich die Länge des Tweets als Indikator für einen Fehler verwendet. Die Kriterien zur Identifikation können aber beliebig erweitert werden.
Aufruf	Der Aufruf kann mittels Python (package *sqlite3*) oder mit einem Datenbankmanagementsystem (z. B. SQLiteStudio) erfolgen.

```sql
UPDATE tweets
    SET flag_faulty = 1
    WHERE text_len > 280
```

Quellcode B.5: SQL: Identifizieren fehlerhafter Datensätze

Erzeugen der Zeitreihen

Funktion	SP_GET_FULL_TIMESERIES.sql
Beschreibung	Aggregiert die Daten (Tweets und Kursdaten) zu einem Datensatz für die Regressionsanalyse.
Aufruf	Der Aufruf kann mittels Python (package *sqlite3*) oder mit einem Datenbankmanagementsystem (z. B. SQLiteStudio) erfolgen.

```sql
SELECT T.date,
       AVG(T.vader_sentiment) AS AVG,
       COUNT(T.vader_sentiment) AS COUNT,
       S.adj_close AS STOCK
  FROM tweets AS T
       LEFT JOIN
       searches AS SE ON SE.search_id = T.search_id
       LEFT JOIN
       stocks AS S ON SE.symbol = S.symbol AND
                      S.date = T.date
```

```
11 WHERE (T.flag_faulty IS NULL OR
12         T.flag_faulty = 0) AND
13       T.date >= ? AND
14       T.date <= ? AND
15       SE.symbol = ?
16 GROUP BY T.date,
17          SE.symbol
```

Quellcode B.6: SQL: Erzeugen der Zeitreihen für die Regressionsanalyse

B.2. Python

Die Implementierung erfolgte in Python 3.6.4. Die meisten Funktionen sind mit Python 3.7 kompatibel, ein ausführlicher Test aller Funktionen wurde für andere Versionen jedoch nicht durchgeführt. Mit Python 2 und Python 3.8 kommt es zu Kompatibilitätsproblemen.

Modul	Quellc Nr.	Funktion	Beschreibung
database	B.7	add_sentiments	Steuerung der Sentiment-Analyse
sentiment	B.8	get_sentiment	Aufruf der verschiedenen Sentiment-Analyse-Modelle
	B.9	vader_sentiment	Für das VADER-Paket angepasster Funktionsaufruf zur Sentiment-Analyse
figures	B.10	get_dataset_statistics	Visualisierung und Datensatz-Statistiken
regression	B.11	preproc_dataset	Datenaufbereitung
	B.12	adfuller_test	Augmented Dickey-Fuller Test
	B.13	get_lag_order	Anwendung der Informationskriterien zum Ermitteln der Lag-Ordnung
	B.14	var_prediction	Vorhersage mit einem VAR-Modell
	B.15	ar_prediction	Vorhersage mit einem AR-Modell
	B.16	forecast_accuracy	Ermitteln der Prognosegüte

Tabelle B.2.: Übersicht der Python-Funktionen

Sentiment-Analyse

Funktion	add_sentiments
Beschreibung	Die Funktion fragt Tweets aus der Datenbank ab, ermittelt ein Sentiment für den Text des Tweets und gibt den analysierten Tweet inklusive seiner Sentiment-Werte an die Datenbank zurück. Die Funktion dient als Wrapper für die Funktion *get_sentiment*, mit der die Sentiment-Analyse durchgeführt wird.
Aufruf	Beim Aufruf können einige optionale Einstellungen vorgenommen werden, z. B. kann eine maximale Anzahl zu analysierender Tweets festgelegt werden.

```
1  def add_sentiments(**kwargs):
2      """
3          fetches datasets without valid sentiment and runs sentiment analyzer on them
4
5          parameters:
6
7          kwargs:
8              max                     max amount of datasets picked from database
9                                           DEFAULT: no max, picks all
10             silent                  whether or not to display status of processing
11                                          DEFAULT: false, displaying progress
12     """
13
14 #    kwargs
15     max = None if not 'max' in kwargs else kwargs['max']
16     silent = False if not 'silent' in kwargs else kwargs['silent']
17
18 #    imports
19     from sentiments import get_sentiment                # for getting sentiment values
20     from progress.bar import ChargingBar as Bar         # progress bar
21
22 #    get unrated tweets
23     sql = """
24         SELECT id,
25                text,
26                date,
27                flag_sentiment
28           FROM tweets
29          WHERE flag_sentiment IS NULL OR
30                flag_sentiment = 0
31     """
32     if not max is None:
33         sql += ('LIMIT ' + str(max))
34     sql += ';'
```

```
35
36  #   connect db
37      con = create_connection(DB_FILE)
38      c = create_cursor(con)
39
40  #   get data
41      c.execute(sql)
42      data = c.fetchall()
43      if not silent:
44          print('Fetched ' + str(len(data)) + ' tweets from ' + DB_FILE + '.')
45
46  #   analyze tweets
47      bar = Bar('Running sentiment analyzer', max = len(data))
48      sent_values = {
49          'tb_polarity': '',
50          'tb_subjectivity': '',
51          'vader_neg': '',
52          'vader_neu': '',
53          'vader_pos': '',
54          'vader_compound': '',
55      }
56      for tweet in data:
57  #       TextBlob
58          sentiment = get_sentiment(tweet[1], engine = 'textblob')
59          sent_values['tb_polarity'] = sentiment[0]
60          sent_values['tb_subjectivity'] = sentiment[1]
61  #       VADER
62          sentiment = get_sentiment(tweet[1], engine = "vader")
63          sent_values['vader_neg'] = sentiment['neg']
64          sent_values['vader_neu'] = sentiment['neu']
65          sent_values['vader_pos'] = sentiment['pos']
66          sent_values['vader_compound'] = sentiment['compound']
67  #       update db
68          sql = '''
69              UPDATE tweets
70              SET tb_polarity = ?,
71              tb_subjectivity = ?,
72              vader_neg = ?,
73              vader_neu = ?,
74              vader_pos = ?,
75              vader_compound = ?,
76              flag_sentiment = 1
77              WHERE id IS ?
78          '''
79          c.execute(sql, (sent_values['tb_polarity'],
80                          sent_values['tb_subjectivity'],
81                          sent_values['vader_neg'],
82                          sent_values['vader_neu'],
83                          sent_values['vader_pos'],
84                          sent_values['vader_compound'],
```

```
85                          tweet[0]))
86 #        progress
87          bar.next()
88
89 #    commit changes
90      con.commit()
91      con.close()
92
93 #    end
94      bar.finish()
```

Quellcode B.7: Python: Abruf der Tweets aus der Datenbank und Aufruf der Sentiment-Analyse

Funktion	get_sentiments
Beschreibung	Die Funktion steuert den Aufruf der einzelnen Sentiment-Analyse-Funktionen.
Aufruf	Beim Aufruf kann das zu verwendende Sentiment-Analyse-Modell optional gewählt werden. Standarmäßig wird das VADER-Paket verwendet.

```
1  def get_sentiment(text, **kwargs):
2      """
3          function to get a sentiment. returns only polarity values mapped to [-1, 1]
4
5          parameter:
6              text:               Text or tweet object to be analyzed
7
8          kwargs:
9              engine:             Analysis engine to be used
10                                     'vader' | 'textblob'
11                                     DEFAULT: 'vader'
12     """
13
14 #    kwargs
15     engine = 'vader' if 'engine' not in kwargs else kwargs['engine']
16
17 #    calls
18     if engine == 'vader':
19         return(vader_sentiment(text, ret = 'dict'))
20     if engine == 'textblob':
21         return(textblob_sentiment(text, ret = 'tuple'))
```

Quellcode B.8: Python: Wrapper für die Funktionen zur Sentiment-Analyse

Funktion	vader_sentiment
Beschreibung	Mit der Funktion wird ein Sentiment unter Verwendung des VADER-Paketes erzeugt und zurückgegeben.
Aufruf	Der Aufruf kann mit optionalen Argumenten erfolgen, wenn andere Rückgabe-Formate gewünscht sind.

```
1  def vader_sentiment(text, silent = True, **kwargs):
2      """
3          VADER Sentiment Analyzer
4
5          kwargs:
6              ret:                 defines the return value
7                                       'dict' returns an array [neg, neu, pos, compound]
8                                       DEFAULT returns only the compound value
9
10     """
11
12 #    kwargs
13     ret = 'dict' if 'ret' not in kwargs else kwargs['ret']
14     silent = True if 'silent' not in kwargs else kwargs['silent']
15
16 #    packages
17     from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
18
19 #    setup
20     analyzer = SentimentIntensityAnalyzer()
21
22 #    return
23     if ret == 'dict':
24         return(analyzer.polarity_scores(str(text)))
25     else:
26         return(analyzer.polarity_scores(str(text))['compound'])
```

Quellcode B.9: Python: Ermittlung eines Sentiments mit dem VADER-Paket.

Visualisierung und Datensatz-Statistiken

Funktion	get_dataset_statistics
Beschreibung	Funktion zum Erzeugen einer Visualisierung eines Sentiment-Datensatzes bestehend aus durchschnittlichem Sentiment, Tweet-Anzahl und Aktienkurs.
Aufruf	Beim Aufruf können die Parameter des Datensatzes (Beginn, Ende, Name des Datensatzes) und das Ausgabeformat (statistische Werte, Graph) gewählt werden.

```
1  def get_dataset_statistics(name, start, end, **kwargs):
2      """
3          function to visualize dataset and calculate some statistics
4
5          parameters:
6              name                        dataset name
7              start                       start date as string (YYYY-MM-DD)
8              end                         end date as string (YYYY-MM-DD)
9
10
11         kwargs:
12             statistics                  calculate and print out statistics
13                                             'full' | 'None'
14                                             DEFAULT: 'None'
15
16             plot                        display a plot of the time series
17                                             'full' | 'None'
18                                             DEFAULT: 'None'
19
20             fill                        fill NA-values in dataframe, else dropping rows
       with NA
21                                             True | False
22                                             DEFAULT: True
23
24     """
25
26 #   handle kwargs
27     statistics = 'None' if 'statistics' not in kwargs else kwargs['statistics']
28     plot = 'None' if 'plot' not in kwargs else kwargs['plot']
29     fill = True if 'fill' not in kwargs else kwargs['fill']
30
31 #   packages
32     from database import get_full_timeseries
33     import matplotlib.pyplot as plt
34
35 #   load data
36     df = get_full_timeseries(name, start = start, end = end)
```

```
37      if fill:
38          df['STOCK'].fillna(method = 'bfill', inplace = True)
39      else:
40          df.dropna(axis = 0, inplace = True)
41
42  #   empty message
43      if statistics == 'None' and plot == 'None':
44          print('Neither the plot nor the statistics option has been chosen. No output was
        produced.')
45          return()
46
47  #   plot
48      if plot == 'full':
49          fig, axs = plt.subplots(3, sharex = True, sharey = False)
50          axs[0].plot('date', 'AVG', data = df, color = '#00709c', label = 'sentiment')
51          axs[0].xaxis.set_major_locator(plt.MaxNLocator(6))
52          axs[0].legend()
53
54          axs[1].plot('date', 'COUNT', data = df, color = '#00709c', label = 'tweet count')
55          axs[1].xaxis.set_major_locator(plt.MaxNLocator(6))
56          axs[1].legend()
57
58          axs[2].plot('date', 'STOCK', data = df, color = '#00709c', label = 'stock')
59          axs[2].xaxis.set_major_locator(plt.MaxNLocator(6))
60          axs[2].legend()
61          plt.show()
62
63  #   do the stats
64      if statistics == 'full':
65          from statistics import mean, stdev, median
66
67          print('Dataset: {}'.format(name))
68          print('  {} rows'.format(str(len(df))))
69          print('  {} - {}\n'.format(start, end))
70          print('  value AVG:')
71          print('    mean     : {}'.format(str(mean(df['AVG']))))
72          print('    std      : {}'.format(str(stdev(df['AVG']))))
73          print('    median   : {}\n'.format(str(median(df['AVG']))))
74
75          print('  value COUNT:')
76          print('    sum      : {}     (total number of
        tweets)'.format(str(sum(df['COUNT']))))
77          print('    mean     : {}'.format(str(mean(df['COUNT']))))
78          print('    std      : {}'.format(str(stdev(df['COUNT']))))
79          print('    median   : {}\n'.format(str(median(df['COUNT']))))
```

Quellcode B.10: Python: Visualisierung und Berechnung statistische Werte eines Datensatzes

Datenaufbereitung

Funktion	preproc_dataset
Beschreibung	Die Funktion beinhaltet alle Funktionen zur Datenaufbereitung vor der Regressionanalyse. Beim Aufruf können einige Optionen zur Datenaufbereitung angepasst werden.
Aufruf	Beim Aufruf können einige Optionen zur Datenaufbereitung angepasst werden. Die möglichen Parameter sind im Quellcode erklärt.

```
def preproc_dataset(df, **kwargs):
    """
        function to pre-process the given dataset

        returns:
            list containing the preprocessed dataframe and a dictionary
            containing the original dataframe and information on which order
            processing had to be performed for stationarity


        parameters:
            df                          data frame to be pre-processed

        kwargs:
            silent                  whether or not to display status of processing
                                        True | False
                                        DEFAULT: True

            fill                    fill missing values in data (NA) with backfill
                                        True | False
                                        DEFAULT: True

            add_cols                list containing names for columns to add
                                        [...] | None
                                        DEFAULT: ['PREDICT']

            stat_test               test time series for stationarity
                                        'ADF' | None
                                        DEFAULT: 'ADF'

    """

#   packages
    import pandas as pd
    import numpy as np

#   handle kwargs
    silent = True if 'silent' not in kwargs else kwargs['silent']
    fill = True if 'fill' not in kwargs else kwargs['fill']
```

```
40      add_cols = ['PREDICT'] if 'add_cols' not in kwargs else kwargs['add_cols']
41      stat_test = 'ADF' if 'stat_test' not in kwargs else kwargs['stat_test']
42
43  #   pre-process
44      df.index = pd.DatetimeIndex(df.date).to_period("D")
45      if fill == True:
46          if not silent: print('Using ffill to close {} gaps in
    data.'.format(str(sum(df['STOCK'].isna()))))
47          df['STOCK'].fillna(method = 'ffill', inplace = True)
48          if sum(df['STOCK'].isna()) > 0:
49              df['STOCK'].fillna(method = 'bfill', inplace = True)            # NA in 1st
50
51
52  #   col for prediction results
53      if not add_cols == [] and not add_cols is None:
54          for col in add_cols:
55              df.insert(loc = 1,
56                        column = 'PREDICT',
57                        value = np.nan)
58
59  #   test stationarity and differentiate when needed
60      if not stat_test is None:
61          adf_output = 'long' if silent is False else None
62          df_orig = df.copy()
63          diffs = {'df_orig': df_orig, 'AVG': 0, 'COUNT': 0, 'STOCK': 0}
64          for col in ['AVG', 'COUNT', 'STOCK']:
65              while not adfuller_test(df[col], signif = 0.05, name = col, print_output =
    adf_output):
66                  diffs[col] += 1
67                  if not silent: print('{} is not stationary and needs to be
    differenced.'.format(str(col)))
68                  df[col] = df[col].diff()
69                  df[col].dropna(inplace = True)
70          if not silent:
71              print('DataFrame was stationarized using differencing:')
72              print('   AVG was differenced {} times'.format(str(diffs['AVG'])))
73              print('   COUNT was differenced {} times'.format(str(diffs['COUNT'])))
74              print('   STOCK was differenced {} times'.format(str(diffs['STOCK'])))
75      else:
76          diffs = None
77
78
79  #   remove NaN's created through differencing
80      df.fillna(0, inplace = True)
81
82  #   return
83      return([df, diffs])
```

Quellcode B.11: Python: Aufbereitung eines Datensatzes für die Analyse

Augmented Dickey-Fuller Test

Funktion	adfuller_test
Beschreibung	Funktion zum Ausführen des Augmented Dickey-Fuller Tests für Stationarität. Die Funktion verwendet das Paket *statsmodels* und die darin implementierte Teststatistik. In der Regressionsanalyse wird die Funktion verwendet, um die Zeitreihen auf Stationarität zu prüfen.
Aufruf	Die Funktion benötigt eine Zeitreihe in Form eines pandas DataFrame und gibt einen Wahrheitswert zurück, ob die Nullhypothese verworfen werden kann. Als optionale Parameter können z. B. Signifikanzniveau (*signif*) und eine Ausgabe der Testergebnisse (*print_output = None | 'short' | 'long'*) übergeben werden.

```
def adfuller_test(series, signif = 0.05, name = '', print_output = None):
    """
        Function to perform ADF Test for Stationarity of given time series and print a report
        returns True if stationary, False if not

        parameter:
            series                          series to perform ADF test on
            signif                          significance level
            name
            output                          DEFAULT = 'None'
                                                None: no print output, only return value True/False
                                                'short': a short version of the output (not yet implemented)
                                                'long': the long version of the report of the ADF test

        returns
    """

    from statsmodels.tsa.stattools import adfuller

    r = adfuller(series, autolag = 'AIC')
    output = {'test_statistic':round(r[0], 4), 'pvalue':round(r[1], 4), 'n_lags':round(r[2], 4), 'n_obs':r[3]}
    p_value = output['pvalue']


#   print report
    if print_output == 'long':
        print(f'    Augmented Dickey-Fuller Test on "{name}"', "\n   ", '-'*47)
        print(f' Null Hypothesis: Data has unit root. Non-Stationary.')
```

```
29          print(f' Significance Level    = {signif}')
30          print(f' Test Statistic        = {output["test_statistic"]}')
31          print(f' No. Lags Chosen       = {output["n_lags"]}')
32
33          for key,val in r[4].items():
34              print(f' Critical value {adjust(key)} = {round(val, 3)}')
35
36          if p_value <= signif:
37              print(f" => P-Value = {p_value}. Rejecting Null Hypothesis.")
38              print(f" => Series is Stationary.")
39          else:
40              print(f" => P-Value = {p_value}. Weak evidence to reject the Null
    Hypothesis.")
41              print(f" => Series is Non-Stationary.")
42          print('\n')
43
44      return(True if p_value <= signif else False)
```

Quellcode B.12: Python: Implementierung des Augmented Dickey-Fuller Tests

Informationskriterium zum Ermitteln der Ordnung

Funktion	get_lag_order
Beschreibung	Funktion zum Ermitteln der optimalen Lag-Ordnung für das VAR/AR-Modell.
Aufruf	Die Funktion benötigt eine Zeitreihe in Form eines pandas DataFrame und gibt die optimale Lag-Ordnung zurück. Beim Funktionsaufruf können einige Parameter angepasst werden. So kann neben dem AIC auch ein anderes Informationskriterium verwendet werden.

```
def get_lag_order(df, **kwargs):
    """
        function to determine the lag order of the model

        parameters:
            df                          data frame to be used

        kwargs:
            silent                  print output
                                        True | False
                                        DEFAULT: True

            crit                    which criterion to use
                                        'AIC' | 'BIC' | 'FPE' | 'HQIC'
                                        DEFAULT: 'AIC'

            orders                  list of orders to be used
                                        DEFAULT: [1,...,6]

            model                   model to be used
                                        'AR' | 'VAR'
                                        DEFAULT: 'VAR' (using AVG, STOCK, COUNT)

    """

#   packages
    import pandas as pd
    import numpy as np
#    from statsmodels.tools.eval_measures import aic


#   handle kwargs
    silent = True if 'silent' not in kwargs else kwargs['silent']
    crit = 'AIC' if 'crit' not in kwargs else kwargs['crit']
    orders = [i for i in range(1, 7)] if 'orders' not in kwargs else kwargs['orders']
    model = 'VAR' if 'model' not in kwargs else kwargs['model']

```

```
38 #    set up df for results
39     lag_results = pd.DataFrame({'lag': orders,
40                                 'AIC': [np.nan for i in range(len(orders))],
41                                 'BIC': [np.nan for i in range(len(orders))],
42                                 'FPE': [np.nan for i in range(len(orders))],
43                                 'HQIC': [np.nan for i in range(len(orders))]})
44     lag_results.set_index('lag', inplace = True)
45
46 #    use criterion
47     if model == 'VAR':
48         from statsmodels.tsa.api import VAR
49         model = VAR(df[['AVG', 'COUNT', 'STOCK']])
50         for i in orders:
51             model_fitted = model.fit(i)
52             lag_results.at[i, 'AIC'] = model_fitted.aic
53             lag_results.at[i, 'BIC'] = model_fitted.bic
54             lag_results.at[i, 'FPE'] = model_fitted.fpe
55             lag_results.at[i, 'HQIC'] = model_fitted.hqic
56     elif model == 'AR':
57         lag_results.drop('FPE', axis = 1,inplace = True)
58         from statsmodels.tsa.ar_model import AutoReg
59         for i in orders:
60             model = AutoReg(df[['STOCK']][:], lags = i, trend = 'n')
61             model_fitted = model.fit()
62             lag_results.at[i, 'AIC'] = model_fitted.aic
63             lag_results.at[i, 'BIC'] = model_fitted.bic
64             lag_results.at[i, 'HQIC'] = model_fitted.hqic
65
66
67 #    print output
68     if not silent:
69         print('Using information criteria for lag orders {}:'.format(str(orders)))
70         print(lag_results)
71
72 #    determine best
73     opt_lag = lag_results[lag_results[crit] == lag_results[crit].min()].index[0]
74     if not silent: print('Optimal lag by {} is {}'.format(str(crit), str(opt_lag)))
75
76 #    return
77     return(opt_lag)
```

Quellcode B.13: Python: Ermitteln der Lag-Ordnung

Vorhersage mit einem $VAR(p)$-Modell

Funktion	var_prediction
Beschreibung	Funktion zum Berechnen einer Vorhersage mit einem $VAR(p)$-Modell.
Aufruf	Die Funktion benötigt Zeitreihen in Form eines pandas DataFrame und erweitert den DataFrame um die Vorhersagewerte, basiernd auf den Modellparametern, die an die Funktion übergeben werden können.

```
def var_prediction(df, order, **kwargs):
    """
        function to determine the lag order of the model

        parameters:
            df                          data frame to be used

            order                       p of VAR(p) - order of autoregression
                                        or 'auto' if order shall be picked automatically


        kwargs:
            silent                  print output
                                        True | False
                                        DEFAULT: True

            window_mode             window mode for testing
                                        'rolling' | 'increasing'
                                        DEFAULT: 'rolling'

            window_sizes            window size for fitting
                                        pos INT
                                        DEFAULT: 20

            variables               variables to be used (if col names changed or
                                    analysis needs to be modified)
                                        [list]
                                        DEFAULT: ['STOCK', 'AVG', 'COUNT']

            crit                    information criterion to be used. obsolete
                                    as lag order is fixed and information criterion
                                    is outsourced to get_lag_order
                                        'aic' | 'bic' | 'hqic' | 'fpe'
                                        DEFAULT: 'aic'

    """

#   packages
    import pandas as pd
    import numpy as np
```

```
41      from progress.bar import ChargingBar as Bar
42      from statsmodels.tsa.api import VAR
43
44  #   handle kwargs
45      silent = False if 'silent' not in kwargs else kwargs['silent']
46      window_mode = 'rolling' if 'window_mode' not in kwargs else kwargs['window_mode']
47      window_size = 20 if 'window_size' not in kwargs else kwargs['window_size']
48      variables = ['STOCK', 'AVG', 'COUNT'] if 'variables' not in kwargs else
        kwargs['variables']
49      crit = 'aic' if 'crit' not in kwargs else kwargs['crit']
50
51
52  #   test with window
53      bar = Bar('Predicting:', max = len(df)-1-window_size)
54      for i in range(window_size, len(df)-1):
55          bar.next()
56          if window_mode == 'rolling':
57              data = df[variables][max(i-window_size, 0):i]
58          else:
59              data = df[variables][:i]
60          model = VAR(data)
61          if order == 'auto':
62              order = get_lag_order(df, crit = 'AIC')
63          model_fitted = model.fit(verbose = False, maxlags = order)
64          forecast = model_fitted.forecast(model_fitted.endog, steps = 1)
65          df.iat[i+1, df.columns.get_loc('PREDICT')] = forecast[0][0]
66      bar.finish()
67
68  #   return
69      return(df)
```

Quellcode B.14: Python: Vorhersage mit einem VAR(p)-Modell

Vorhersage mit einem $AR(p)$-Modell

Funktion	ar_prediction
Beschreibung	Funktion zum Berechnen einer Vorhersage mit einem $AR(p)$-Modell. Die Funktion wird benötigt, da die *VAR*-Klasse des Pakets *statsmodels* keine Vektorautoregression mit nur einer Variablen (d.h. eine normale Autoregression) unterstützt.
Aufruf	Die Funktion benötigt Zeitreihen in Form eines pandas DataFrame und erweitert den DataFrame um die Vorhersagewerte basierend auf den Modellparametern, die an die Funktion übergeben werden können.

```
1  def ar_prediction(df, order, **kwargs):
2      """
3          function to create an AR prediction of simply using the STOCK values
4      """
5
6
7  #   import
8      import pandas as pd
9      import numpy as np
10     from progress.bar import ChargingBar as Bar
11     from statsmodels.tsa.ar_model import AutoReg
12
13 #   handle kwargs
14     silent = False if 'silent' not in kwargs else kwargs['silent']
15     prog_bar = True if 'prog_bar' not in kwargs else kwargs['prog_bar']
16     window_mode = 'rolling' if 'window_mode' not in kwargs else kwargs['window_mode']
17     window_size = 20 if 'window_size' not in kwargs else kwargs['window_size']
18
19 #   test with window
20     if not silent: print('Using AR({}) model'.format(str(order)))
21     if prog_bar: bar = Bar('Predicting:', max = len(df)-1-window_size)
22     for i in range(window_size, len(df)-1):
23         bar.next()
24         if window_mode == 'rolling':
25             data = df['STOCK'][max(i-window_size, 0):i]
26         else:
27             data = df['STOCK'][:i]
28         if order == 'auto':
29             order = get_lag_order(df[:i], crit = 'AIC', model = 'AR')
30         model = AutoReg(data, trend = 'n', lags = order)
31         model_fitted = model.fit()
32         forecast = model_fitted.predict(start = window_size - 1, end = window_size - 1)
33         df.iat[i+1, df.columns.get_loc('PREDICT')] = forecast[0]
34     bar.finish()
35
36 #   return
```

```
37      return(df)
```

Quellcode B.15: Python: Vorhersage mit einem AR(p)-Modell

Beurteilung der Prognosegüte

Funktion	forecast_accuracy
Beschreibung	Berechnet einige Maße zur Beurteilung der Prognosegüte: MAPE (mean absolute percentage error), ME (mean error), MAE (mean absolute error), MPE (mean percentage error), RMSE (root-mean-square error), CORR (correlation coefficient), MINMAX, ACC_BIN (binary accuracy = Anzahl Vorhersagen mit richtiger Richtung). Nicht alle Maße wurden für die Fehleranalyse verwendet.
Aufruf	Die Funktion benötigt zwei Zeitreihen (Prognosewerte und echte Werte) in Form von pandas Series und gibt ein Dictionary mit den ermittelten Werten zurück.

```
def forecast_accuracy(forecast, actual, **kwargs):
    """
        function to calculate forecast accuracy measures

        parameters:
            forecast                    series with forecasted values
            actual                      series with actual values

        kwargs:
            silent                  print output
                                        True | False
                                        DEFAULT: True

            dropna                  drop NaN values in series
                                        True | False
                                        DEFAULT: True

    """

#   packages
    import numpy as np

#   handle kwargs
    silent = True if 'silent' not in kwargs else kwargs['silent']
    dropna = True if 'dropna' not in kwargs else kwaergs['dropna']


#   drop NaN values
    if dropna:
        df = pd.DataFrame({'forecast': forecast,
                           'actual': actual})
        df.dropna(axis = 0, inplace = True)

```

```
34
35 #    calculate measures
36      # mean absolute percentage error
37      mape = np.mean(np.abs(df['forecast'] - df['actual']) / np.abs(df['actual']))
38      # mean error
39      me = np.mean(df['forecast'] - df['actual'])
40      # mean absolute error
41      mae = np.mean(np.abs(df['forecast'] - df['actual']))
42      # mean percentage error
43      mpe = np.mean((df['forecast'] - df['actual'])/df['actual'])
44      # mean squared error
45      mse = np.mean((df['forecast'] - df['actual'])**2)
46      # root-mean-squared error
47      rmse = mse**.5
48      # correlation
49      corr = np.corrcoef(df['forecast'], df['actual'])[0,1]
50      mins = np.amin(np.hstack([df['forecast'][:,None],
51                                df['actual'][:,None]]), axis=1)
52      maxs = np.amax(np.hstack([df['forecast'][:,None],
53                                df['actual'][:,None]]), axis=1)
54      # minmax
55      minmax = 1 - np.mean(mins / maxs)
56
57      # binary classify of direction
58      forecast_diff = df['forecast'].diff() / df['forecast'].diff().abs()
59      actual_diff = (df['actual'].diff() / df['actual'].diff().abs())
60      df_diff = pd.DataFrame({'forecast_diff': forecast_diff,
61                              'actual_diff': actual_diff})
62      df_diff.dropna(axis = 0, inplace = True)
63      acc_bin = sum(df_diff['forecast_diff'].eq(df_diff['actual_diff'])) /
        max(len(df_diff), 1)
64
65
66 #    return
67      return({'mape':mape, 'me':me, 'mae': mae,
68      'mpe': mpe, 'mse': mse, 'rmse':rmse, 'corr':corr, 'minmax':minmax,
69      'acc_bin': acc_bin})
```

Quellcode B.16: Python: Berechnung der Prognosegüte

C. Literatur

Akaike, Hirotogu (1969). „Fitting autoregressive models for prediction“. In: *Annals of the Institute of Statistical Mathematics* 21.1, S. 243–247.

– (1973). „Information Theory and an Extension of the Maximum Likelihood Principle“. In: *Selected Papers of Hirotugu Akaike*. New York, NY: Springer New York, S. 199–213.

Balakrishnan, Vimala und Lloyd-Yemoh Ethel (Jan. 2014). „Stemming and Lemmatization: A Comparison of Retrieval Performances“. In: *Lecture Notes on Software Engineering* 2, S. 262–267.

Boiy, Erik und Marie-francine Moens (2009). „A Machine Learning Approach to Sentiment Analysis in Multilingual Web Texts“. In: *Information Retrieval*, S. 526–558.

Bollen, J., H. Mao und X. Zeng (2011). „Twitter mood predicts the stock market“. In: *Journal of Computational Science*.

DevTeam, Statsmodels (2019). *statsmodels*. `http://www.statsmodels.org/`. Version 0.11.0 Abgerufen am 15.01.2020.

Dhaoui, Chedia, Cynthia Webster und Lay Tan (Aug. 2017). „Social media sentiment analysis: lexicon versus machine learning“. In: *Journal of Consumer Marketing* 34.

Dodds, Peter Sheridan und Christopher M. Danforth (Aug. 2010). „Measuring the Happiness of Large-Scale Written Expression: Songs, Blogs, and Presidents“. In: *Journal of Happiness Studies* 11.4, S. 441–456.

Fama, Eugene F. (1965). „The Behavior of Stock-Market Prices“. In: *The Journal of Business* 38.1, S. 34–105.

Fama, Eugene F. u. a. (1969). „The Adjustment of Stock Prices to New Information“. In: *International Economic Review* 10.1, S. 1–21.

Hromkovic, Juraj (2011). *Theoretische Informatik*. Wiesbaden: Vieweg+Teubner.

Hutto, C.J. und Eric Gilbert (2019). *VADER-Sentiment-Analysis*. `https://github.com/cjhutto/vaderSentiment`. Abgerufen am 14.12.2019.

Hutto, Clayton J und Eric Gilbert (Jan. 2014). „Vader: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text“. In: *Eighth international AAAI conference on weblogs and social media*.

Ito, Kiyosi (2006). *Essentials of Stochastic Processes*. Providence: American Mathematical Society.

Kanayama, Hiroshi und Tetsuya Nasukawa (Juli 2006). „Fully Automatic Lexicon Expansion for Domain-oriented Sentiment Analysis“. In: *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*. Sydney, Australia: Association for Computational Linguistics, S. 355–363.

Kiss, Tibor und Jan Strunk (2006). „Unsupervised Multilingual Sentence Boundary Detection“. In: *Computational Linguistics* 32.4, S. 485–525.

Kolchyna, Olga u. a. (2015). *Twitter Sentiment Analysis: Lexicon Method, Machine Learning Method and Their Combination*.

Kunze, Claudia und Lothar Lemnitzer (2002). „GermaNet – representation, visualization, application“. In: *Proceedings of LREC 2002, main conference, Vol V*, S. 1485–1491.

Liu, Jingjing und Stephanie Seneff (Jan. 2009). „Review Sentiment Scoring via a Parse-and-Paraphrase Paradigm.“ In: S. 161–169.

Lütkepohl, Helmut (2005). *New Introduction to Multiple Time Series Analysis*. Berlin Heidelberg: Springer Science & Business Media.

Manning, Christopher D., Prabhakar Raghavan und Hinrich Schütze (2008). *Introduction to Information Retrieval*. USA: Cambridge University Press.

Mao, Yuexin u. a. (2012). „Correlating S&P 500 Stocks with Twitter Data“. In: *Proceedings of the First ACM International Workshop on Hot Topics on Interdisciplinary Social Networks Research*. HotSocial '12. Beijing, China: Association for Computing Machinery, S. 69–72.

Marcus, Mitchell P., Beatrice Santorini und Mary Ann Marcinkiewicz (1993). „Building a Large Annotated Corpus of English: The Penn Treebank“. In: *Computational Linguistics* 19.2, S. 313–330.

Murphy, John J. (2014). *Technische Analyse der Finanzmärkte: Grundlagen, Strategien, Methoden, Anwendungen. Inkl. Workbook. Grundlagen, Strategien, Methoden, Anwendungen. Inkl. Workbook*. Übers. von Hartmut Sieper. 11. Aufl. FinanzBuch Verlag.

Nofsinger, John (Apr. 2003). „Social Mood and Financial Economics“. In: *Journal of Behavioral Finance* 6.

Pak, Alexander und Patrick Paroubek (Mai 2010). „Twitter as a Corpus for Sentiment Analysis and Opinion Mining“. In: *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*. Valletta, Malta: European Language Resources Association (ELRA).

Pelat, Camille u. a. (Aug. 2009). „More Diseases Tracked by Using Google Trends“. In: *Emerging infectious diseases* 15, S. 1327–8.

Pfaffenberger, F. (2016). *Twitter als Basis wissenschaftlicher Studien. Eine Bewertung gängiger Erhebungs- und Analysemethoden der Twitter-Forschung*. Springer VS.

Qian, Bo und Khaled Rasheed (Feb. 2007). „Stock market prediction with multiple classifiers“. In: *Appl. Intell.* 26, S. 25–33.

Rosenblatt, F. (1958). „The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain“. In: *Psychological Review*, S. 65–386.

Said, Said E. und David A. Dickey (1984). „Testing for Unit Roots in Autoregressive-Moving Average Models of Unknown Order“. In: *Biometrika* 71.3, S. 599–607.

Schiller, Anne, Simone Teufel und Christine Thielen (1995). „Guidelines f ur das Tagging deutscher Textcorpora mit STTS“. In: *Universität Stuttgart, Universität Tübingen, Germany*.

Services, Colt Technology (2013). *Stock prices influenced by Twitter and Facebook, according to UK finance professionals*. `https://www.colt.net/resources/stock-prices-influenced-by-twitter-and-facebook-according-to-uk-finance-professionals/`. Abgerufen am 03.12.2019.

Shittu, Olanrewaju (Jan. 2009). „Comparison of Criteria for Estimating the Order of Autoregressive Process: A Monte Carlo Approach“. In: 30, S. 1450–216.

Sims, Christopher A. (1980). „Macroeconomics and Reality“. In: *Econometrica* 48.1, S. 1–48.

Soergel, Dagobert (Okt. 1998). *WordNet. An Electronic Lexical Database*. Cambridge, MA: MIT Press.

Statistisches Bundesamt (2019). *Most popular social networks worldwide as of October 2019, ranked by number of active users*. https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/. Abgerufen am 18.12.2019.

Stone, Philip J. (1966). *The General Inquirer: A Computer Approach to Content Analysis*.

Taboada, Maite u. a. (Juni 2011). „Lexicon-Based Methods for Sentiment Analysis“. In: *Computational Linguistics* 37, S. 267–307.

Turney, Peter D. (2002). „Thumbs up or Thumbs down? Semantic Orientation Applied to Unsupervised Classification of Reviews“. In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. ACL '02. Philadelphia, Pennsylvania: Association for Computational Linguistics, S. 417–424.

Varian, Hal und Hyunyoung Choi (Apr. 2011). „Predicting the Present with Google Trends“. In: *Economic Record* 88.

Zhang, Xue Sophia, Hauke Fuehres und Peter A. Gloor (2011). „Predicting Stock Market Indicators Through Twitter“. In: *Procedia - Social and Behavioral Sciences*. Bd. 26, S. 55–62.