TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

Diplomarbeit

STOCK PRICE PREDICTION BASED ON A SENTIMENT ANALYSIS OF FINANCIAL NEWS

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs (Dipl.-Ing oder DI), eingereicht an der TU Wien, Fakultät für Maschinenwesen und Betriebswissenschaften, von

Stefan Salbrechter

Matr.-Nr.: 1327435

unter der Leitung von

Ao.Univ.Prof. Mag.rer.nat. Dr.rer.soc.oec. Dr.techn. Thomas Dangl

Institut für Managementwissenschaften
Finanzwirtschaft & Controlling

Wien, Juni 2020

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass die vorliegende Arbeit nach den anerkannten Grundsätzen für wissenschaftliche Abhandlungen von mir selbstständig erstellt wurde. Alle verwendeten Hilfsmittel, insbesondere die zugrunde gelegte Literatur, sind in dieser Arbeit genannt und aufgelistet. Die aus den Quellen wörtlich entnommenen Stellen, sind als solche kenntlich gemacht. Das Thema dieser Arbeit wurde von mir bisher weder im In- noch Ausland einer Beurteilerin/einem Beurteiler zur Begutachtung in irgendeiner Form als Prüfungsarbeit vorgelegt. Diese Arbeit stimmt mit der von den Begutachterinnen/Begutachtern beurteilten Arbeit überein.

Stadt und Datum

Unterschrift

Acknowledgements

I would especially like to thank my supervisor Univ.Prof. Thomas Dangl, for making this master's thesis possible and for his guidance, help and constant support during the entire research work. We also thank Refinitiv for providing us with a comprehensive collection of financial news data covering the range from January 1996 to January 2020. Furthermore, I would also like to express my sincerest gratitude to my parents, Renate and Dietmar Salbrechter, for their invaluable support during my entire education. Moreover, I thank my sisters Ina and Bettina Salbrechter as well as all my friends and fellow students for their support and the great time.

Kurzfassung

Das Ziel dieser Arbeit ist es, eine Stimmungsanalyse von Finanznachrichten durchzuführen um festzustellen ob diese Nachrichtendaten für die kurzfristige Vorhersage von Aktienkursbewegungen geeignet sind. Der Datensatz enthält alle Finanznachrichten[1], welche von Thomson Reuters zwischen Januar 1996 und Januar 2020 veröffentlicht wurden. Zusätzlich werden insgesamt 921 Indexkonstituenten und ihr Total-Return-Index (ein Performanceindex, der Dividenden und andere performancerelevante Kapitalmaßnahmen berücksichtigt) aus Refinitiv DataStream abgerufen. Darüber hinaus werden alle 1220 Unternehmen, die in diesem Zeitraum im S&P 500 indexiert sind, für die Aktienkursprognose berücksichtigt.[2]

In einem ersten Schritt werden alle Nachrichtenartikel, die sich auf diese 921 Unternehmen beziehen, aus dem Nachrichtendatensatz extrahiert. Anschließend werden aus deren Total-Return-Indizes tägliche Renditen berechnet, die zur Kennzeichnung der Nachrichtenartikel als positiv, negativ oder neutral verwendet werden. Um einen besseren Einblick in den Datensatz zu erhalten, werden diverse Datenvisualisierungen durchgeführt. Wörter, die Stimmungsinformationen tragen, werden extrahiert und über Wortwolken visualisiert.
Für die Klassifizierung werden verschiedene neuronale Netzwerkarchitekturen betrachtet. Genauer gesagt werden sowohl klassische feedforward-Netze (NN) als auch Convolutional Neural Networks implementiert und deren Ergebnisse verglichen. Darüber hinaus werden zwei verschiedene Verfahren zur Merkmalsextraktion, nämlich Bag-of-Words und Worteinbettungen, weiter untersucht.
Worteinbettungen werden mit word2vec - einem selbstüberwachten Lernalgorithmus[3] - auf den Trainings- und Validierungsdatensätzen erlernt. Da diese Worteinbettungen Stimmungsinformationen nicht hinreichend erfassen, erfolgt eine Verfeinerung dieser Einbettungen über einen trainierbaren Einbettungslayer eines neuronalen Netzes. Zusätzlich werden n-Gramme mit Längen von eins bis drei berücksichtigt, um den Bag-of-Word-Ansatz weiter zu verbessern.
Beim anschließenden Training der neuronalen Netze wurde beobachtet, dass sie zur Überanpassung der Daten neigen, wenn die Merkmalsvektoren groß sind. Folglich führte die Verkleinerung der Merkmalsvektoren von 50000 auf 25000 zu einer Erhöhung der Genauigkeit und der Trainingsgeschwindigkeit, während die Neigung zur Überanpassung der Daten deutlich verringert wurde.
Die Leistungsfähigkeit der verschiedenen Modelle und ihre Fähigkeit, Aktienkurse vorherzusagen, wird mit einem Handelsalgorithmus bewertet. Die Daten von Januar 1996 bis

[1] Der Datensatz wurde von Refinitiv dem Institut für Managementwissenschaften der TU Wien zu Forschungszwecken zur Verfügung gestellt und enthält insgesamt 74.382.002 Artikel. Ich danke Refinitiv für die Überlassung des umfassenden Datensatzes.

[2] Die Daten wurden für insgesamt 1220 Unternehmen heruntergeladen. DataStream konnte jedoch mehrere Ticker-Codes von hauptsächlich alten, nicht mehr existierenden Unternehmen nicht finden. Diese Unternehmen wurden entfernt, um eine zeitaufwändige, manuelle Suche zu vermeiden. Infolgedessen blieben 921 Unternehmen übrig (siehe Abschnitt 2.1.1).

[3] Ob word2vec ein überwachter oder unüberwachter Lernalgorithmus ist, hängt davon ab, wie es betrachtet wird. Er ist unüberwacht, da kein Mensch erforderlich ist, um Daten zu kennzeichnen. Der Algorithmus selbst erstellt jedoch seine eigenen Labels und optimiert durch Backpropagation. Er kann also als selbstüberwachtes Lernen betrachtet werden.

Dezember 2017 werden zum Training und zur Validierung verwendet. D.h. der Algorithmus wird darauf trainiert, einen Nachrichtentext mit einem Stimmungsmaß zu versehen, welches die Wahrscheinlichkeit charakterisiert, dass der Aktienkurs der zugehörigen Firma am kommenden Tag steigen wird. Der Zweijahreszeitraum zwischen Januar 2018 und Januar 2020 wird für das Testen und Bewerten mit dem Handelsalgorithmus verwendet. Der Handelsalgorithmus analysiert alle veröffentlichten Nachrichten jeden Tages und weist jedem Artikel eine Stimmungsbewertung zu. Da Nachrichtenartikel vorausschauende Informationen über Aktienrenditen enthalten und das Stimmungsmaß in der Lage ist, diese Informationen (teilweise) zu erfassen, sollte eine positive Nachrichtenstimmung einen Anstieg des Aktienkurses vorhersagen und eine negative Stimmung einen fallenden Aktienkurs im Laufe des nächsten Tages vorhersagen. Dies wird über den Evaluierungszeitraum getestet. Daher kauft der Algorithmus jeden Tag Aktien mit positiver Nachrichtenstimmung und verkauft Aktien mit negativer Stimmung. Die Funktionsweise des Algorithmus wird im Abschnitt 3.5 näher beschrieben.

Schließlich wurde eine risikobereinigte Benchmark auf der Grundlage des Capital Asset Pricing Model (CAPM) verwendet, um die Leistung des Handelsalgorithmus zu ermitteln. Als Ergebnis dieser Arbeit wurde festgestellt, dass der Handelsalgorithmus in der Lage ist, die Benchmark deutlich zu übertreffen. Dies liefert den empirischen Beweis dafür, dass (i) Nachrichtenartikel mit prädiktiver Information ausgestattet sind und (ii) die vorgestellten Algorithmen in der Lage sind, ein Stimmungsmaß aus Artikeln zu extrahieren, welche prädiktive Informationen enthalten, die weiters für die Portfolioauswahl verwendet werden können.

Abstract

The aim of this thesis is to conduct a sentiment analysis of financial news articles, published by Thomson Reuters, in order to determine whether this news data is suitable for the short-term prediction of stock price movements. The dataset[4] contains all financial news articles published by Thomson Reuters[5] between January 1996 and January 2020. Additionally, a total of 1220 index constituents and their total return index (a performance index that considers dividends and other performance relevant capital measures) are retrieved from Refinitiv DataStream.[6]

In a first step all the financial news articles related to those 921 companies are extracted from the news dataset. Then, daily returns are calculated from their total return indices which are used to label news articles as positive, negative or neutral.
In order to get a better sense of the data set, several data visualisation tasks are performed. Words that carry sentiment information are extracted and visualised via word clouds.
For the classification task different neural network architectures are considered. To be more precise, classical feed-forward neural networks (NN) as well as convolutional neural networks are implemented and their results are compared. Furthermore two different feature extraction methods, namely bag-of-words and word embeddings are further investigated.
Word embeddings are learned with word2vec - a self-supervised learning task[7] - on the training and validation data sets. Since these word embeddings don't capture sufficient sentiment information, a refinement of those embeddings is carried out via a trainable embedding layer of a neural network. Apart from that, n-grams with lengths from one to three are considered in order to enhance the bag-of-word approach.
Subsequently, during the training of the neural networks, it was observed that they tend to overfit the data when the feature vectors are large. Consequently, minimizing the size of the feature vectors from 50000 to 25000 led to an increase in accuracy and training speed, while the tendency to overfit the data was significantly reduced.
The performance of the different models and their ability to predict stock prices is evaluated with a trading algorithm. Data from January 1996 until December 2017 is used for training and validation. I.e., the algorithm is trained to assign a sentiment measure to a news text which characterizes the likelihood that the associated firm's share price will rise over the coming day. The two-year period between January 2018 and January 2020 is used for testing and evaluation with the trading algorithm. The trading algorithm analyses all

[4]Refinitiv provided the institute of Management Science of TU Wien with this dataset for research purposes. The dataset consists of 74,382,002 individual news items. I thank Refinitiv for providing us with this comprehensive collection of financial news data.
[5]The Financial and Risk business of Thomson Reuters is now Refinitiv
[6]Data was downloaded for a total of 1220 companies. However, DataStream could not find several ticker codes of mainly old, no longer existing companies. These companies were removed to avoid a time-consuming manual search. As a result, 921 companies remained (see Section 2.1.1).
[7]In more detail whether word2vec is a supervised or unsupervised learning algorithm depends on how one looks at it. It is unsupervised since no human is required to label any data. The algorithm itself however, creates its own labels and optimizes via back propagation. So it can be seen as self-supervised learning.

published news of each day and assigns every article a sentiment score. Given that news articles contain predictive information about stock returns and the sentiment measure is able to capture (part of) this information, positive news sentiment should predict a rise in the stock price and negative sentiment should predict a falling stock price over the following day. This is tested over the evaluation period. Therefore, the algorithm buys stocks of positive news sentiment and sells stocks with negative sentiment each day. The functioning of the algorithm is described in more detail in Section 3.5.

Finally, a risk-adjusted benchmark based on the Capital Asset Pricing Model (CAPM) was used to determine the performance of the trading algorithm. As a result of this work it was found that the trading algorithm is able to significantly outperform the benchmark. This gives empirical evidence that (i) news articles come with predictive information and (ii) the presented algorithms are able to extract a sentiment measure from articles that contain predictive power which can be used in a portfolio selection.

Contents

List of Figures

Chapter 1

Introduction

A stock market is a large, complex information processing system that is influenced by many different socio-economic factors. Today, the financial markets are explained to a great extent by the traditional financial theory. It is built upon the work of some famous economists. Merton Miller and Franco Modigliani formulated the arbitrage principles, Harry Markowitz established the modern portfolio theory, John Lintner and William Sharpe developed the capital asset pricing model (CAPM) and Fischer Black, Myron Scholes and Robert Merton are the originators of the option-pricing theory (Statman, 1995).

Further important contributions to traditional financial theory came from E. Fama. He developed the theory that stock prices follow a random walk. Fama is also associated with the efficient market hypothesis (EMH) which states, that all available information is fully reflected in the asset prices (Malkiel and Fama, 1970). It hypothesizes that stocks are always trading on their fair value which implies that it is impossible to outperform the market over the long term (Downey, 2003). Basically, the modern financial economic theory relies on the fundamental assumption, that the "representative agent" is rational, unbiased and makes decisions according to the axioms of utility theory (Thaler, 1999).

However, in the real markets, many participants don't act fully rationally and unbiased. Many studies have documented long-term historical developments in securities markets that are contrary to the hypothesis of an efficient market. Bubbles and deep recessions are such market events, that can not be explained with the standard financial theory (Kenton, 2003). Behavioral Finance aims to complement traditional financial theory by assuming that psychological influences play a significant role in the actions of market participants (Herschberg, 2012). Furthermore, Baker and Wurgler (2007) have found, that investor sentiment can be linked to price movements of the stock market. According to Tetlock (2007) financial news influence psychology and sociology of investors. This led to the finding that financial news can be used to successfully predict stock price movements. In the upcoming years, numerous studies have been conducted in order to examine to which extent financial news are able to explain and predict stock price movements. Some of them are described in Section 1.3.

1.1 Using Text as Data

With the rise and steady growth of the Internet, the amount of available digital text has become increasingly large. According to a study conducted by IBM in 2017, 90% of the data has been created in the last two years (Petrov, 2019). Large quantities of this data has been published in the form of written text. While it is an easy task for humans to understand this type of data, it is much more difficult for computers to handle it. However, advances in computing power and progress in machine learning and AI in recent years have significantly improved the ability of computers to handle this form of unstructured data.
The number of researchers using text as data has increased substantially within the last years. Textual data is used in macroeconomics in order to predict variation in inflation and unemployment rates. It is used by marketing departments to analyse the text from ads and reviews in order to get a deeper understanding of consumer decision making. In the field of political economy, texts are used to interpret political agendas and debates. As discussed in this paper, textual data in the form of financial news, social media posts or company filings is also used to predict stock market activity and to study the impact of new information on the market (Gentzkow, Kelly, and Taddy, 2017).

Conducting a sentiment analysis in the field of finance has to be distinguished from a classical sentiment analysis. In a financial context, the vocabulary containing sentimental information is very different from other domains. This is the reason why classic sentiment analysis models often perform badly in a financial context. Early work on this topic of sentiment analysis largely relied on hand designed sentiment dictionaries. However, this approach comes along with the disadvantage that sentiment lexicons are domain-specific and therefore not well suited for certain tasks. Moreover, they are limited in scope and do not contain the full range of words that convey sentimental information. In the following years, researchers found that machine learning algorithms are capable of improving performance in contrast to dictionary-based approaches. However, supervised machine learning methods need large amounts of labelled data to achieve good results. Since manual labelling of large quantities of financial news is costly, there is a lack of available large-scale labelled records. Therefore, the approach from Kelly, Ke, and Xiu (2019) was adopted in this thesis, which makes use of the joint behaviour of financial news and stock returns to label financial news articles.

Furthermore, an important task in textual analysis is the transformation of text in a machine readable form. The most common approach is called 'bag-of-words' which represents texts by its individual words without considering word order (see Section 2.2.3.1). Although it may be difficult for people to understand a text without a specific word order, these algorithms perform quite well, but still leave a lot of space for improvement. Today, modern approaches are largely based on word embeddings. Word embeddings are dense vector representations of words that carry semantic and syntactic information. Thus, similar words have similar vector representations (see Section 2.2.3.5). Furthermore, word embeddings are computationally more efficient than bag-of-word models due to the dense representation.
Earlier work often used classifiers such as Naive Bayes, Support Vector Machines (SVM), k-nearest neighbors (KNN) and random forest for sentiment analysis (Rechenthin, Street, and Srinivasan, 2013). In recent years, however, the best performance has been achieved with different architectures of deep neural networks. Especially convolutional neural networks perform very well in combination with word embeddings (Severyn and Moschitti, 2015; Vargas, Lima, and Evsukoff, 2017).

The textual data used for this thesis is derived from a large dataset that covers all financial news published by Thomson Reuters between 1996 and 2020. The overall size of the dataset exceeds 50 GB of .gzib compressed files, totalling to 74,382,002 individual news items. For sentiment analysis, however, only news articles related to companies indexed in the S&P 500 are considered, which makes up a total of 921 companies. This thesis focuses on the application of neural network classifiers for sentiment prediction. Different models, which use bag-of-words as well as word embeddings as input features, are implemented in order to find the most suitable approach. The network architectures used in this thesis are classic feedforward neural networks (NN) (see Section 2.4.3) and convolutional neural networks (CNNs) (see Section 2.4.4). In order to evaluate the performance of the different input features and network architectures, a trading algorithm is implemented and tested on news data between January 2018 and January 2020.

1.2 Aim of this Work

This thesis is conducted in order to find answers to the following research questions:

(i) Are the selected algorithms able to extract a measure of sentiment from financial news that shows a positive correlation with out-of-sample stock returns. In other words, does this news contain predictive information about future stock returns.

(ii) Is it possible to implement an event-driven trading strategy that benefits from an inefficient stock market and generates a positive alpha by using financial news as a predictor of stock price movements.

(iii) Is this strategy able to outperform the risk adjusted benchmark, described in Section 3.6.1, when run on the evaluation dataset.

1.3 Research Background and Related Work

Malkiel and Fama (1970) found evidence that it is possible to predict stock price movements with past information for time frames of less than one day. The authors were able to develop a trading strategy that is slightly profitable as long as no transaction costs are taken into account. In the long term, however, they state, prices follow a random walk and can therefore not be predicted. In addition, Antweiler and Frank (2004) attempts to find an answer to the question of whether posts on Internet stock message boards affect the stock market. The authors are able to predict negative returns one day in advance. Although the results are statistically significant, performance is too weak to generate returns that exceed transaction costs. Further important research in this area was carried out by Baker and Wurgler (2007). In their frequently cited study, they derived a sentiment indicator from six proxies using principal component analysis. The proxies they used are the trading volume measured by NYSE turnover, dividend premium, closed-end-fund discount, number of first day returns on IPOs and the equity share in new issues. The authors found, that low capitalization, young, unprofitable, low-dividend paying, high-volatility and growth stock companies are difficult to arbitrage or to value according to the traditional financial theory and are therefore very sensitive to investor sentiment.

Another important contribution to this field of research was provided by Tetlock (2007), who argued that financial news influences investor sentiment. In his work he studied the effects of news media on the stock market and found evidence that news media can be used to predict price changes on the financial markets. For this study, he derived news data from the Wall Street Journal (WSJ) over a 16-year period from 1984 until 1999. Furthermore, he made use of the General Inquirer's Harvard IV-4 psychosocial dictionary which consists of 77 word categories. News are converted into a numerical representation by counting the number of words that fall into different word categories. Subsequently, he used basic vector autoregression (VARs) to find a correlation between pessimism in media and stock market activities. The results of this study show, that news can be used to predict price movements. In addition, he found that high or low values of the media pessimism factor result in high trading volume. Dictionary-based approaches are widely used to determine the sentiment of written text. Another dictionary-based study was conducted by Bollen and Mao (2011), which uses Twitter sentiment as a predictor of daily price fluctuations in the Dow Jones Industrial Average (DJIA). The authors collected 9,853,498 public tweets posted between February 28^{th} and December 19^{th} 2008. Sentiment analysis is done with the tools GPOMS and OpinionFinder. GPOMS measures six different dimensions of mood: calm, alert, sure, vital, kind, happy and opinion finder classifies text into positive and negative classes. Their results support the assumptions of behavioral finance regarding the influence of emotions on financial markets. Loughran and McDonald (2011) showed that the commonly used Harvard IV-4 dictionary is not appropriate for textual analysis in the field of financial analysis. The authors examined 10-K financial reports and found that almost four-thirds of negatively classified words in the Harvard IV-4 dictionary are not negative in a financial context. To overcome high misclassification rates, the authors created a new dictionary suitable for the purpose of financial media. Furthermore they implement a term weighted scheme to reduce the impact of high frequency terms and amplify low frequency terms. Thereby the authors are able to reduce noise caused by misclassification. Jegadeesh and Wu (2013) refined this approach by determining term weights via market reactions. Therefore the authors implement a regression analysis and use return data on the 10-K publication dates to find individual word weights. This approach resulted in much better correlations between document tone and market reactions for positive terms.

Rechenthin, Street, and Srinivasan (2013) aimed to predict stock prices of eleven well known stocks via linguistic data from the Yahoo Finance message boards. Instead of relying on pre defined sentiment dictionaries, the authors used five different supervised learning algorithms to determine the sentiment of the posts. The algorithms included support vector machines (SVM), naive bayes, boosted decision tree, decision table and k-nearest neighbor (kNN). The textual data was transformed into a bag-of-word representation with bigrams as features. This approach is explained in more detail in Section 2.2.3.1. Furthermore, stemming was implemented to reduce the size of the vocabulary. The best results were achieved with boosted decision trees. The sentiment derived from this model was then utilized to determine the sentiment of the posts in the test-dataset. This in turn, was used as input for a neural network classifier with one hidden layer to predict future up and down movements of the stocks. The authors were able to predict price movements on the following trading day with an accuracy of 62.86%. Additionally, H. Lee et al. (2014) utilized 8-K reports of S&P 500 companies to predict stock price movements. The authors used a combination of numeric and linguistic data. Textual data was transformed into a bag-of-words representation with unigrams as features. A random forest classifier with 2000 trees was then trained

with 21 numerical and 2319 linguistic features. As a result, the prediction accuracy of price movements one day ahead was improved by 10% when textual data was included relative to a baseline that only considered financial features.

In more recent papers a shift from sparse bag-of-words representations to dense vector representations with word embeddings as features can be observed. Severyn and Moschitti (2015) used word embeddings in combination with a deep convolutional neural network (DCNN) for a sentiment analysis of tweets. The network architecture proposed in this paper is used in a similar form in this thesis in Section 2.4.4. With this approach a new state-of-the-art at the phrase level subtask of the 2015 SemEval[1] challange was achieved. Furthermore, Ding et al. (2015) implemented an event driven approach for stock price prediction. Events were retrieved from news articles and transformed to dense vectors which were fed into a neural tensor network. As an example, the phrase "Microsoft sues Barnes&Noble" is transformed in a structured event representation as (Actor = Microsoft, Action = sues, Object = Barnes&Noble). As a result, the authors found that events are more valuable features than words. With this set-up, a profitable trading strategy was achieved.
Vargas, Lima, and Evsukoff (2017) compared different neural network architectures for the task of predicting intra day price movements of the S&P 500. Therefore word embedding features were derived from news data, published by Thomson Reuters between October 2006 and November 2013. In total, 106,494 news articles are considered. The authors found, that a recurrent convolutional neural network (RCNN) slightly beats a convolutional neural network (CNN). They also noted an improvement of sentence embeddings compared to word embeddings. The performance further increased when linguistic features were combined with a set of technical features which were derived from historical price data.

In a recent paper Kelly, Ke, and Xiu (2019) published a novel machine learning technique that constructs sentiment scoring models out of text corpora without relying on pre-existing dictionaries. The core idea is to learn the sentiment scoring model via a common behaviour of stock returns and news articles. This idea is adopted in this thesis for labelling news articles as positive, neutral or negative. The authors used news data from the Dow Jones Newswires in the period from January 1989 until July 2017 for their analysis. They compared the effects of fresh and stale news and found, that the impact of sentiment is 70% larger for fresh news. While fresh news take four days to be fully reflected into stock prices, it takes just two days for stale news. They also find that the price responses are approximately four times larger for small and volatile stocks than for stocks with high market capitalisation. These results are consistent with the findings of Baker and Wurgler (2007). For large stocks, it takes one day for the new information to be fully reflected in the prices, while for small stocks it takes three days. It is argued, that large stocks receive more attention from investors, which leads to quicker price responses. Furthermore, since smaller stocks are less liquid, higher transactions costs associated with trading on information related to small stocks might further slow down price reactions. These results raise doubts about the validity of the efficient market theory, which states that new information is immediately reflected in prices.
In addition, the authors implemented a simple trading strategy that buys stocks if the news sentiment is positive and sells stocks if the news sentiment is negative. The results show, that this approach is capable to outperform commercially available sentiment metrics.

[1]SemEval (the International Workshop on Semantic Evaluation) is an ongoing series that focuses on the evaluation of computer-based semantic systems. It is organized under the umbrella of SIGLEX, the Special Interest Group on the Lexicon of the Association for Computational Linguistics.

Chapter 2

Methodology

This chapter describes all the steps needed to obtain a sentiment prediction model applied to the Thomson Reuters news article dataset. The dataset contains all financial news published between January 1996 and January 2020 by Thomson Reuters. The following section describes the data mining tasks that are conducted to extract all relevant news articles. Furthermore, Section 2.1.2 deals with the labelling of news articles by return data, which is a necessary prerequisite for the supervised learning task. The different features and the way they are extracted from the dataset are described in Section 2.2. Section 2.3 gives a better insight into the dataset by extracting words that carry sentimental information. These words are visualized via word clouds. Subsequently, Section 2.4 describes the different machine learning architectures in detail.

2.1 Data Mining

Data mining describes the task of filtering out useful information from a large amount of data. A special field of data mining is known as text mining, which is defined as a process of editing, organizing, and analysing large amounts of textual data. It has the purpose to provide and filter distinct information to discover the characteristics and relationships of those features (Sullivan, 2001).

2.1.1 News Data

The news data is stored in the JavaScript Object Notation (JSON) format and compressed as gzip. The entire dataset is split into 289 monthly files with an average size of 174 MB, resulting in a total size of about 50 GB. It contains news articles in several languages like English, German, Chinese, etc. and covers a wide range of topics. For this sentiment analysis, however, only English-language articles referring to companies listed in the Standard & Poor's (S&P) 500 stock market index between January 1996 and January 2020 are considered. The index consists of 500 major US equity companies, while the portfolio structure of the index changes over time. Some companies are removed from the index, while others are included. For this reason, a total of 1220 constituents must be considered.

Each article in the dataset is marked with several tags which include the time stamp, the language of the article, a list of topic subjects and also the ticker codes of companies in the case of business related news. Finally, in order to extract the company news, a list of all companies and their ticker codes were downloaded from Refinitiv DataStream. It turned

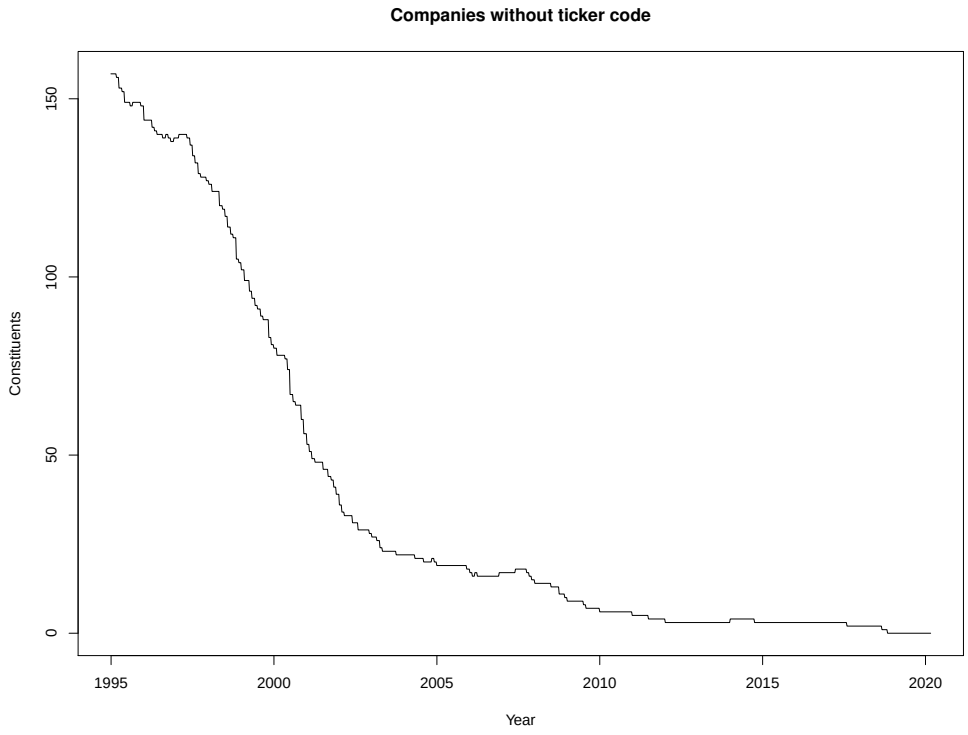


Figure 2.1: Number of companies in the S&P 500 without ticker code as a function of time

out that not all companies can be linked to ticker codes. In particular, many codes from old companies, which no longer exist today, are missing. A total of 299 constituents were not assigned to ticker codes. Figure 2.1 shows the number of missing ticker codes as a function of time. Nevertheless, a very time-consuming search for missing ticker codes was not conducted. Instead, only the remaining 921 companies with available ticker codes are considered. As a consequence, this leads to a survivorship bias since many companies dropped from the index due to a negative performance, and firms with positive performance (i.e. positive news) remained in the index. However, this only affects the training data set, as almost all constituents are available after 2010. The survivorship bias can therefore influence the training of the models, but does not distort the results of the test data set in the out-of-sample evaluation.

In addition, the total return indexes of all companies are also downloaded from DataStream for the entire period. The data mining tasks are performed with the open source language R, since it is a very powerful language for handling large amounts of data. Furthermore, the natural language processing (NLP) tasks and the meachine learning models are implemented in Python.

The R code for loading a data file into the workspace is as followed:

[63]: data <- fromJSON(filename, simplifyVector = FALSE)

If the argument 'simplifyVector' is set to TRUE, R creates a data frame object that is convenient and clear. But this is a very time-consuming task. Loading a single file took almost 10 minutes and the entire dataset, which contains 289 files, would take about 47 hours.

To overcome this issue, the 'simplifyVector' argument is set to FALSE, which loads the data as a nested list object. This is less convenient to handle, but the loading time decreased to only 30 seconds with the same hardware setup. Several selectors are then used to extract business information. The first selector extracts all English articles. In addition a second selector is used to extract news articles about each individual company in the S&P 500. The index composition is updated weekly to ensure that only news articles of companies that are actually indexed in the S&P 500 are loaded.

Thomson Reuters uses subjects in order to assign news articles to specific topics and companies. Each news article that refers to a listed company contains a ticker code subject. For Apple.Inc this code is "R:AAPL.N", where R denotes RIC for Reuters Instrument Code, "AAPL" is Apple's ticker code followed by a letter that represents the stock exchange on which the company is traded. In this case, "N" stands for the New York Stock Exchange. The news data for January 2010 counts in total 169,381 English articles with an average number of 12.02 subjects for each article. The algorithm checks whether a company ticker code appears in the subject list and assigns the article to each company that is on the list. This task is computationally expensive, since it iterates over a total of 289 month (01-01-1996 until 31-01-2020) and all 921 tickers.

Furthermore, Thomson Reuters regularly publishes updated news articles with additional information. In order to avoid duplicate text, only the latest updated version of the news articles is extracted from the dataset. Finally, the extracted news articles, consisting of the article headline and the article body, are linked to the publication date and the company ticker. All news articles of one day are then combined into one document for each company. Although only company-related news articles are extracted, some news articles contain very little information about a particular company. These are often longer essays dealing with several different firms. This is a poor precondition for sentiment analysis. An article may contain positive sentiment about one company and negative sentiment about another one. This can lead to an overall neutral sentiment of the document despite company-specific polarities. To achieve meaningful results, it is important that the news actually relates to the company under consideration.

Several approaches are possible to overcome this issue:
One possibility is to consider only short news articles with a body length below a maximum number of words. This increases the likelihood that the article actually deals with the company it is related to. With this strategy, however, a lot of data hidden in longer texts, which may contain relevant information, gets lost.
Another, more advanced possibility is to check where the company name appears in the text. Text passages that frequently contain the company name should be extracted while paragraphs that don't include the company name should be discarded. Nevertheless, these paragraphs could still contain useful sentiment information. Overall, this approach may be a good choice, but a simpler method is chosen for this work.
In this thesis, relevant articles are identified by counting the occurrences of the company name (or the ticker) in the headline and body of news articles. If the company name appears in the headline, the article is considered as relevant. Furthermore, the density of the company name (or the ticker) in the body text is calculated with Formula 2.1.

$$\text{density} = \frac{k}{l} * 100 \tag{2.1}$$

with:

k ... quantity of the company name occurrences in the text
l ... word count of the text

The value of the minimum density required to rate an article as relevant was chosen to be 2%. For later tasks, only articles with firm specific RIC codes in the subject field, the company name in the headline or a density greater than the minimum density are used.

In addition, when counting the occurrences of the company name, several aspects must be taken into account. First, some company names consist of two or more words (e.g. "Ford Motor") but authors often use just one word (e.g. "Ford") in the article. Therefore the algorithm splits the company names into word tokens and checks if any of those tokens appears in the text. If the company name consists of several words, the algorithm considers only the first two words. Furthermore, it is necessary to recognize if the full company name appears in the text to avoid counting it twice. For example, if the author writes in his article "Ford Motor" the algorithm would find the first word of the company name "Ford" as well as the second word "Motor" in the text and would count two occurrences. So if the author writes the full company name, the algorithm checks if the first word of the company name is followed by the second word and counts just one occurrence in this case. It is also necessary to consider company names consisting of words that are not strictly related to the company but have ambiguous meanings. For example, splitting the company name "NRG Energy" into the word tokens "NRG" and "Energy" is problematic. The term "Energy" frequently appears in the text without explicitly referring to the company NRG Energy. For this case, the term "Energy" is deleted and only the term "NRG" is used.

After the filter task, the size of the data set decreased drastically. An uncompressed .Rdata file containing all business related news between January 2010 and December 2011 has a size of 241 MB before and a size of 16 MB after filtering relevant news. Figure 2.2 shows the number of annual news articles from 1996 to 2020 after filtering relevant news. One can observe that in the years between 1996 and 2004 the number of relevant news articles is quite low. This is due to the fact that Thomson Reuters published fewer news articles at that time and the news articles were linked to fewer subjects, which makes it difficult to relate those to companies. After that, there was a peak in 2008 until 2009 which is presumable due to the financial crisis in 2008. Thereafter, the number of annual news articles is fairly constant. In addition, the cumulative number of news articles is shown in Figure 2.3. It can be observed that the total number of news articles sums up to 1,842,547 articles. Additionally, Figure 2.4 displays the annual number of companies for which news data was available.

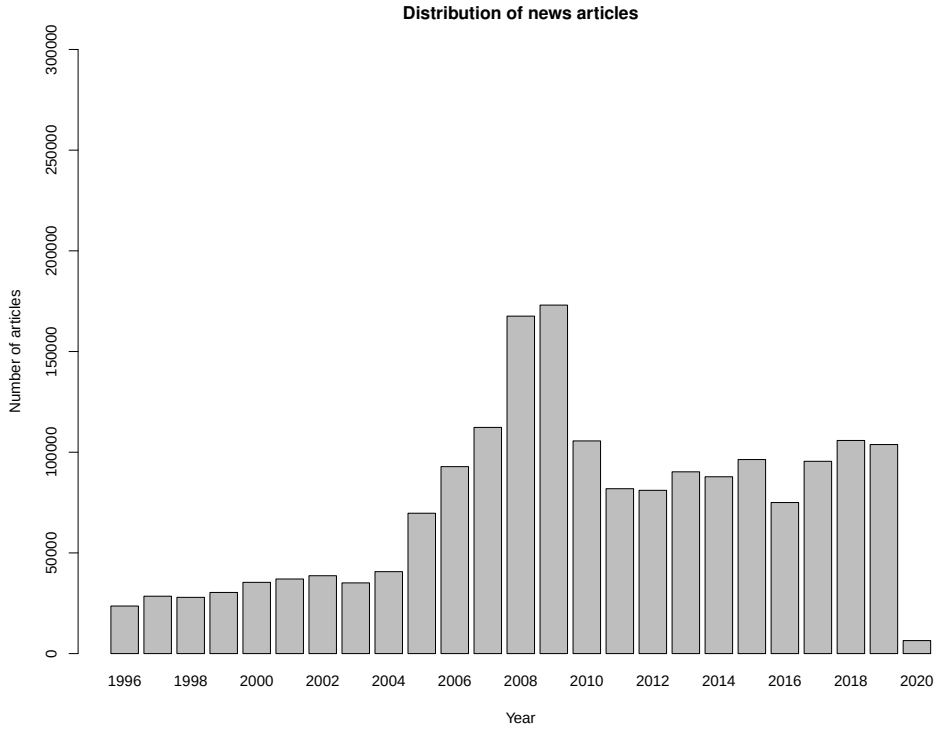


Figure 2.2: Number of annual news articles

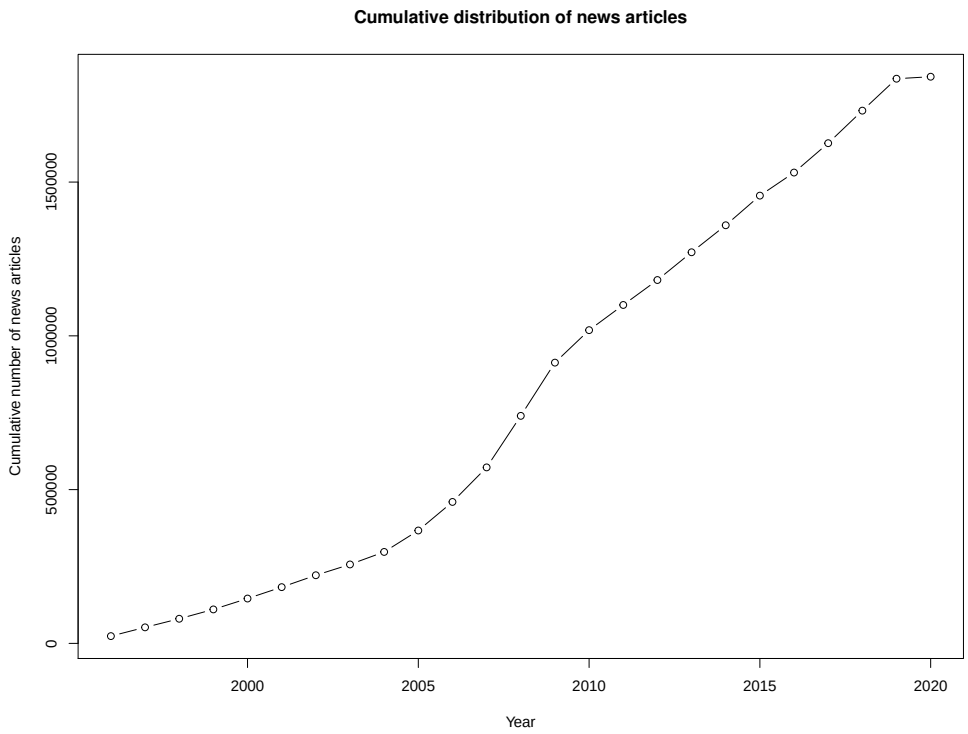


Figure 2.3: Cumulative number of annual news articles

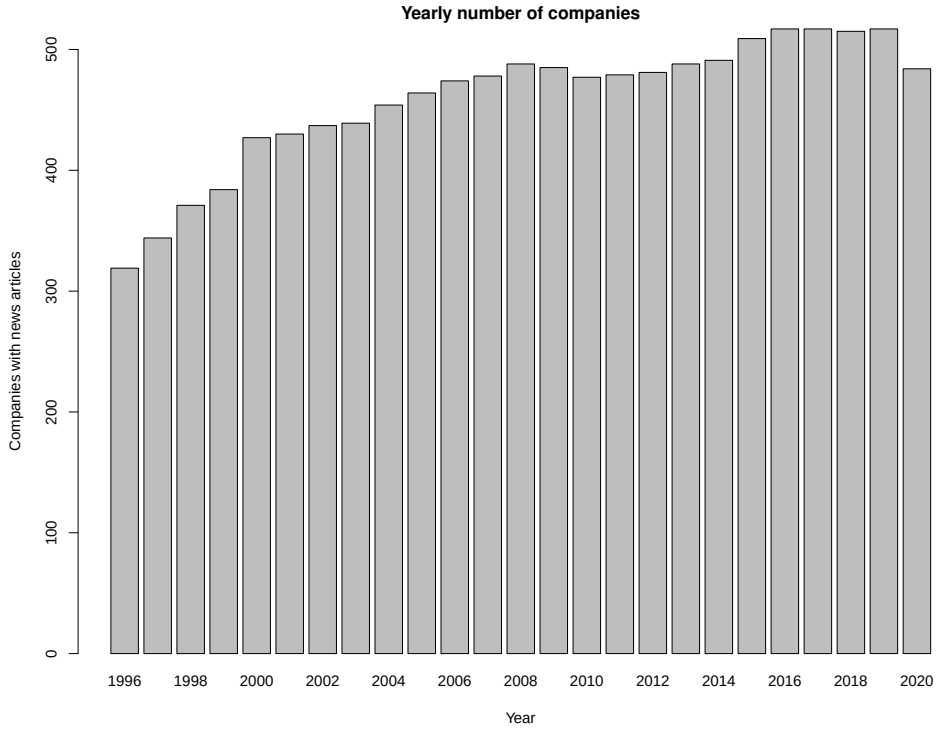


Figure 2.4: Annual number of companies with news

2.1.2 Deriving Labels from Return Data

The news dataset from Thomson Reuters does not contain any labels with sentiment information. However, labelled data is necessary for supervised machine learning. In this thesis, a similar approach as in Kelly, Ke, and Xiu (2019) is chosen, where labels with sentiment information are derived from return data.
Daily share prices for each of the 921 companies are downloaded for the entire period between January 1996 and January 2020 from Datastream. A small part of the data can be seen in Table 2.1.

Date	MLM	TSCO	ALB	QLGC	AVB	CPRT	AKS	NKTR	LEHMQ	ASND
1996-01-03	87.15	89.56	134.55	117.31	130.51	185.47	148.57	135.00	115.68	2040.00
1996-01-04	86.10	87.91	132.78	113.46	127.77	176.07	148.04	135.00	115.68	1846.67
1996-01-05	86.10	89.56	133.66	107.69	127.09	174.36	147.50	136.67	112.44	1873.33
1996-01-08	86.62	86.81	131.89	111.54	127.77	174.36	149.11	135.00	115.03	1893.33
1996-01-09	87.67	91.21	131.89	117.31	128.46	176.07	148.57	136.67	112.44	1720.00

Table 2.1: Price data of ten companies in 1996

Thereafter, daily returns of each company are calculated from the price data. A small excerpt of the return data table is shown in Table 2.2.
In order to derive the sentiment labels, daily returns are first normalized by the rolling window mean return (2.2). The window size was chosen to be 100 days. The normalized return r_{Tnorm} is than calculated with Formula (2.3) by subtracting the rolling window mean return from the daily returns.

Date	MLM	TSCO	ALB	QLGC	AVB	CPRT	AKS	NKTR	LEHMQ	ASND
1996-01-03	0.006119	0.018769	-0.006498	-0.016103	-0.020489	-0.018106	0.007254	-0.012219	0.052977	-0.057009
1996-01-04	-0.012048	-0.018423	-0.013155	-0.032819	-0.020995	-0.050682	-0.003567	0.000000	0.000000	-0.094770
1996-01-05	0.000000	0.018769	0.006628	-0.050855	-0.005322	-0.009712	-0.003648	0.012370	-0.028008	0.014437
1996-01-08	0.006039	-0.030706	-0.013243	0.035751	0.005351	0.000000	0.010915	-0.012219	0.023035	0.010676
1996-01-09	0.012122	0.050685	0.000000	0.051730	0.005400	0.009807	-0.003621	0.012370	-0.022516	-0.091548

Table 2.2: Daily returns of ten companies in 1996

$$r_{T100} = \frac{\sum_{t=T-1}^{T-101} r_t}{100} \tag{2.2}$$

$$r_{Tnorm} = r_T - r_{T100} \tag{2.3}$$

with:

r_T ... return at time step T
r_{T100} ... rolling window mean return evaluated at time step T

It is assumed that influences such as market trends, industry performance, key financial figures, etc. are already reflected in the price development and are thus eliminated after subtraction of the rolling mean return. Additionally, a second approach for return normalization is to subtract the market return on day t from the stock return on day t. Both strategies were implemented and a comparison showed that a normalization with market returns achieves better results.

The objective of both approaches is to eliminate non-news related influences and filter the impact of news on stock prices. Therefore, a barrier is defined that increases the probability that news articles have a reasonable influence on stock prices. This is discussed in more detail later in this Section. But there is another issue that needs to be considered first.
According to Kelly, Ke, and Xiu (2019) it takes two to four days until news is fully incorporated into stock prices. The authors distinguish between fresh and stale news. Stale news contain little new information related to recently published news. Fresh news in contrast, contain a lot of new information. They also point out that news from large companies are reflected more quickly in share prices, as compared to small companies, which receive less media attention. Since the S&P 500 is entirely composed of large companies, it is assumed that news are fully reflected in the share prices after a maximum of two days. Furthermore, it is necessary to determine which day most accurately reflects the sentiment information contained in a news article. Very negative or very positive news may cause the share price to fall or jump significantly on the same day. Furthermore, after reacting to new information, share prices tend to revert towards the initial price level over the days following. Consider for example a company that is the subject of negative news causing the share price to fall by 7%. The next day the share price recovers and rises by 3%. If the sentiment is derived from the return of the following day, the article would be labelled as positive although it came

along with a massive price drop on the same day. This strategy increases the likelihood that positive articles are labelled as negative an vice versa, which is undesirable.
Another possibility is to use the return of the day on which the articles were published. This strategy has the drawback that news published late in the evening may not be reflected in the return of that day.
To reduce misclassification, the arithmetic mean between the return on the publication date of the news and the subsequent day is calculated. This strategy increases the probability that news articles will be labelled as neutral. It is less harmful to classify sentiment charged news as neutral than to mistakenly classify them as negative or positive.

Furthermore, as already mentioned above, a barrier is defined in order to increase the likelihood that news articles are labelled correctly. If the absolute value of the normalized return exceeds the barrier, chances are high that news on that day contain information that caused a high return. In this case, the sentiment label is set positive with integer 2. On the other hand if the return is negative and its absolute value is above the barrier, the sentiment label is set to negative which is represented by integer 1. Otherwise if the return is close to zero, between the upper and lower barrier, news are considered as neutral and are labelled with integer 0. The barrier is set to 1% which results in a dataset that contains 118,567 positive, 326,231 neutral and 116,519 negative labelled news articles. Table 2.3 shows an extract of the derived sentiment labels. Finally, the news data is concatenated with the sentiment data. An excerpt of the resulting data frame is shown in Table 2.4.

	Date	MLM	PPL	ALB	QLGC	AVB	CPRT	AKS	PLD	NKTR
2	1996-01-03	0	0	1	1	1	1	0	1	0
3	1996-01-04	0	0	0	1	1	1	0	1	0
4	1996-01-05	0	0	0	1	0	0	0	1	0
5	1996-01-08	0	2	0	2	0	0	0	0	0
6	1996-01-09	0	2	1	2	0	0	0	0	0

Table 2.3: Sentiment labels derived of the daily return data which are normalized by the rolling window mean returns

	News	Sentiment
100	says board authorized approved amendments r...	1.0
101	says ceo howard nye fy compensation mln vs ...	0.0
102	cuts from	2.0
103	sees net earnings attributable mln mln re...	2.0
104	stifel cuts from says injured cited inc...	0.0
105	barclays raises from	0.0
106	result david maffucci resignation directors...	0.0
107	raises rating	0.0
108	sees expenditures mln mln reports results...	1.0
109	stifel raises from	0.0

Table 2.4: Excerpt of the labelled news dataset that is used for the supervised machine learning task

2.2 Text Mining

Text mining is "the discovery by computer of new, previously unknown information, by automatically extracting information from different written resources" (Hearst, 2009).
Since textual data is inherently high-dimensional, it is necessary to reduce the dimensionality to a lower level. Otherwise computation would be very expensive and the models would easily overfit (Gentzkow, Kelly, and Taddy, 2017).

Data cleaning is the first step to reduce the dimensionality of the data. Words that do not carry sentimental information, such as stop words and proper nouns, are removed, as well as HTML tags, punctuation marks and numbers. Furthermore, since all news articles of one day are combined into one document, it often occurs that identical text passages occur multiple times. Section 2.2.2 describes in more detail how duplicate content is removed. Additionally, feature extraction is another important step to reduce dimensionality and convert the data into a machine readable form. The different feature extraction approaches are described in more detail in Section 2.2.3.

2.2.1 Data Cleaning

The raw news data is cleaned in several steps. First, the text is transformed into lower case and all contractions are expanded (e.g. ain't → am not, can't → cannot, etc.). Next, all characters like numbers, special characters and punctuation marks get removed, only letters remain. Furthermore, also all single characters are deleted since they occur in high numbers and carry no sentiment information. The next step is to split the text into word tokens. As an example, the sentence 'apple computer revealed new iphone' becomes a list of tokens:

['apple', 'computer', 'revealed', 'new', 'iphone']

Stop words like 'and', 'how', 'is', 'the', etc. don't carry any sentiment information and are removed from the list of word tokens. Afterwards also some special strings, that appear very often in this dataset, get removed. Those special strings are:

('na', 'inc', 'ltd', 'us', 'nyse', 'com', 'thomson', 'reuters', 'last', 'bid', 'ask', 'moc', 'bln')

Like the stop words, those special strings also contain no sentiment information and thus get removed. In addition, after detailed analysis of the data, it was found that the phrase "nyse order imbalance shares sell (buy) side" occurs several hundred thousand times. Although this sentence might carry some sentiment information, it is removed to get rid of repeating information that might affect the learning task negatively.
Since the sentiment is calculated on a daily basis, all daily news of a specific company are aggregated into one document that is later used for sentiment classification. However, those daily news articles often contain very similar content which results in a lot of duplicate paragraphs and phrases. To solve this issue and to reduce the number of words in the documents, an algorithm is developed to delete duplicate phrases. The characteristics of this algorithm are described in Section 2.2.2 in more detail. In total, more than 10 million words that occurred in duplicate paragraphs were removed from the dataset. The final step is to get rid of all company names. If one company performs very well, the learning algorithm could associate the frequently occurring company name with positive sentiment information. This would end up in biased results. To avoid any influence of the company itself, the company name as well as the companies' ticker codes, are removed from the entire text corpus.

2.2.2 Removing Duplicate Content

In order to detect duplicate content in news documents, they must first be converted into a form a computer can process. Therefore, each word is encoded with the use of the Label Encoder function that is part of the python library *scikit-learn*. This function converts each word into an integer number. Consider the following paragraph that is extracted from a news article after the cleaning task and contains duplicate content (underlined). The encoder transforms this text into a sequence of integers.

'san francisco feb computer declined comment reports gil amelio former chairman nsm named replace chief executive michael spindler sources said ousted san francisco feb computer declined comment'

[18 9 7 4 5 3 16 10 0 8 1 13 12 15 2 6 11 20 19 17 14 18 9 7 4 5 3]

Thereafter, the function *duplicate_phrases* is defined which takes the integer encoded text as input. Furthermore, the parameter w determines the size of the sliding window which is used for duplicate content detection. In this thesis, the window size is set to 5. That means repeating patterns of five or more consecutive words will be removed from the article.
The window slides over the encoded text with an increment of one. After each increment the window patterns are appended to the 'seen' list. If the actual window pattern is already part of the 'seen' list, then the indices of the actual window are appended to the list of duplicates. After the window is slid over the entire text, the list of duplicates contains all indexes of those words that occur in recurring phrases. Those indices are then deleted from the encoded text. Finally, the label decoder transforms the numbers back to words and the result is a text without duplicate phrases.

```
def duplicate_phrases(encoded_text, w=5):
    seen = []
    duplicates = []
    for i in range(0, len(encoded_text)-w+1):
        window = encoded_text[i:(i+w)]
        for orig in seen:
            if np.array_equal(window, orig):
                for k in range(i,i+w):
                    duplicates.append(k)
                break
        else:
            seen.append(window)
    return list(set(duplicates))

duplicate_indizes = duplicate_phrases(text_num)
```

2.2.3 Feature Extraction

Feature extraction is a crucial task in machine learning in order to reduce the dimensionality of the data. With meaningful features, the training speed of the model will increase and the tendency to overfit will be reduced (DeepAI, 2019). According to Md Tayeen et al. (2019) three modern categories of word level feature extraction can be distinguished: frequency based, context based and hybrid approach.
Bag-of-words is a frequency-based approach that simply quantifies the frequency of all words occurring in a document. A further advancement of bag-of-words is the use of short phrases, called n-grams, as features. In addition, a more sophisticated weighting scheme called tf-idf (Term Frequency Inverse Document Frequency) uses weighted n-grams to reduce the effects of noise caused by very common words. Those approaches are described in more details in the Sections 2.2.3.1 to 2.2.3.4.
While the information about word order and context is completely lost with frequency-based methods, this drawback is not present within context-based approaches. Thereby an underlying neural network captures the context of words and transforms each word into a vector representation. Words with similar meanings receive vectors that point in similar directions. This concept is described in more detail in Section 2.2.3.5.
The third category is a hybrid between the frequency-based and the context-based approach. This uses frequency-based methods to weight the feature vectors resulting from the context-based approach. However, this is not further discussed in this thesis.

2.2.3.1 Bag-of-Words Model

The bag-of-words representation is a very simplified representation of textual data. This approach represents text as a list of its word counts, without regard to word order and grammar. Let c_i be a vector with a length that is equal to the number of words in the vocabulary. The elements c_{ij} of the vector represent the count of occurrences of word j in document i (Gentzkow, Kelly, and Taddy, 2017). Despite its simplicity, this approach can lead to good results, as shown by Rechenthin, Street, and Srinivasan (2013) who were able to predict price movements on the following trading day with an accuracy of 62.86%.

As an example, consider the following text corpus consisting of three documents:

Document 1: "The Apple stock is the best performing stock this week"
Document 2: "The Fed increases interest rates this week"
Document 3: "Morgan Stanley increases the price target of Apple"

After the cleaning procedure the text corpus becomes:

Document 1: "stock best perform stock week"
Document 2: "fed increase interest rate week"
Document 3: "morgan stanley increase price target"

To turn these sentences into a bag-of-word representations, a vocabulary list must first be created. The list of vocabulary is:

['stock' 'best' 'perform' 'week' 'fed' 'increase' 'interest' 'rate' 'morgan' stanley' 'price' 'target']

As a result each sentence is represented by a vector with a length equal to the number of words in the vocabulary. All entries of those vectors are zero, despite those that represent the words that actually appear in each document. Table 2.5 shows the bag-of-words representation of the three documents. For this tiny list of vocabulary, the vectors are quite small. However, if there is a broad vocabulary, this leads to very large sparse vectors.

	stock	best	perform	week	fed	increase	interest	rate	morgan	stanley	price	target
Document 1	2	1	1	1	0	0	0	0	0	0	0	0
Document 2	0	0	0	1	1	1	1	1	0	0	0	0
Document 3	0	0	0	0	0	1	0	0	1	1	1	1

Table 2.5: Example of a bag-of-words representation

2.2.3.2 Count Vectorizer

The transformation to a bag-of-words representation is done in python with the function CountVectorizer() of the library *scikit learn* (Pedregosa et al., 2011).
It takes the following lines of code:

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CoutVectorizer(max_features=25000, ngram_range=(1,1))
vectorizer.fit(x_train)
x_train_vectorized = vectorizer.transform(x_train)
```

Taking the previous three documents as input for x_train. The output is the same as above.

	best	fed	increase	interest	morgan	perform	price	rate	stanley	stock	target	week
Document 1:	1	0	0	0	0	1	0	0	0	2	0	1
Document 2:	0	1	1	1	0	0	0	1	0	0	0	1
Document 3:	0	0	1	0	1	0	1	0	1	0	1	0

Table 2.6: Count vectorized representation of the text corpus in Section 2.2.3.1

The parameter *max_features* limits the maximum length of the input feature vector. After data cleaning, the data set counts a total vocabulary of 163,477 words. A higher number of input features increases the risk of overfitting, so the number of features is limited to 25,000. With this restriction, the count vectorizer uses the 25,000 most frequent words while the remaining words are not considered.

Table 2.7 shows the ten most frequent unigram features of the cleaned training dataset together with their word frequencies. The frequencies are only shown for the classes positive and negative for reasons of simplicity and due to the different observation count of the neutral class in contrast to the positive and negative classes (the neutral class contains 326,231 observations while the positive class contains 118,567 and the negative class contains 116,519 observations).
Additionally, Figure 2.5 plots the word frequencies of negative labelled articles on the horizontal axis and word frequencies of positive labelled articles on the vertical axis for the

10,000 most frequent features (excluding the ten most frequent features for a better scaling). It can be observed that all features appear almost equally often in positive as well as negative classes. To quantify the result, a measure is defined in (2.4) which determines the deviation of the data points between the positive and negative class.

$$\text{deviation} = \sum_{i=1}^{10000} \frac{|pos_i - neg_i|}{\left(\frac{pos_i + neg_i}{2}\right)} = 1830.71 \tag{2.4}$$

	negative	positive	total
said	152962	150480	303442
from	95374	93840	189214
percent	80771	82558	163329
shares	61505	61608	123113
year	53794	54413	108207
million	52319	53238	105557
quarter	47956	47147	95103
billion	45238	46683	91921
share	43250	46103	89353
not	40354	37498	77852

Table 2.7: The ten most frequent bag-of-words unigram features derived from the count vectorizer for positive and negative classes

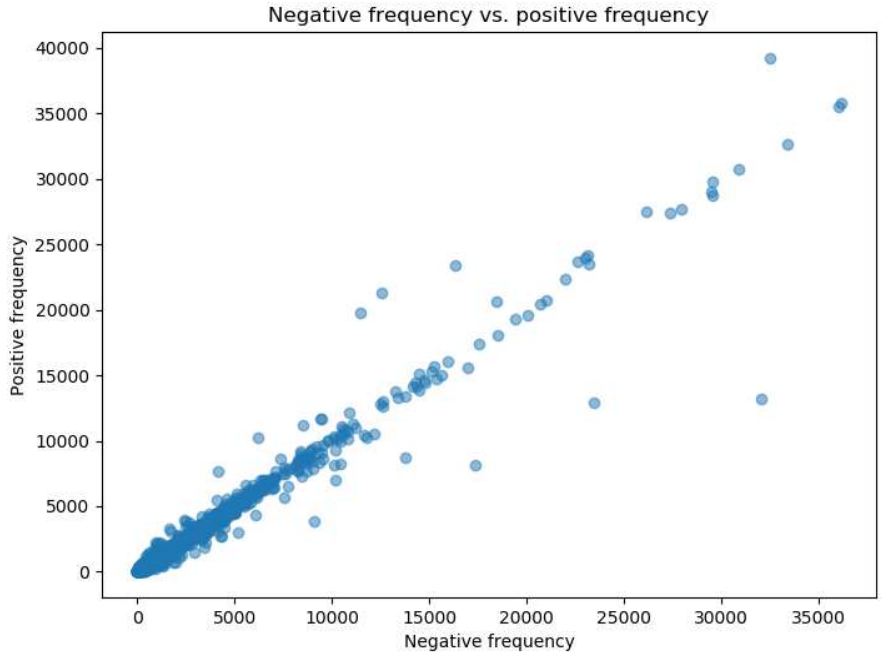


Figure 2.5: Frequency of positive labelled features vs. negative labelled features of a bag-of-words model with unigram features

2.2.3.3 N-Grams

To obtain a richer representation of features, the bag-of-words model can be extended to count word phrases instead of individual words. A phrase of length n is denoted as n-gram. N-grams with a length of one are denoted as unigrams, n-grams with a length of two are known as bigrams and n-grams with a length of three are called trigrams. One drawback of using n-grams is that the dimension of c_i increases exponentially with the order n. For this reason, n-grams with a maximum length of three are used in practice (Gentzkow, Kelly, and Taddy, 2017). The ContVectorizer takes the argument *ngram_range* to determine the n-gram sizes. The following line of code extracts bigram features from the documents.

```
vectorizer = CoutVectorizer(max_features=25000, ngram_range=(2,2))
```

With bigrams, the features from the previous example are:

['stock best' 'best perform' 'perform stock' 'stock week' 'fed increase' 'increase interest' 'interest rate' 'rate week' 'morgan stanley' ' stanley increase' 'increase price' 'price target']

	best perform	fed increase	increase interest	increase price	interest rate	morgan stanley	perform stock	price target	rate week	stanley increase	stock best	stock week
Document 1:	1	0	0	0	0	0	1	0	0	0	1	1
Document 2:	0	1	1	0	1	0	0	0	1	0	0	0
Document 3:	0	0	0	1	0	1	0	1	0	1	0	0

Table 2.8: Count vectorized representation of the text corpus in Section 2.2.3.1 with bigrams as features

Table 2.9 shows the ten most frequent bigram features of the cleaned training dataset. Additionally, Figure 2.6 gives a better insight to the distribution of all features. It can be observed that the plot is more scattered than the one with unigram features. But still, the majority of features occur at low frequencies. Therefore, a second plot with a different scaling is shown in Figure 2.7. The deviation (2.5) increased by 11.31% to 2037.73 in contrast to unigram features. Due to this richer feature representation, higher classification accuracy can be expected.

$$\text{deviation} = \sum_{i=1}^{10000} \frac{|pos_i - neg_i|}{\left(\frac{pos_i + neg_i}{2}\right)} = 2037.73 \tag{2.5}$$

	negative	positive	total
per share	20741	21851	42592
imbalance shrs	15665	22767	38432
stock exchange	10287	10233	20520
chief executive	9671	10139	19810
shrs imbalance	7531	11758	19289
fourth quarter	9410	9035	18445
rose percent	6698	11384	18082
cents per	8954	9106	18060
from rating	8586	8855	17441
second quarter	8285	7993	16278

Table 2.9: Top ten most frequent bag-of-words bigram features of the count vectorizer

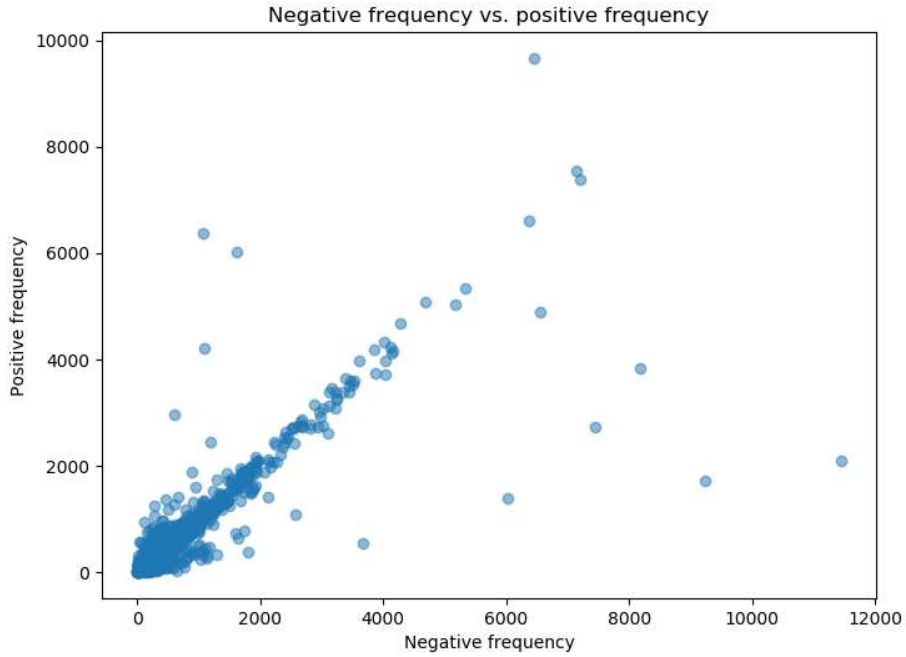


Figure 2.6: Frequency of positive labelled features vs. negative labelled features of a bag-of-words model with bigram features

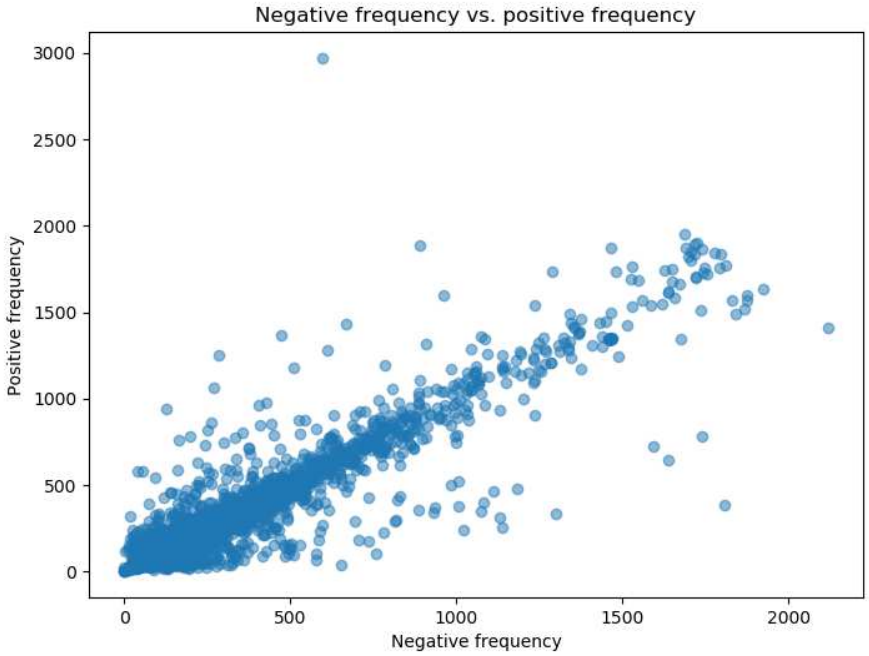


Figure 2.7: Frequency of positive labelled features vs. negative labelled features of a bag-of-words model with bigram features at a different scaling

2.2.3.4 Tf-idf Vectorizer

Another feature extracting method often used in natural language processing is called term frequency-inverse document frequency (tf-idf). The tf-idf value increases proportional to the frequency a word i occurs in document j and decreases with the overall number of documents containing the word i.
Basically, this calculation determines how relevant a word is in a distinct document. Words that appear only in a small number of documents receive higher tf-idf values than words that appear frequently in the entire corpus of documents (Ramos, 2003).
The tf-idf value is calculated as follows:

$$tf\text{-}idf_{i,j} = tf_{i,j} * idf_i \tag{2.6}$$

with:

$tf_{i,j}$ ... frequency of the word i in document j
idf_i ... inverse document frequency of word i

The first term $tf_{i,j}$ is the frequency of the word i in document j. The second term idf_i is used to adjust the weight of a word relative to it's overall frequency in the text corpus. It is calculated with Equation 2.7 (Pedregosa et al., 2011).

$$idf_i = ln\left(\frac{N+1}{df_i+1}\right) + 1 \tag{2.7}$$

with:

df_i ... number of documents containing word i
N ... total number of documents

To implement this in python, the function TfidfVectorizer() of the library *scikit learn* is used.

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max_features=25000, ngram_range=(1,1))
vectorizer.fit(x_train)
x_train_vectorized = vectorizer.transform(x_train)
```

The tf-idf representation of the example in Section 2.2.3.1, consisting of three documents is shown in Table 2.10.

	best	fed	increase	interest	morgan	perform	price	rate	stanley	stock	target	week
Document 1:	0.39	0.00	0.000	0.00	0.000	0.39	0.000	0.00	0.000	0.78	0.000	0.297
Document 2:	0.00	0.49	0.373	0.49	0.000	0.00	0.000	0.49	0.000	0.00	0.000	0.373
Document 3:	0.00	0.00	0.355	0.00	0.467	0.00	0.467	0.00	0.467	0.00	0.467	0.000

Table 2.10: Tf-idf representation of the text corpus in Section 2.2.3.1 with unigram features

A more detailed explanation of the calculation makes the outcome more comprehensible. As an example, consider the calculation of the tf-idf value of the word 'stock' in document one. The word 'stock' appears two times in the first document. Therefore, the term $tf_{stock,1}$ equals 2. Furthermore, the word 'stock' only appears in one document, thus df_{stock} equals 1 and the number of documents N equals 3. Now, idf_{stock} can be calculated as:

$$idf_{stock} = ln\left(\frac{3+1}{1+1}\right) + 1 = ln\left(\frac{4}{2}\right) + 1 = ln(2) + 1 = 1.693 \tag{2.8}$$

Multiplying the two terms results in:

$$tf\text{-}idf_{stock,1} = 2 * 1.693 = 3.386 \tag{2.9}$$

To get the same value as above, the resulting tf-idf value is normalized by the Euclidean norm: (Pedregosa et al., 2011)

$$v_{norm} = \frac{v}{\|v\|_2} = \frac{v}{\sqrt{v_1^2 + v_2^2 + ... + v_n^2}} \tag{2.10}$$

The $idf_{i,1}$ values of the remaining words in document one are:

$idf_{best,1} = 1.693$
$idf_{perform,1} = 1.693$
$idf_{week,1} = 1.288$

Now, the normalized value of $tf\text{-}idf_{stock,1}$ can be calculated:

$$tf\text{-}idf_{stock,1,(norm)} = \frac{3.386}{\sqrt{3.386^2 + 1.693^2 + 1.693^2 + 1.288^2}} = 0.78 \tag{2.11}$$

Table 2.11 shows the top ten unigrams of the tf-idf vectorizer whereas Table 2.12 shows the top ten features with bigram features. Additionally, a plot with bigram features of the tf-idf vectorizer is shown in Figure 2.8.
The tf-idf vectorizer further improved the feature extraction for unigrams as well as for bigrams in contrast to the count vectorizer. The deviation between the data points of the positive and the negative class is 1935.62 for unigrams and 2126.87 for bigrams, according to Equation (2.4).

	negative	positive	total
imbalance	12903.75	13938.12	26841.87
shrs	5682.91	8375.39	14058.29
from	4280.30	4348.44	8628.74
said	3129.29	3176.32	6305.61
rating	2757.03	2818.19	5575.22
shares	2607.96	2669.37	5277.33
raises	2107.88	3114.21	5222.10
cuts	3191.56	2026.00	5217.57
says	2180.23	2229.91	4410.14
pct	2077.75	2132.60	4210.34

Table 2.11: Top ten most frequent bag-of-words unigram features of the tf-idf vectorizer

	negative	positive	total
imbalance shrs	7183.42	10550.90	17734.32
shrs imbalance	4154.16	6634.85	10789.01
raises from	1639.00	2112.57	3751.58
from rating	1851.38	1873.52	3724.90
cuts from	1601.50	1455.21	3056.72
per share	1147.54	1252.61	2400.15
down pct	1602.75	249.06	1851.81
shares down	1511.12	212.66	1723.79
shares pct	163.11	1264.13	1427.24
shr view	713.13	706.72	1419.86

Table 2.12: Top ten most frequent bag-of-words bigram features of the tf-idf vectorizer

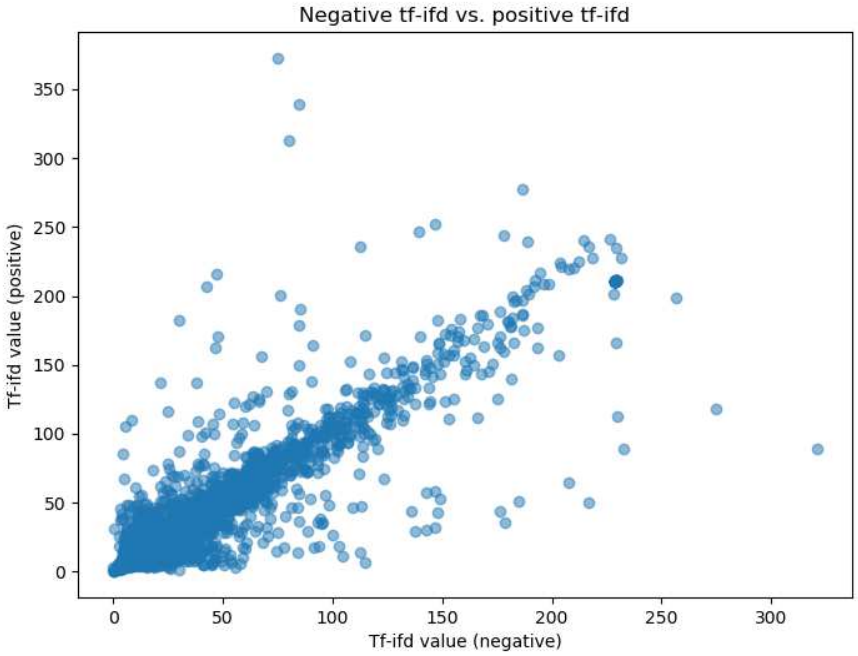


Figure 2.8: Tf-idf values of positive labelled features vs. negative labelled features of a bag-of-words model with bigram features

In this thesis the performance of different n-grams is evaluated with a feedforward neural network that has two fully connected hidden layers with 256 nodes each. All fully connected layers are followed by a dropout layer with a dropout-rate of 0.2. For more details, see Section 2.4.3. Furthermore, the tf-idf vectorizer is chosen for feature extraction and the network is trained over 10 epochs. The resulting training losses are shown in Figure 2.9.
A combination of unigrams and bigrams reduces the training loss in contrast to unigrams. With trigrams the performance does not further increase. Presumable this is because the input size is limited to the 25,000 features with the highest tf-idf scores. In contrast, the total vocabulary contains 163,477 unigrams, the combined number of unigrams and bigrams is 6,317,073 and with trigrams included, this number rises to a total of 22,023,067 features. The number of input features is quite small compared to the total number of available features. With a larger number of input features trigrams may show further improvements. Nevertheless, experiments have shown that a higher number of input features increases the tendency of overfitting, which is not beneficial.

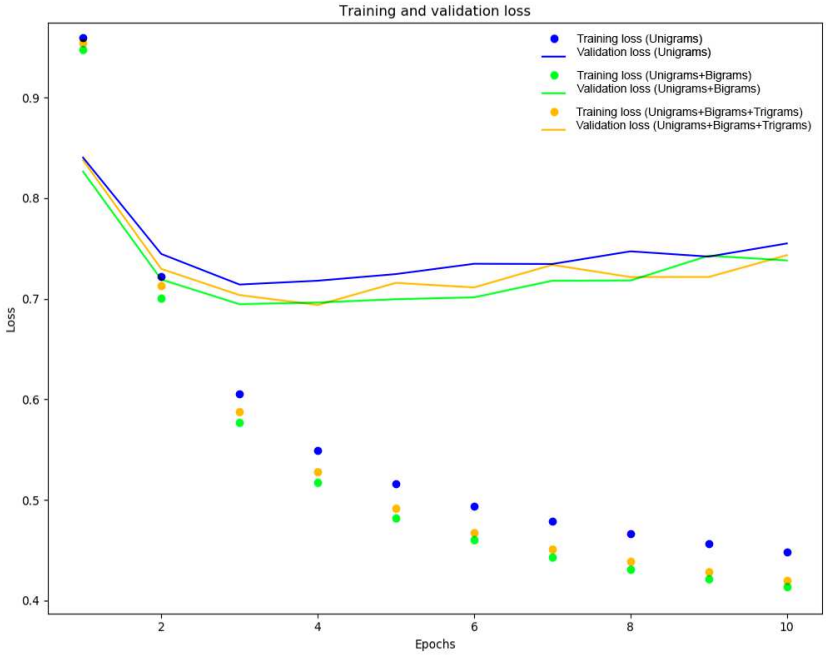


Figure 2.9: Training and validation loss of a neural network trained with different n-gram features.

2.2.3.5 Word Embeddings

Word embeddings are a sophisticated, context-based feature extraction approach that uses a distributional similarity based representation. Previous to that, the vast majority of NLP tasks was done with discrete, rule based or statistical approaches. However, these concepts have two major disadvantages (Manning and Socher, 2017).

1. Discrete representations do not have any inherent notion of similarity.

Consider two one-hot encoded words:

motel: [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 ..... 0]
hotel: [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 ..... 0]

Problem: The document vectors are orthogonal to each other, thus they contain no information about relationships between words.

2. When the text corpus and thus the number of vocabulary is large, the result are big spares vectors that are inefficient to compute (Tomar, 2019).

Word embeddings overcome those issues since it aims to map semantic meaning of words into a geometric space. Each word of a text corpus is therefore associated by a dense numerical vector.

$$\text{increase} = \begin{bmatrix} 0.2231 \\ 0.0043 \\ 0.4502 \\ \dots \\ 0.0421 \end{bmatrix}$$

The cosine distance between two vectors captures part of the semantic relationship between two related words. The geometric space created by these vectors is known as the embedding space (Chollet, 2016).
Consider Figure 2.10, which shows a simplified vector representation in a 2-dimensional embedding space. It can be observed that related words have similar vector representations, e.g. a low cosine distance. In practice, the embedding space is much larger. It ranges between 50 and 1000.

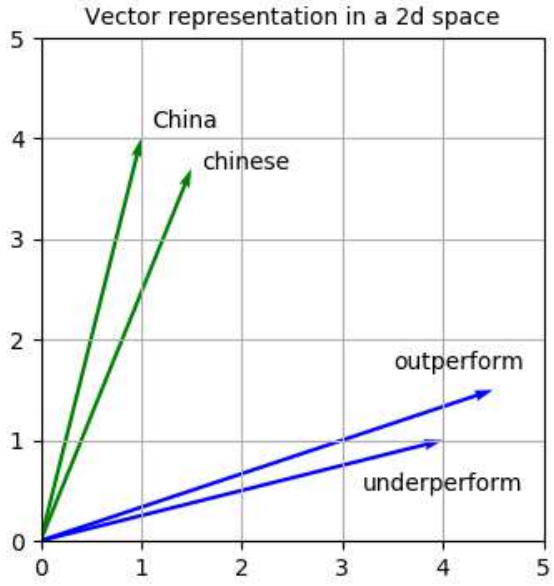


Figure 2.10: Word embeddings in a 2-dimensional embedding space

For this thesis, the popular word embedding approach word2vec is used. This approach was developed by Mikolov et al. (2013) at Google. The word2vec model is a shallow neural network with one hidden layer. In order to learn the representation of a word, the algorithm iterates over the entire text corpus. At each position t, a center word c is defined as well as it's context words o. The number of context words is determined by the window size m (Tomar, 2019). Mikolov et al. (2013) achieved the best performance with a window size of eight, considering four future and four history words.

Within word2vec, two model architectures can be distinguished which are called CBOW (continuous bag-of-words) and skip-gram. The difference is that CBOW uses context words to predict the current word, while skip-gram uses the current word to predict the context words. Both model architectures are shown in Figure 2.11.

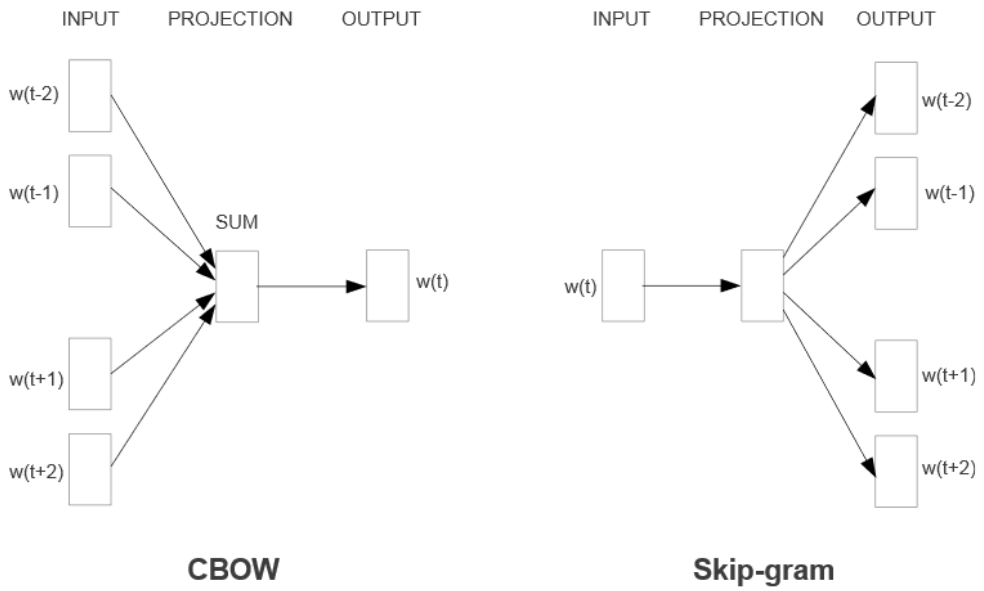


Figure 2.11: CBOW and skip-gram model architectures (Mikolov et al., 2013)

Consider the following example, which for simplicity uses only two future and two history words:

"... Donald Trump is the president of the United States of America ..."

After data cleaning the sentence is:

"... donald trump president united states america ..."

The CBOW architecture takes the surrounding words 'donald', 'trump', 'united' and 'states' as the inputs and 'president' as the desired output.

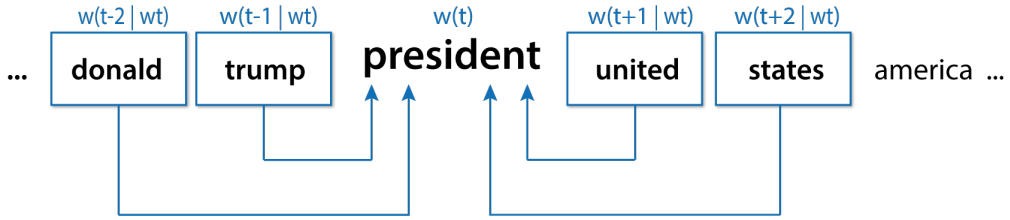


Figure 2.12: Example of a CBOW prediction

The skip-gram architecture takes the center word 'president' and aims to predict the surrounding context words.

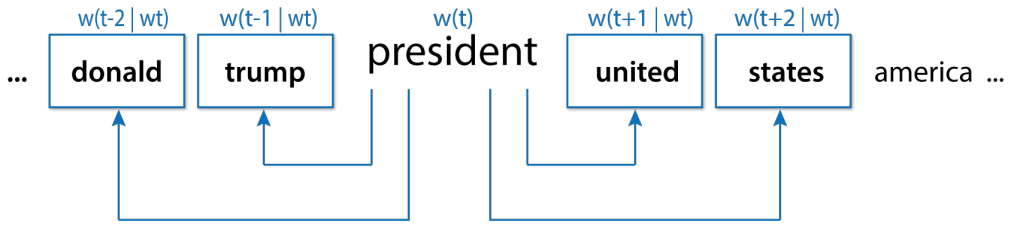


Figure 2.13: Example of a skip-gram prediction

The algorithm iterates over all positions t = 1 ... T of a large text corpus and aims to predict the surrounding words of a center word w_t in a window of size m. It keeps adjusting the vector representations of words in order to minimize the loss (Manning and Socher, 2017).

Loss function J: Maximize the probability of any context word given the current center word c (or w_t):

$$J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j}|w_t;\theta) \tag{2.12}$$

θ is a vector of parameters that are optimized within the model. These parameters are the vector representations of the words. For simpler calculus, Equation 2.12 is converted to log-probabilities.

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} logP(w_{t+j}|w_t) \tag{2.13}$$

The inner product is a measure for similarity between vectors. If two vectors are similar, the inner product becomes larger.

$$dot\_product = u^T v = u * v = \sum_{i=1}^{n} u_i v_i \tag{2.14}$$

The softmax function in 2.15 is used to transform numbers, in this case the calculated inner product, into a distribution:

$$P_i = \frac{e^{u_i}}{\sum_j e^{u_j}} \tag{2.15}$$

During iteration through all words, the similarity between the context words u_o and the center word v_c is calculated. The result is the probability that a context word o appears with a given center word c:

$$P(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w=1}^{v} exp(u_w^T v_c)} \tag{2.16}$$

with:

v_c ... center vector of the center word
u_o ... outside vector associated with context word in index o
v ... number of words

The models is trained with gradient descent. The gradient of Equation 2.16 results in:

$$\frac{\partial}{\partial v_c} log \frac{exp(u_o^T v_c)}{\sum_{w=1}^{v} exp(u_w^T v_c)} = u_o - \sum_{x=1}^{v} P(x|c) u_x \tag{2.17}$$

with:

u_x ... expectation vector

$\sum_{x=1}^{v} P(x|c)$ ... probability for a word appearing in the context

In every iteration parameters are updated with gradient descent, where α is the learning rate:

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta) \tag{2.18}$$

For the implementation of word2vec in python, the package *gensim* is used. The model is trained on a text corpus consisting of the training and validation set.

	'rise'	Cosine distance
0	jump	0.897825
1	fall	0.857342
2	climb	0.852943
3	drop	0.819427
4	surge	0.777524
5	soar	0.754075
6	slip	0.739273
7	dip	0.735898
8	decline	0.697211
9	plunge	0.682116

Table 2.13: Most similar words of the center word 'rise', trained with CBOW

It can be observed that word2vec produces meaningful results when trained on the Reuters news dataset. The nearest neighbours of the target word "rise" are all semantically similar. However, this approach does not capture the sentiment polarity of the words. The words "fall", "drop", "dip", "decline", "slip" and "slide" have all opposite polarities in relation to the target word. Tang et al. (2016) has shown, that the performance of sentiment classification can be improved, when word embeddings take sentiment polarity into account.
Therefore, the word embeddings are refined in this thesis to capture both, semantic similarity and sentiment polarity. This is achieved with the neural network, described in Section 2.4.3.2. The first layer of the network is the Keras embedding layer, which is initialized with word embeddings, derived from word2vec. The parameter *trainable* of the embedding layer is set to True for this task. This refines the word embeddings as it is trained on the labelled training data set for 20 epochs. Finally, Table 2.14 shows the cosine similarities after the

refinement process. It can be observed that the refinement was successful since the derived embeddings all have the same polarity.

	word2vec		Refined word embeddings	
	'rise'	Cosine distance	'rise'	Cosine distance
0	jump	0.897825	jump	0.825556
1	fall	0.857342	climb	0.782654
2	climb	0.852943	surge	0.748178
3	drop	0.819427	soar	0.653012
4	surge	0.777524	jumped	0.634171
5	soar	0.754075	climbed	0.617613
6	slip	0.739273	rose	0.61282
7	dip	0.735898	surged	0.596018
8	decline	0.697211	rises	0.583132
9	plunge	0.682116	swell	0.554758
10	rises	0.680514	soared	0.548713
11	slide	0.674310	jumps	0.540866
12	tumble	0.670736	gained	0.515594
13	rose	0.645209	rally	0.51484
14	plummet	0.630832	increase	0.505893
15	jumped	0.625305	beat	0.503931
16	increase	0.620173	grew	0.501561
17	surged	0.608561	outpace	0.495921
18	climbed	0.601571	surging	0.486243
19	rising	0.598488	up	0.486083

Table 2.14: Most similar words of the center word 'rise' after the refinement process

2.3 Data Visualization

The visualizations in the Sections 2.2.3.2 to 2.2.3.4 give only a limited insight into the data, since the majority of the data points are located very close to the lower left corner of the diagrams. In this section several metrics are calculated to find sentimental words and get a better idea of the data. As in the previous sections, only the classes positive and negative are considered for the data visualization tasks described below.
The first metric that is calculated is the positive rate. It is the ratio of the word frequencies in the positive classes to the overall frequencies of the words. With this number one gets a better sense for the word polarity. It is calculated with Formula 2.19 (Kim, 2017).

$$pos\_rate_i = \frac{pos\_frequency_i}{pos\_frequency_i + neg\_frequency_i} \tag{2.19}$$

Another metric is the positive frequency. It is calculated by dividing the positive frequency of one word by the sum of all positive word frequencies (see Equation (2.20)). At this point, this measure does not provide much new information as it is only the pos_frequency scaled

by a factor. But combining the positive rate with the positive frequency will end up in a more meaningful metric.

$$positive\_freq_i = \frac{pos\_frequency_i}{\sum_{i=1}^{n} pos\_frequency_i} \tag{2.20}$$

with:

n ... total number of words in the vocabulary

The values of the pos_rate and the positive_freq are very different in size which can be observed in Table 2.15. While the pos_rate ranges from 0 to 1, the pos_freq is squeezed between 0 and 0.023. Therefore, the arithmetic mean is not suitable to combine both values because the pos_rate would be too dominant. Instead, the harmonic mean is used (Kim, 2017). The harmonic mean has the tendency to enhance the influence of small outliers while minimizing the influence of large outliers (DeepAI, 2020). It is calculated with Equation (2.21).

$$pos\_hmean_i = \frac{2}{\frac{1}{pos\_rate_i} + \frac{1}{pos\_freq_i}} \tag{2.21}$$

	negative	positive	total	pos_rate	pos_freq	pos_hmean
shrs imbalance	2579.27	4120.18	6699.45	0.615003	0.008608	0.016979
from	3372.28	3422.78	6795.06	0.503716	0.007151	0.014102
said	2943.44	2984.08	5927.52	0.503428	0.006235	0.012317
raises	1562.09	2314.34	3876.42	0.597029	0.004835	0.009593
shares	2159.33	2224.72	4384.05	0.507459	0.004648	0.009212
rating	2083.94	2128.26	4212.21	0.505261	0.004447	0.008816
says	2018.78	2053.15	4071.93	0.504220	0.004290	0.008507
percent	1799.33	1912.78	3712.12	0.515281	0.003996	0.007931
pct	1612.16	1673.00	3285.15	0.509260	0.003495	0.006943
raises from	1174.24	1504.17	2678.40	0.561591	0.003143	0.006250

Table 2.15: Positive rate, positive frequency and harmonic mean of the ten most frequent bag-of-words unigram and bigram features of the tf-idf vectorizer

Next, the cumulative distribution function (CDF) for both, the pos_rate and the pos_freq is calculated. The distributions of the pos_rate and the pos_freq are shown in Figure 2.14 and 2.15. It can be observed that the distribution of the pos_rate is quite symmetric with a median of 0.5040, whereas the distribution of the pos_freq is skewed to the left with a median of 0.000037971. Table 2.16 shows the CDF values for the positive rate which is denoted as pos_rate_cdf and for the positive frequency, denoted as pos_freq_cdf. Furthermore, all previously described steps are also conducted for the negative classes which results in the CDF values for the negative rate, denoted as neg_rate_cdf and the negative frequency,

denoted as neg_freq_cdf. In a final step, the harmonic mean values of the positive class, denoted pos_hmean_cdf and for the negative class, denoted neg_hmean_cdf, are calculated by the Equation (2.22).

$$pos\_hmean\_cdf_i = \frac{2}{\frac{1}{pos\_rate\_cdf_i} + \frac{1}{pos\_freq\_cdf_i}} \tag{2.22}$$

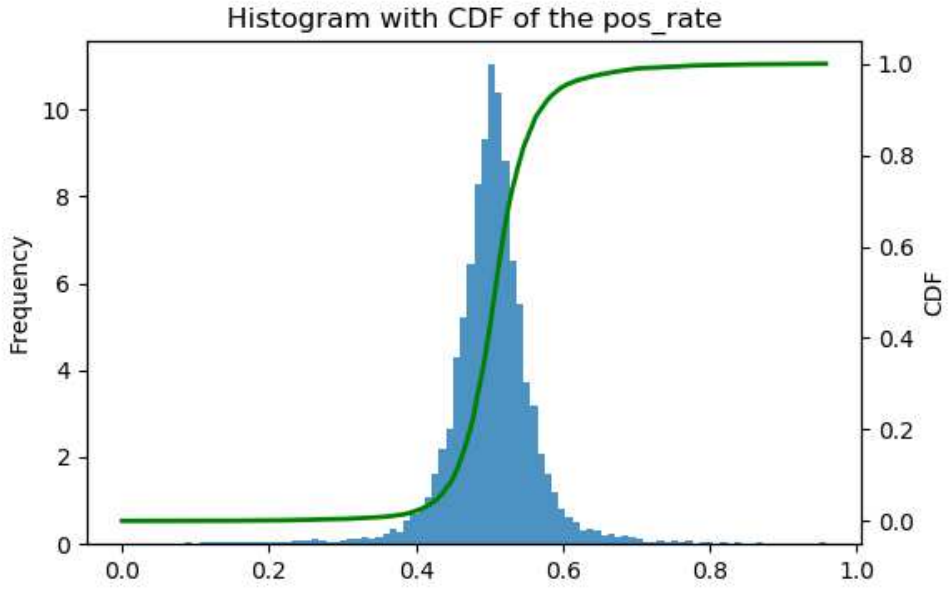


Figure 2.14: Histogram with the cumulative distribution function (CDF) of the pos_rate

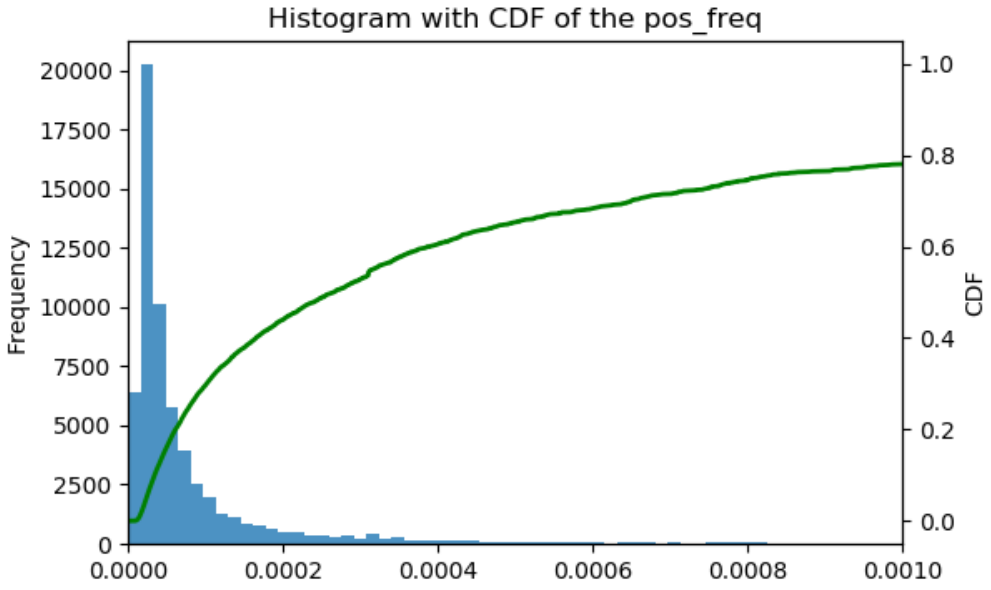


Figure 2.15: Histogram with the cumulative distribution function (CDF) of the pos_freq

	negative	positive	total	pos_rate_cdf	pos_freq_cdf	neg_rate_cdf	neg_freq_cdf	pos_hmean_cdf	neg_hmean_cdf
raises	1562.09	2314.34	3876.42	0.946852	0.926364	0.027986	0.911886	0.936496	0.054304
shares pct	108.56	880.35	988.91	0.999254	0.844261	0.000072	0.479380	0.915242	0.000143
rose	394.75	691.57	1086.32	0.970877	0.825010	0.012653	0.769306	0.892020	0.024897
raises from	1174.24	1504.17	2678.40	0.878417	0.897506	0.078855	0.882508	0.887859	0.144774
rise	305.98	553.99	859.97	0.974186	0.805436	0.010769	0.707440	0.881810	0.021214
raised	209.26	472.59	681.85	0.987706	0.779752	0.003918	0.633490	0.871495	0.007788
ups	227.35	459.08	686.43	0.982023	0.775844	0.006622	0.648556	0.866842	0.013111
from neutral	202.62	447.06	649.68	0.986881	0.771093	0.004291	0.629083	0.865743	0.008525
high	312.83	455.72	768.55	0.941882	0.773931	0.031407	0.712094	0.849687	0.060160
energy imbalance	441.65	578.60	1020.25	0.893113	0.808973	0.067368	0.783800	0.848963	0.124072

Table 2.16: CDF of the positive and negative rate, the positive and negative frequency as well as the positive and negative harmonic mean of the ten most frequent bag-of-words unigram and bigram features of the tf-idf vectorizer

Figure 2.16 shows the pos_cfd_hmean on the vertical axis and the neg_cfd_hmean on the horizontal axis. A segregation between positive and negative tokens can be observed. Positive tokens gather in the upper left corner, while negative tokens gather in the lower right corner. Words with no sentiment polarity can be found in the center of this plot.

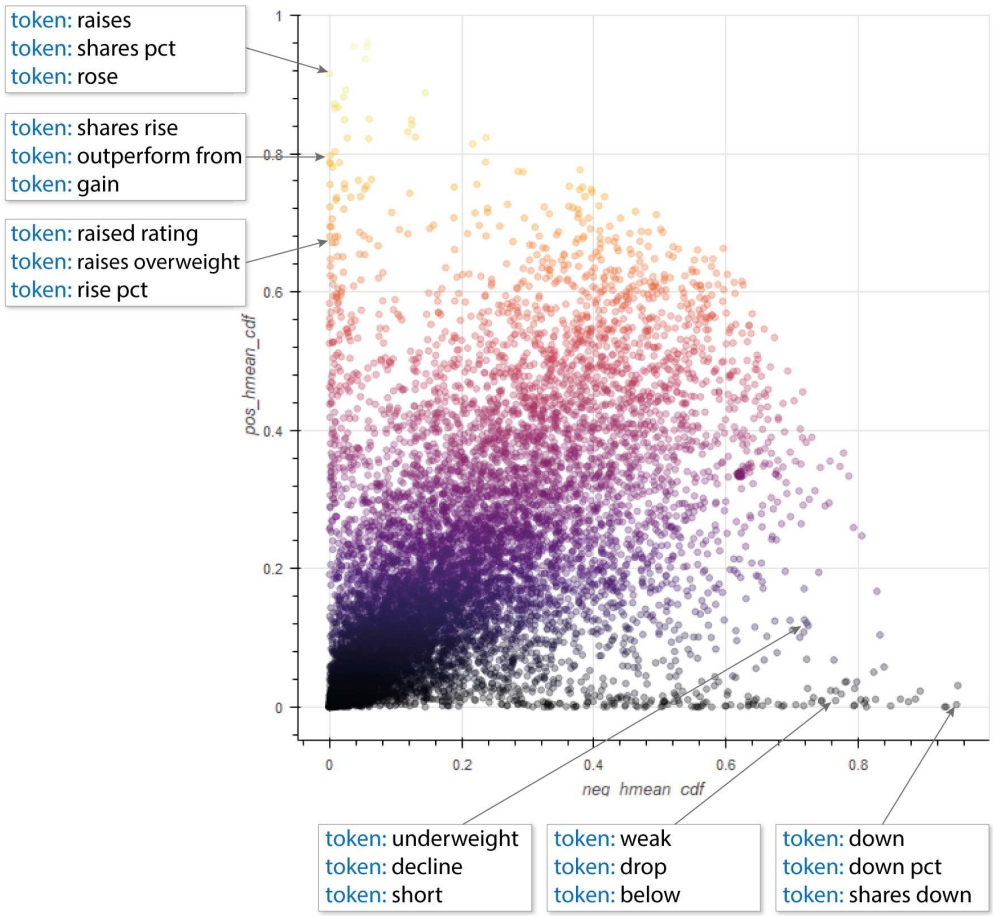


Figure 2.16: Segregation of positive charged and negative charged word tokens

Furthermore, word clouds are created to get an idea about the most negative and positive word tokens. The word cloud in Figure 2.17 shows the top 500 positive word tokens while Figure 2.18 shows the top 500 negative word tokens. It can be observed that word tokens of positive and negative sentiment are well detected and separated.

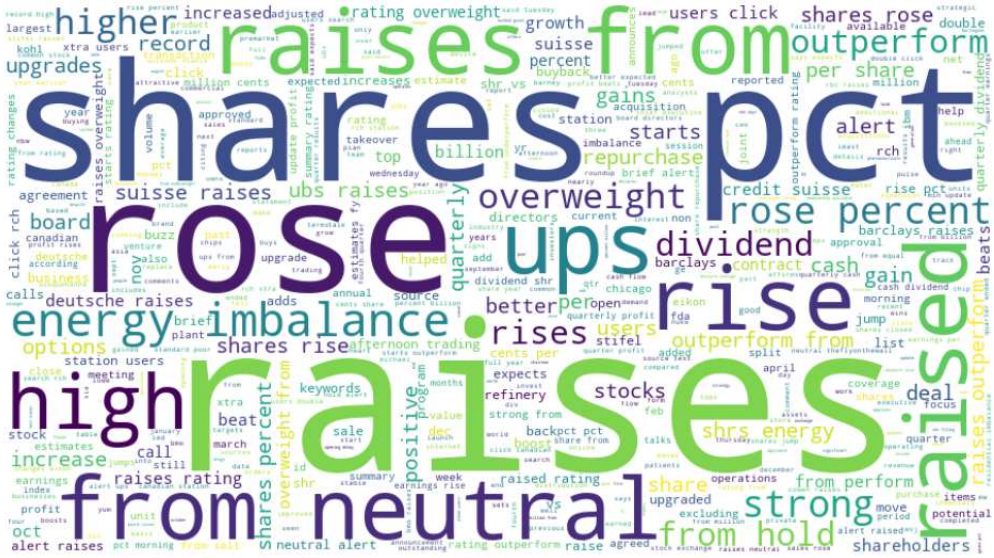


Figure 2.17: Word cloud of the top 500 positive word tokens with unigrams and bigrams as features

Figure 2.18: Word cloud of the top 500 negative word tokens with unigrams and bigrams as features

2.4 Supervised Learning

In this thesis, supervised machine learning techniques are used for the classification task. Supervised learning is a sub-field of machine learning, that uses labelled input data for training (Géron, 2017). Two different network architectures are implemented. The first is a classical feedforward neural network (NN) with several hidden layers. The second architecture is a convolutional neural network (CNN). CNNs have proven to be very effective for image classification tasks and became the standard architecture in this field. Furthermore, researchers found that CNNs are also well suited for NLP tasks, especially for sentiment analysis (Severyn and Moschitti, 2015; Vargas, Lima, and Evsukoff, 2017).

As already mentioned in Section 2.1.2, the cleaned dataset contains in total 118,567 positive, 326,231 neutral, and 116,519 negative labelled news articles. The next step is to split the data into train, validation and test sets. Data ranging from January 1996 until December 2017 is used for training and validation which results in a training and validation dataset consisting of 108,200 positive, 290,566 neutral and 106,845 negative observations. As can be seen, this data set is not well balanced, as there are almost three times as many neutral observations than positive or negative ones. For illustration, consider a very simple model that classifies all observations as neutral. Due to the imbalanced dataset, always predicting a neutral return is correct with a probability of 58.41%, which is also known as the null-accuracy. Neutral news would be classified with an accuracy of 100%, while for positive and negative articles this model would achieve 0% accuracy. Neural networks are usually designed to increase the overall accuracy. However, with an unbalanced dataset as input, the model would learn to predict all observations as neutral since this leads to a high overall accuracy (Elite Data Science, 2017). In order to overcome this issue, minority classes are up-sampled to obtain equal class counts. This is achieved by randomly duplicating positive and negative news with use of the *resample* function of the *scikit-learn* library in python. This results in a dataset with 290,566 observations in each class.
The next step is to split the training and validation dataset into training data and validation data in a ratio of 90% to 10%. This results in a total of 784,528 observations for training and 87,170 observations for validation.

Furthermore, the test dataset consists of news articles published between January 2018 and January 2020. It comprises of 10,367 positive, 35,665 neutral and 9,674 negative articles. For the majority of classification tasks, temporal order of observations has no influence. In the case of financial news though, temporal order makes a difference since the sentiment associated with features may change over time. President elections or a pandemic like the spread of the corona virus can change the sentiment of words associated with those topics. To obtain an unbiased test performance, temporal order of news must be taken into account. Since no one can look into the future, it must be ensured that the algorithm makes predictions based on past information. Therefore, the test set is upfront separated from the trainings and the validation set.

After splitting the data, the feature extraction methods described in Section 2.2.3 are applied. In the following sections, several feedforward neural networks are implemented that use bag-of-words as well as word embeddings as input features. Furthermore, a convolutional neural network is implemented which uses word embeddings as input features.

2.4.1 Multi-Layer Perceptron

The multi-layer perceptron (MLP) is a widely used supervised learning algorithm. It learns a function $f(\cdot) : R^m \rightarrow R^o$ by training on a dataset. The superscript m denotes the number of input dimensions and o indicates the number of output dimensions. The basic structure of this algorithm is shown in Figure 2.19. The layer on the left side, is the input layer $\mathbf{x}$ which consists of a set of neurons x_i ... x_n that represent the input features (Pedregosa et al., 2011). Layers between the input and output layer are called hidden layers. In addition, the model consists of parameters, the weights $\mathbf{W}$ and biases $\mathbf{b}$. Before training, those parameters are randomly initialized.
For the sentiment classification task in this thesis, the output layer is three dimensional since news articles are classified into three categories: positive, neutral and negative.

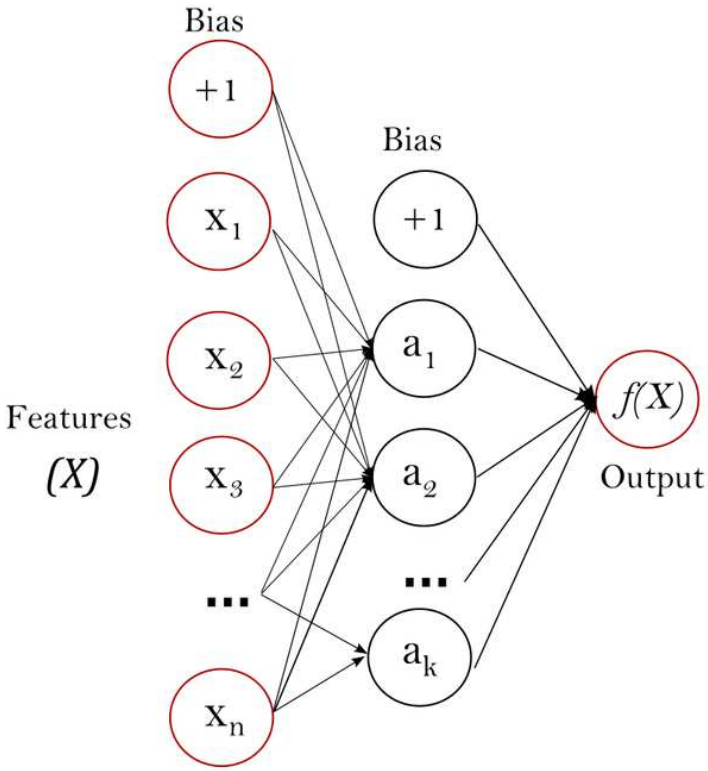


Figure 2.19: Multi-layer perceptron with one hidden layer (Pedregosa et al., 2011)

Training a neural network is an iterative process. Each iteration consists of four steps:

1. Forward propagation
2. Computing the cost function
3. Backward propagation
4. Gradient descent

During forward propagation, the network takes an input vector $\mathbf{x}$ and computes an output vector $\hat{\mathbf{y}}$. Consider the following neural network in Figure 2.20 with three hidden layers. Each layer of the network is connected with the next layer via a weight matrix $\mathbf{W_1}$ ... $\mathbf{W_4}$. The weights of these matrices are the learnable parameters of the network. The number of units in layer l is denoted as n_l and the total number of layers is denoted as L.

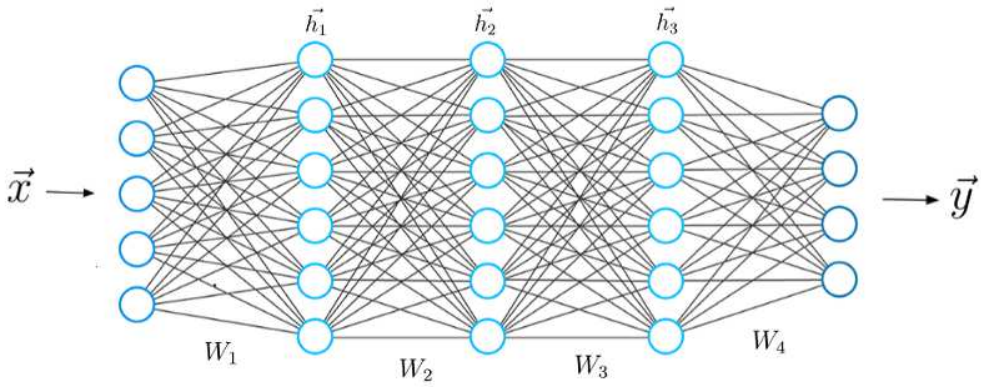


Figure 2.20: Neural network with three hidden layers (Oppermann, 2019)

Every node of the network computes a linear operation followed by a non-linear activation function g (see Section 2.4.2). Figure 2.21 shows a single node of the neural network in more detail. The vector $\mathbf{z}^{[1]}$ of the linear operation in layer h_1 is calculated with the dot product of the weight matrix $\mathbf{W}^{[1]}$ with the input vector $\mathbf{x} = \mathbf{a}^{[0]}$ plus a bias term $\mathbf{b}^{[1]}$ (2.23). The output vector $\mathbf{a}^{[1]}$ is then calculated by applying an activation function $g^{[1]}$ on $\mathbf{z}^{[1]}$ (2.24). Those calculations are executed for each layer. The resulting output vector $\mathbf{a}^{[1]}$ is the input vector of the next layer h_2 where the same calculations are computed again. The process is repeated until the output $\mathbf{a}^{[\mathbf{L}]}$ of the last layer, which equals the output vector $\hat{\mathbf{y}}$, is calculated.

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]} \tag{2.23}$$

$$\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]}) \tag{2.24}$$

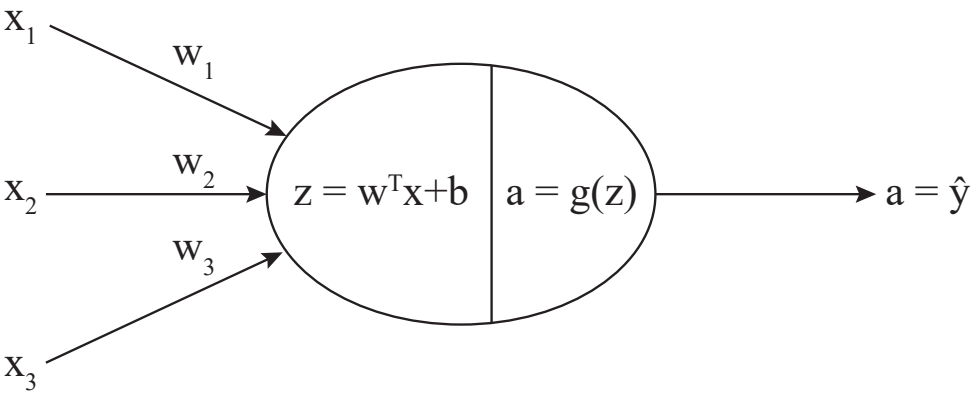


Figure 2.21: A single node of the MLP

The model is trained with thousands of observations to achieve a high prediction accuracy. For efficient computing, those input vectors are combined to one input matrix $\mathbf{X} = \mathbf{A}^{[0]}$. This matrix is of size ($n_x \times m$) where n_x is the number of input features and m is the number of observations. Equations (2.25) and (2.26) describe the forward propagation in a general form:

$$\mathbf{Z}^{[\mathbf{l}]} = \mathbf{W}^{[\mathbf{l}]}\mathbf{A}^{[\mathbf{l-1}]} + \mathbf{B}^{[\mathbf{l}]} \tag{2.25}$$

$$\mathbf{A}^{[\mathbf{l}]} = g^{[l]}(\mathbf{Z}^{[\mathbf{l}]}) \tag{2.26}$$

The weight matrix $\mathbf{W}$ is of shape ($n_l \times n_{l-1}$), whereas the matrices $\mathbf{Z}$, $\mathbf{A}$ and the matrix $\mathbf{B}$ are of shape ($n_l \times m$) with the superscript l ranging from 0 to L. The next step is to calculate the cost function J(W, b). The cost function takes the predicted output and the actual label as input and computes the loss. This loss is than averaged over all observations of the training sample.

$$J(W^{[1]}, b^{[1]}, \ ..., \ W^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) \tag{2.27}$$

Here, ŷ represents the predicted output, while y denotes the labelled output. For classification tasks, it is common to use the cross-entropy loss function $\mathcal{L}(\hat{y}^{(i)}, y^{(i)})$ in (2.28) (Pedregosa et al., 2011).

$$\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -(y \ \log(\hat{y}) + (1-y) \ \log(1-\hat{y})) \tag{2.28}$$

The objective when training a neural network is to minimize the cost function. This is achieved via backpropagation. The backpropagation algorithm operates backward through the network, from the output to the input. Therefore, it is necessary to calculate the derivative of each parameter with regard to the cost function (Géron, 2017).

$$dW^{[l]} = \frac{dJ}{dW^{[l]}} \tag{2.29}$$

$$db^{[l]} = \frac{dJ}{db^{[l]}} \tag{2.30}$$

The last step of each iteration is to update the parameters. This is achieved with gradient descent.

$$W^{[l]} := W^{[l]} - \alpha dW^{[l]} \tag{2.31}$$

$$b^{[l]} := b^{[l]} - \alpha db^{[l]} \tag{2.32}$$

The learning rate α determines the step size that is decreased with each iteration. Figure 2.22 shows the trajectory of gradient descent visualized in a three dimensional space. Usually neural networks are of much higher dimensionality, but this would be difficult to visualize. The learning rate is an important tuning parameter. It determines the number of iterations that are necessary to reach the minima and thus it also influences the training time. If the learning rate is too low, the algorithm would take a long time to train, while a high learning rate could lead to a divergent trajectory that never reaches the global minima.

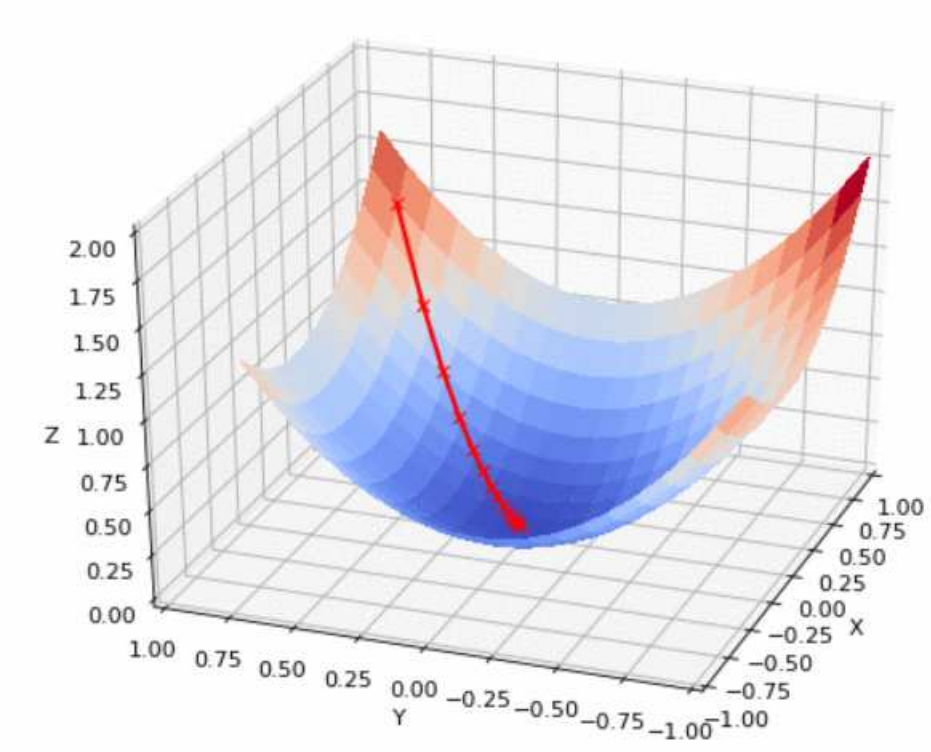


Figure 2.22: Gradient descent in three dimensional space (Kathuria, 2018)

2.4.2 Activation Functions

The purpose of the activation function is to add a non-linearity to the model. Only by this non-linearity the model is able to learn complex patterns. Without the activation function, the model would just consist of linear-affine operations which are the dot product and summation. The model would remain a linear classifier (Oppermann, 2019). An overview of the most important activation functions is shown in Figure 2.23. A few years ago, the sigmoid function was the most commonly used activation function. It maps the inputs in a range between 0 and 1. However, this function has some drawbacks:

1. The sigmoid function has flat tails for large and small input values. When the gradient is computed in those areas, its values become very small. The result is a poor learning algorithm. Furthermore, it amplifies the vanishing gradient problem of deep neural networks (Adventures in Machine Learning, 2018).

2. In addition, the results of the sigmoid function are not zero-centered, which makes learning

more difficult and unstable (Oppermann, 2019). The tanh function in Figure 2.23 solves this issue since it ranges from -1 to 1. But the problem of vanishing gradients still exists due to its flat tails.
A function that solves the issue of vanishing gradients is the ReLU (rectified linear unit) function, which is the most commonly used activation function today. The function has a simple threshold at 0: ReLU(z) = max(0, z). If z < 0: ReLU(z) = 0, otherwise ReLU(z) = z. A further advantage of this function is that its gradients are easy and fast to compute since it does not contain any exponential functions. However, like the sigmoid function, the ReLU has the disadvantage that the results are not zero-centered. In practice, this problem is solved by batch normalization, especially for deep neural networks (Ioffe and Szegedy, 2015). Another drawback exists for negative input values. In those cases the gradient gets zero which deactivates neurons. Once neurons are deactivated, they output zero and can no longer be activated (Géron, 2017).
The Leaky-ReLU has therefore a flat slope for negative values to avoid zero gradients. The hyperparameter a defines the steepness of this slope. It is typically set to 0.01 which ensures that the gradients of the Leaky-ReLU never become zero (Géron, 2017).

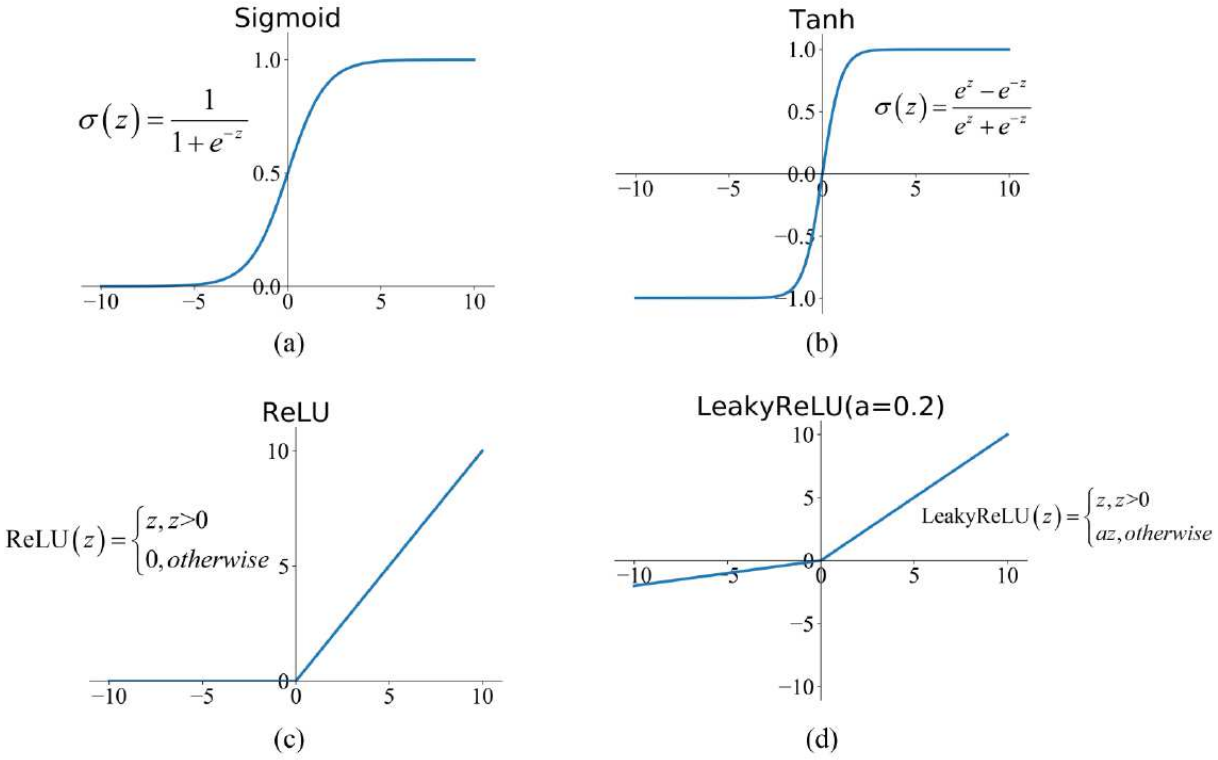


Figure 2.23: Commonly used activation functions: (a) Sigmoid function, (b) Tanh function, (c) ReLU function, (d) Leaky-ReLU function (Junxi Feng et al., 2019)

2.4.3 Feedforward Neural Network (NN)

2.4.3.1 Neural Network with Bag-of-Words as Input Features

Machine learning is an iterative process. Therefore, several models are build, tested and optimized. Both, sparse representations from the bag-of-words approach, as well as dense word embeddings are considered as input features. Many hyper parameters can be tuned to increase the performance of the models. Some of these tuning parameters are found in the field of feature extraction. One parameter is the n-gram size. Tests are conducted with unigrams, bigrams and trigrams as well as with combinations of those. It turned out that a combination of unigrams and bigrams achieved the highest accuracy when the input size is limited to 25,000 (see Figure 2.9).

Another parameter is the maximum number of input features. Figure 2.24 shows the frequency of input features with a combination of unigrams, bigrams and trigrams for the 50,000 most common features. While few characteristics occur very frequently, there is a very long tail with rare n-grams. In total there are 163,477 unique unigrams, 6,153,596 unique bigrams and 15,705,994 unique trigrams which sums up to a total of 22,023,067 features. Considering all features would be computationally expensive as well as overfitting would be very likely. This is why only the most frequent features are considered, while rare features are not taken into account. Tests are done with 25,000 and 50,000 input features.

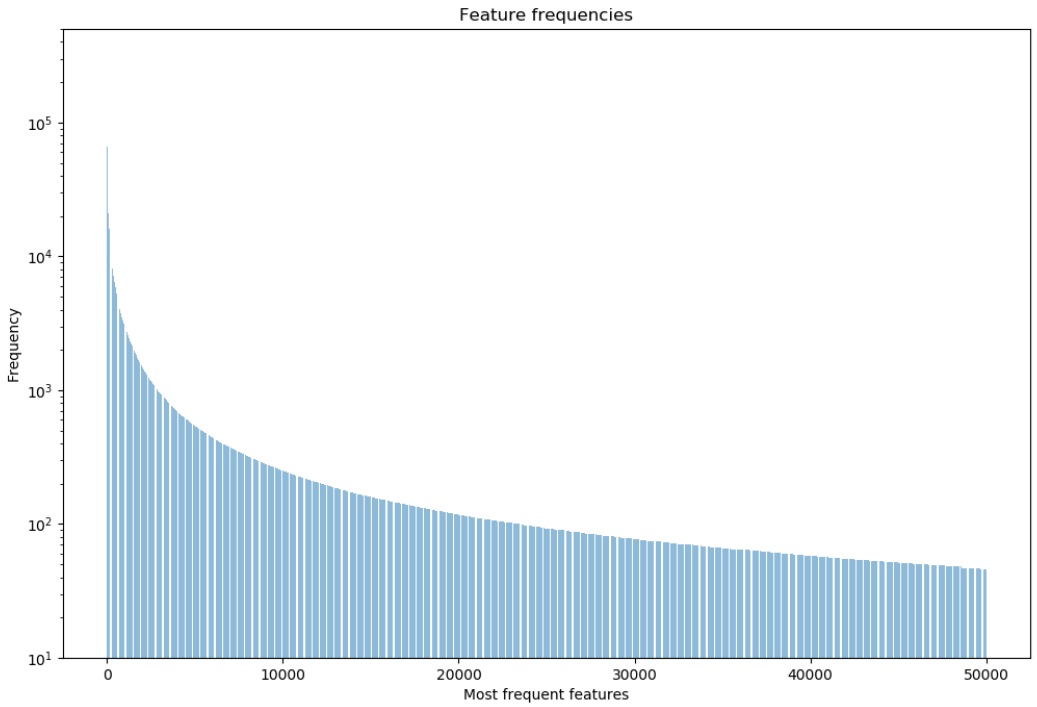


Figure 2.24: Frequency of bag-of-words features considering unigrams, bigrams and trigrams

In addition, experiments are carried out to determine whether stemming can increase performance. Stemming is a dimension reduction technique that removes morphological and inflexional endings from words. As an example, a stemmer would reduce the words democrats

(noun), democratic (adjective), and democratize (verb) to the same root democrat (Cambria et al., 2017). With stemming, the size of the vocabulary is reduced from 163,477 to 136,025 unique words. The results show, that stemming further increases the accuracy between 1 to 2% (see Table 3.9).

Neural networks come with another set of tuning parameters. The number of hidden units, the number of layers, the optimization algorithm, the learning rate and the mini-batch size are the most important ones. In addition, regularisation techniques like l2-regularization and dropout are considered to reduce overfitting. Figure 2.25 shows a neural network architecture with three fully connected hidden layers. The ReLU function is the chosen activation function for all hidden layers, while the softmax function (2.15) is used in the output layer to obtain probabilities for each class, positive, neutral and negative.

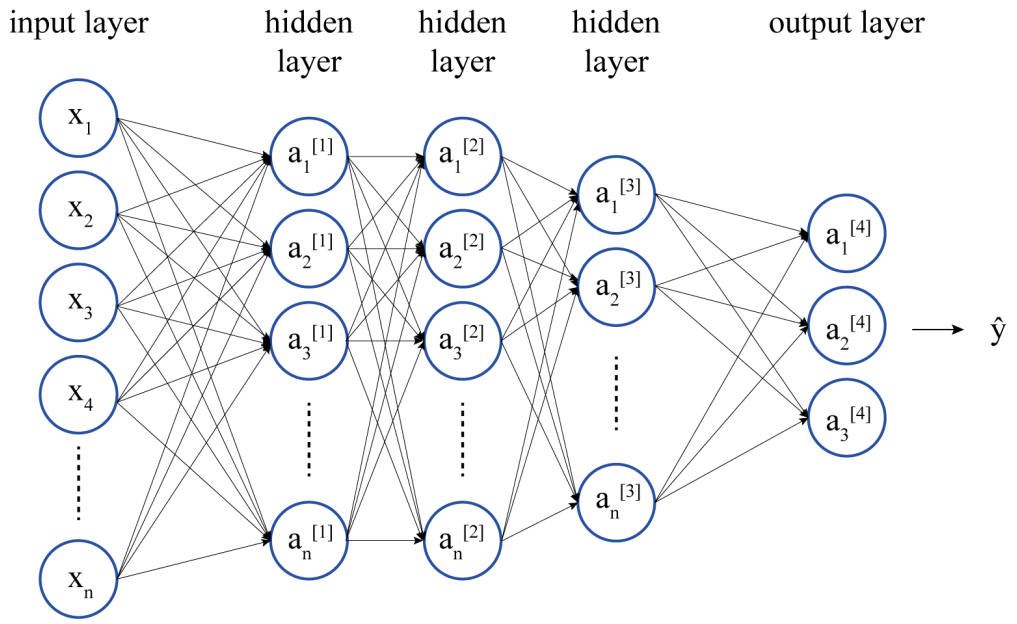


Figure 2.25: Neural network with three hidden layers

Formula (2.33) shows the calculation of the softmax function for the first node in the output layer.

$$P_1 = \frac{e^{a_1^{[4]}}}{\sum_n e^{a_n^{[4]}}} \tag{2.33}$$

The number of nodes in the input layer is determined by the number of input features. The highest accuracy was achieved with a network of 256 nodes in the first and second hidden layer and 64 nodes in the third hidden layer.
The implementation in python is done with the library *Tensorflow* and its high level API *Keras*. The following code imports the libraries and creates the neural network architecture from Figure 2.25.

```
# Import:
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import models

max_features = 25000

# Implement the neural network architecture:
model = Sequential()
model.add(Dense(256, activation='relu', input_dim=max_features))
model.add(Dense(256, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.summary()
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=[
                                    'accuracy'])\\
```

Adam, which stands for adaptive moment estimation, is an efficient stochastic optimizer. It combines the advantages of the two optimizers RSMProp and AdaGrad. Adam calculates individual adaptive learning rates for different parameters from first and second order gradient moments (Kingma and Ba, 2014). This optimization method is used for all models in this thesis. The model is then trained with the following command:

```
history = model.fit_generator(generator=batch_generator(x_train, y_train,
                                    batch_size), epochs=10,
                                    validation_data=(x_valid, y_valid),
                                    steps_per_epoch=x_train.shape[0]/
                                    batch_size)
```

The training data x_data is stored as a sparse matrix for reasons of memory efficiency. It is of the shape ($max\_features \times \#observations\ (m)$) which is $(25,000 \times 784,528)$ for the case of 25,000 input features. The fit function, however, can not handle matrices of the sparse format. Therefore, it needs to be converted into an array. In order to avoid memory issues, the training data is split into batches which are then converted into arrays and used for training. This is done by calling the batch_generator function.

```
def batch_generator(x_data, y_data, batch_size):
    samples_per_epoch = x_data.shape[0]
    number_of_batches = samples_per_epoch/batch_size
    counter=0
    index = np.arange(np.shape(y_data)[0])
    while 1:
        index_batch = index[batch_size*counter:batch_size*(counter+1)]
        x_batch = x_data[index_batch,:].toarray()
        y_batch = y_data[index_batch, :]
        counter += 1
        yield x_batch,y_batch
        if (counter > number_of_batches):
            counter=0
```

2.4.3.2 Neural Network with Word Embeddings as Input Features

Using dense vector representations as input features has several advantages in contrast to sparse representations. As already mentioned in Section 2.2.3.5, dense representations are memory efficient and thus more efficient to compute (Tomar, 2019).
In addition, a machine learning technique called transfer learning can be applied. Transfer learning is often applied when the available dataset is rather small. In the field of natural language processing, word embeddings are usually learned on large datasets, often with several hundred million words by very powerful computers. The learned weights are then used to initialize models for smaller datasets. According to Chung, H.-Y. Lee, and Glass (2017) this can increase performance and reduce the data required for fine tuning. Severyn and Moschitti (2015) achieve a significant performance increase with transfer learning. Similar to their work, in this thesis word embeddings are also used to initialize the parameters of the neural network. Since the training and validation dataset is large, this is used for learning the word vectors with the model word2vec (Mikolov et al., 2013).
To feed a neural network with word embeddings, news articles must first be converted into a numerical representation. This is achieved with a one-hot encoding, that transforms each word into a integer number. Therefore, the function *Tokenizer* of the *Keras* library is used:

```
from keras.preprocessing.text import Tokenizer

tk = Tokenizer(num_words = vocab_size)
tk.fit_on_texts(x_train)

x_train_seq = tk.texts_to_sequences(x_train)
x_valid_seq = tk.texts_to_sequences(x_valid)
```

The *fit_on_texts* function creates the vocabulary index in relation to word frequency. Consider the following example:

["The stock market rises for the third day in a row",
"The apple stock outperforms the market"]

After the cleaning process, the text becomes:

["stock market rises third day row,
"stock outperforms market"]

The *fit_on_texts* function creates a dictionary where every word receives a unique integer number. Lower numbers means more frequent words.

'stock': 1, 'market': 2, 'rises': 3, 'third': 4, 'day': 5, 'row': 6, 'outperforms': 7

The *text_to_sequences* function transforms each article into a sequence of integers. Each word of the articles is replaced by the corresponding integer value that is derived from the *fit_on_texts* dictionary:

[[1, 2, 3, 4, 5, 6], [1, 7, 2]]

What is more, is that neural network can only handle input feature vectors with equal length. Therefore news articles need to be limited in length to a maximum number of words. An analysis of the training data shows the following statistics:

count	783,607
mean	78.99
std	173.03
min	1.00
25th percentil	8.00
50th percentil	14.00
75th percentil	69.00
max	8983.00

Table 2.17: Descriptive statistics of the training dataset

The training dataset consists of a total of 783,607 news articles with an average length of 78.99 words. 75% of all articles contain less than 69 words. The longest article has a length of 8983 words. In order to get observations with equal length, articles are limited to a maximum of 1000 words. Articles that contain more words are cropped while articles with fewer words are filled up with zeros. This is done with the *Keras* function *pad_sequences*:

```
from keras.preprocessing.sequence import pad_sequences

x_train_emb = pad_sequences(x_train_seq, maxlen=1000)
x_valid_emb = pad_sequences(x_valid_seq, maxlen=1000)
```

Articles are now transformed to sequences of integers, each with a length of 1000:

$$[[0, 0, 0, 0, ..., 0, 1, 2, 3, 4, 5, 6],$$
$$[0, 0, 0, 0, ..., 0, 0, 0, 0, 1, 7, 2]]$$

When word embeddings are used as input features, the model architecture looks slightly different (see Figure 2.26). Input features are of the size ($max\_features \times embedding\ dimension \times \#observations\ (m)$). The embedding dimension is set to 150, thus the dimension of the input features becomes $(1,000 \times 150 \times 784,528)$. Furthermore, a flatten layer is used to transform this three dimensional embedding layer into an two dimensional array of the size $(150,000 \times 784,528)$.

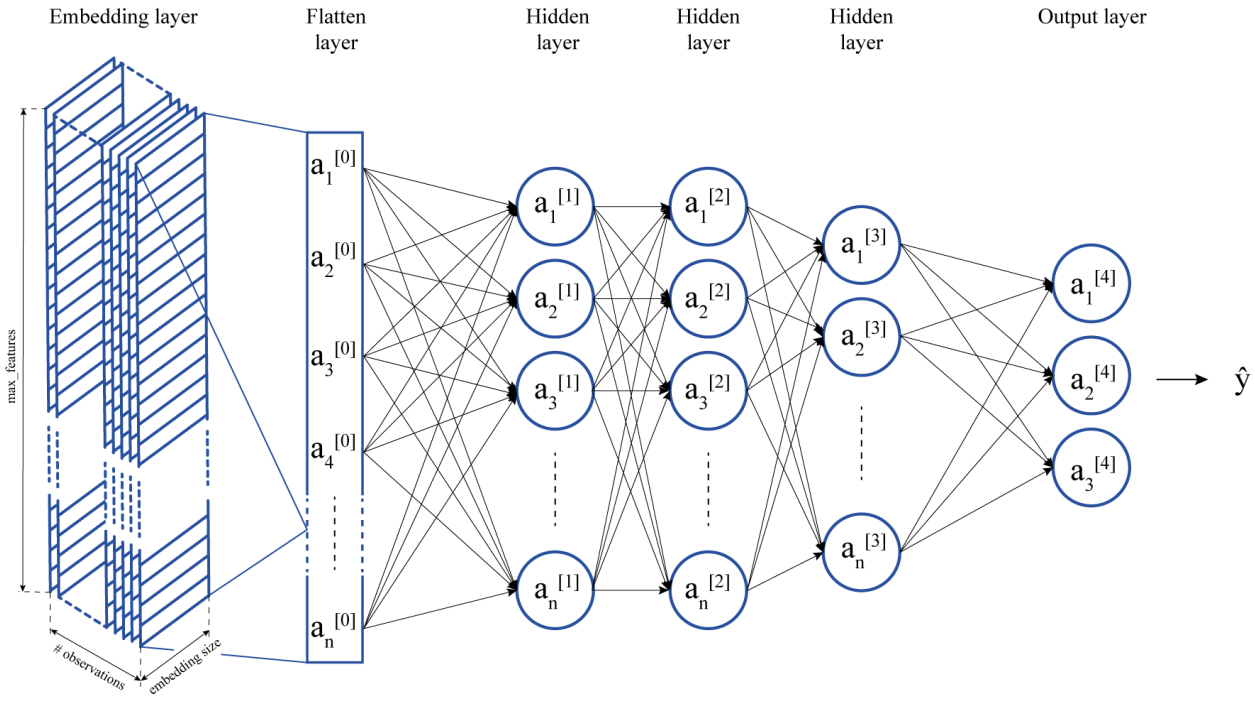


Figure 2.26: Neural network with three hidden layers and embedding layer

The first layer of the model is the embedding layer. It transforms the integer numbers of the tokenized articles into dense vectors of fixed size. The embedding dimensions are determined by the arguments, max_features, embedding_dimension and #observations (m). The weights of the embedding layer are initialized with the embedding matrix. This matrix is simply created by concatenating word embeddings for all words in the vocabulary. The argument 'trainable' determines if those pre-trained embeddings should also be trained or not.

Dropout is a popular regularization technique used to prevent overfitting. In every interation, the dropout algorithm sets randomly a distinct amount of nodes to zero during the forward propagation. In this model, the dropout rate is set to 0.4. This deactivates 40% of the nodes in each iteration. Dropout is explained in more detail in Section 3.2.1.

Them implementation of this model is done with the following lines of code:

```
# Import:
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Embedding, Dropout
from tensorflow.keras import models

vocab_size = 25000
max_len = 1000
embed_size = 150

# Implement the neural network architecture:
model = Sequential()
model.add(Embedding(vocab_size, embed_size, input_length=max_len, weights=
                                    [embedding_matrix], trainable=False))
model.add(Flatten())
model.add(Dropout(0.4))
model.add(Dense(256, activation='relu', input_dim=max_features))
model.add(Dropout(0.4))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(256, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.summary()
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=[
                                    'accuracy'])
```

2.4.4 Convolutional Neural Network (CNN)

Convolutional neural networks are originally designed for image classification tasks. Due to their impressive performance they became the state-of-the art in this domain. But CNNs also proved to be very powerful in the field of natural language processing (NLP). Yih et al. (2011) have demonstrated the effectiveness of CNNs for semantic parsing and Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom (2014) has successfully applied CNNs for sentence modelling. Severyn and Moschitti (2015) has implemented a classifier for sentiment analysis with CNNs and achieved excellent results. A very similar network is implemented in this thesis. The network architecture is shown in Figure 2.27

A CNN is structured as a series of stages. It's main building blocks are convolutional layers and pooling layers (LeCun, Bengio, and Hinton, 2015). The input layer of the network, the embedding layer is of the dimension ($max\_features$ (s) $\times$ $embedding\_dimension$ (d) $\times$ $\#observations$). A convolution is a operation between the input matrix $S \in \mathbb{R}^{d \times |s|}$ and a filter $F \in \mathbb{R}^{d \times |m|}$ that results in a vector $c \in \mathbb{R}^{|s|-m+1}$. Each component is computed with Formula 2.34:

$$c_i = (S * F)_i = \sum_{k,j} (S_{[:,i-m+1:]} \otimes F)_{kj} \tag{2.34}$$

$\otimes$ denotes the element-wise multiplication and the term $S_{[:,i-m+1:]}$ is a matrix slice of the width m that slides across the columns. A bias vector $b \in \mathbb{R}^n$ is added to the result of the convolution which enables the network to learn appropriate thresholds (Severyn and Moschitti, 2015).

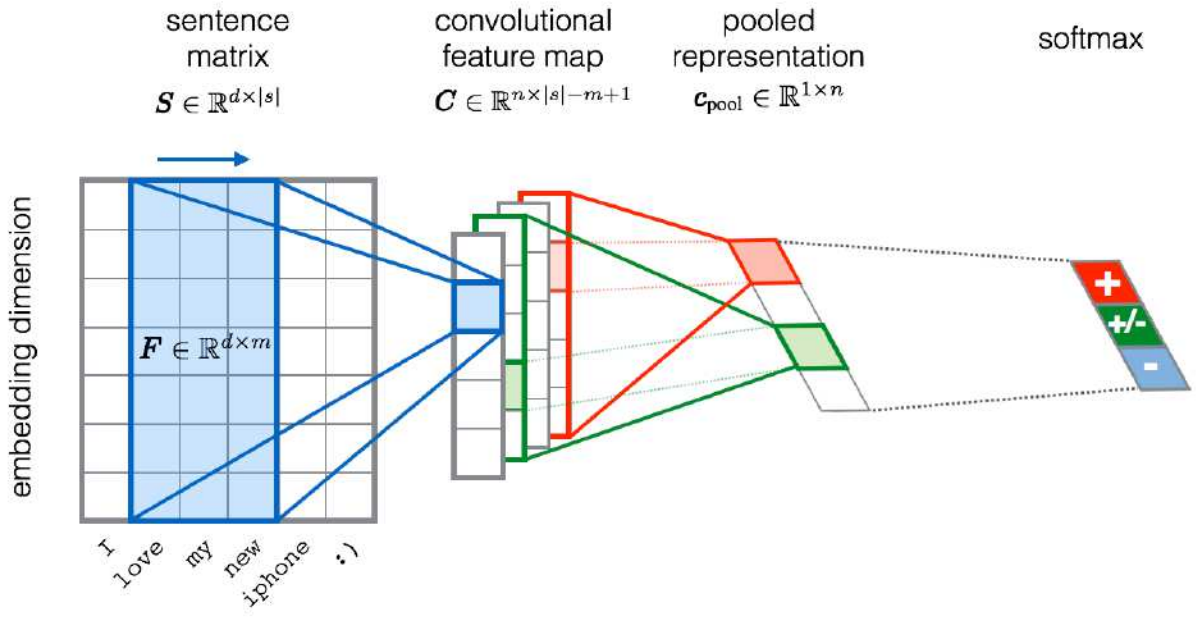


Figure 2.27: CNN architecture for sentiment analysis (Severyn and Moschitti, 2015)

In practice, multiple filters convolve in parallel across the input matrix to generate multiple feature maps. A set of filters forms a filter bank $F \in \mathbb{R}^{n \times d \times |m|}$ that results in a feature map matrix $C \in \mathbb{R}^{n \times (|s|-m+1)}$ (Severyn and Moschitti, 2015).

Each filter bank has a constant width m. The proposed network consists of three filter banks with different widths of m in {3,4,5}. The different filters are used to extract different patters from the input matrix. The idea behind several filters with different widths is the same as using different n-grams. That enables the network to derive information from word order. The second building block of this CNN are pooling layers. Their purpose is to reduce dimensionality and extract the most active features from the feature maps. Several different pooling techniques such as average pooling, k-max pooling and max pooling can be distinguished. The proposed CNN uses max pooling which simply extracts the maximum value of each feature vector c_i with $pool(c_i) : \mathbb{R}^{1\times(|s|-m+1)} \rightarrow \mathbb{R}$. The output layer is a fully connected softmax layer which is already described in Section 2.4.3.2.

The following code snippet shows the implementation of this network architecture in python with slight modifications. A fully connected dense layer, followed by a dropout layer is included right before the softmax layer. Each filter bank consists of 150 filters.

```
text_seq_input = Input(shape=(MAX_LEN,), dtype='int32')
text_embedding = Embedding(vocab_size, embed_size, input_length=MAX_LEN,
                                    weights=[embedding_matrix], trainable
                                    =True)(text_seq_input)

filter_sizes = [3,4,5]
convs = []
for filter_size in filter_sizes:
    l_conv = Conv1D(filters=150, kernel_size=filter_size, padding='same',
                                        activation='relu')(text_embedding
                                        )
    l_pool = MaxPool1D(filter_size)(l_conv)
    convs.append(l_pool)

l_merge = Concatenate(axis=1)(convs)
l_cov1= Conv1D(150, 5, activation='relu')(l_merge)
l_pool1 = MaxPool1D(5)(l_cov1)
l_flat = Flatten()(l_pool1)
l_dense_1 = Dense(256, activation='relu')(l_flat)
l_dropout_1 = Dropout(0.4)(l_dense_1)
l_out = Dense(3, activation='softmax')(l_dropout_1)
model = Model(inputs=[text_seq_input], outputs=l_out)
model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=[
                                    'categorical_accuracy'])
```

Chapter 3

Experiments and Evaluation

3.1 Performance Metrics

Different metrics are used to determine the model performance. For simplicity, those metrics are first introduced for binary classification where each observation m_i is either positive or negative. A classifier then maps from the observations to predicted classes. There are four possible results for the prediction of each observation. If the true class of the observation is positive and it is predicted as positive, it is considered as true positive. Otherwise, if it is predicted as negative, it is considered as false negative. If the true class of the observation is negative and it is correctly predicted as negative, it is counted as true negative. Otherwise, if it is predicted as positive it is considered as false positive. Those relationships can be visualized by drawing a confusion matrix, which is shown in Figure 3.1. The columns represent the true classes while the rows represent the predicted classes. Several performance metrics can be derived from this confusion matrix. The accuracy (3.1) of the classifier is calculated by dividing the true positive (TP) and true negative (TN) values by the total number of observations (P+N) (David L. Olson, Dursun Delen, 2008). Precision (3.2) denotes the proportion of predicted positive cases that are actually true positives. It is calculated by dividing the true positives (TP) by the sum of positive predicted observations. Recall (3.3) is the fraction of true positive cases that are correctly predicted to be positive. It is calculated by dividing the true positives (TP) by the sum of positive observations (Powers, 2008).
With those two measures, the F_1-score (3.4) can be calculated. It is defined as the harmonic mean of precision and recall and ranges between 0 and 1, where 1 is the highest score and 0 the lowest score (Pedregosa et al., 2011).
The F_1-score is an frequently used metric to compare the overall performance of different models since it takes precision and recall into account. Furthermore, it need to be mentioned that recall and precision and thus also the F_1-score focus only on positive observations and predictions. None of them takes negative observations into account. Inverse metrics, that focus solely on negative observations and predictions can be calculated as Powers (2008) has shown.

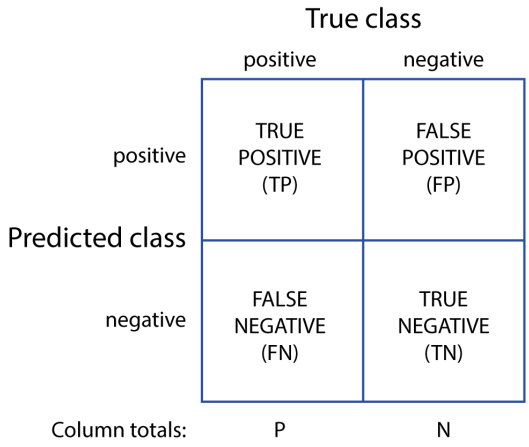


Figure 3.1: Confusion matrix for binary classification

$$accuracy = \frac{TP + TN}{P + N} \tag{3.1}$$

$$precision = \frac{TP}{TP + FP} \tag{3.2}$$

$$recall = \frac{TP}{P} \tag{3.3}$$

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} \tag{3.4}$$

For the case of multi-class prediction, the number of possible results raises in relation to the number of different classes c by c^2. The confusion matrix for the three classes positive, neutral and negative is of the shape (3×3) which is shown in Figure 3.2. It describes the results with regard to the positive class. In the multi-class case, the performance metrics (3.5 - 3.8), which are shown for the positive class, need be calculated for all three classes. With those results, the macro-avg.-F_1-score is calculated by the arithmetic mean of the three F_1-scores.

$$accuracy_{\_P} = \frac{TP + TN_{EE} + TN_{NE} + TN_{EN} + TN_{NN}}{P + E + N} \tag{3.5}$$

$$precision_{\_P} = \frac{TP}{TP + FP_E + FP_N} \tag{3.6}$$

$$recall_{\_P} = \frac{TP}{P} \tag{3.7}$$

$$F_{1\_P} = \frac{2}{\frac{1}{precision_{\_P}} + \frac{1}{recall_{\_P}}} \tag{3.8}$$

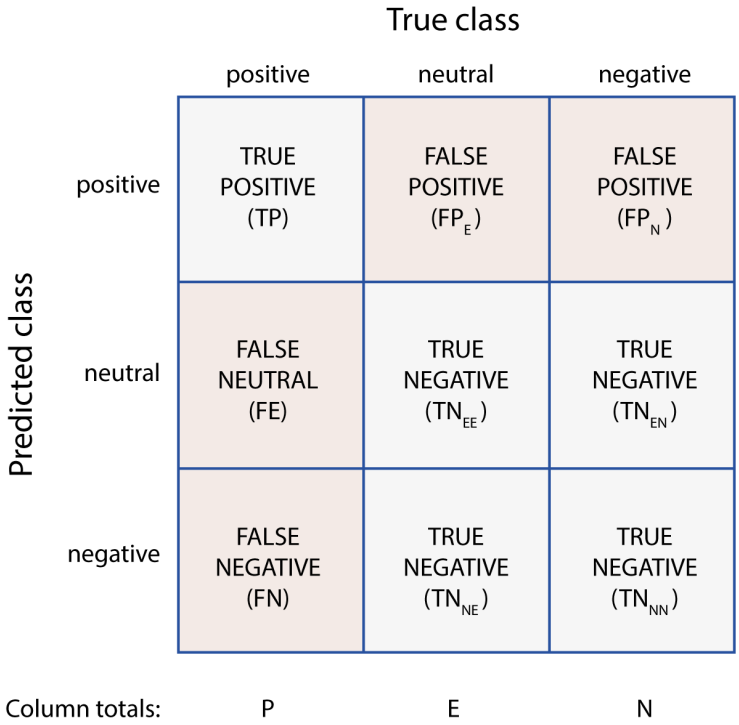


Figure 3.2: Confusion matrix for multi-class predictions shown in regard to the positive class (Starmer, 2018)

3.2 Bag-of-Words Models

The following networks are trained with a combination of unigrams, bigrams and triagrams as input features, derived from the tf-idf vectorizer. The number of features is limited to 25,000 and the news articles are not stemmed. All models use the adam optimizer with a learning rate of 0.001 and a beta of 0.9.
Each of the following four deep neural networks has seven hidden layers and a dropout layer after every dense layer. The first neural network architecture (DNN_1) has 64 nodes in each hidden layer and a dropout rate of 0.15. The model is summarized in Table 3.1. The second architecture (DNN_2) has 128 nodes in each hidden layer and a dropout rate of 0.30 (see Table 3.2). The third architecture has 200 nodes in each hidden layer and a dropout rate of 0.40 (see Table 3.3). The 4^{th} network has 800 nodes in the first five layers, 600 nodes in the 6^{th} hidden layer and 400 nodes in the 7^{th} hidden layers. The dropout rate is 0.40 for all dropout layers.

The training and validation loss of those four networks is shown in Figure 3.3 and the training and validation accuracy is shown in Figure 3.4. A high variance can be observed, since the models perform much better on the training set than on the validation set. The reason for this is probably the high noise in the training data.
The best performing model, with the lowest validation loss and the highest accuracy, is the 4^{th} deep neural network (DNN_4). It achieves the lowest loss after 9 epochs. In general, the performance rises from DNN_1 to DNN_4 with an increasing number of nodes in each layer.

The average training time for the models DNN_1 and DNN_2 is 120 seconds per epoch. The third model, DNN_3 has an average training time of 130 seconds and the 4^{th} model takes 190 seconds for one epoch. The models are trained with a laptop computer on a Nvidia GTX 1650 GPU.

Table 3.5 contains the performance metrics of all four models. Model four has the highest macro-F_1-score, which is 0.726. Nonetheless, a closer look is necessary in order to find the best model for the task of stock price prediction. Consider the results of the model DNN_4. The recall of the negative class is 0.838 which means that 83.8% of all negative news articles are predicted correctly as negative. However, precision of the negative class is only 0.669. This means, of all negative predictions only 66.9% are true negative. As a consequence, chances are high that this model will perform not very well on the test set since a large amount of the predictions it makes are actually wrong. Thus, in this scenario, precision is a more important measure than recall.

Input features		**Deep neural network (DNN_1)**	
Stemming	No	# hidden layers	7
N-grams:	unigrams + bigrams + trigrams	# nodes in the hidden layers	64, 64, 64, 64, 64, 64, 64
# input features	25000	Dropout	0.15
Return normalization	Rolling window	Learning rate	0.001
		Beta	0.9

Table 3.1: Summary of the deep neural network DNN_1 and its input features

Input features		**Deep neural network (DNN_2)**	
Stemming	No	# hidden layers	7
N-grams:	unigrams + bigrams + trigrams	# nodes in the hidden layers	128, 128, 128, 128, 128, 128, 128
# input features	25000	Dropout	0.30
Return normalization	Rolling window	Learning rate	0.001
		Beta	0.9

Table 3.2: Summary of the deep neural network DNN_2 and its input features

Input features		**Deep neural network (DNN_3)**	
Stemming	No	# hidden layers	7
N-grams:	unigrams + bigrams + trigrams	# nodes in the hidden layers	200, 200, 200, 200, 200, 200, 200
# input features	25000	Dropout	0.40
Return normalization	Rolling window	Learning rate	0.001
		Beta	0.9

Table 3.3: Summary of the deep neural network DNN_3 and its input features

Input features		**Deep neural network (DNN_4)**	
Stemming	No	# hidden layers	7
N-grams:	unigrams + bigrams + trigrams	# nodes in the hidden layers	800, 800, 800, 800, 800, 600, 400
# input features	25000	Dropout	0.40
Return normalization	Rolling window	Learning rate	0.001
		Beta	0.9

Table 3.4: Summary of the deep neural network DNN_4 and its input features

	DNN_1			DNN_2			DNN_3			DNN_4			
Class	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score	# of observations
Neutral	0.721	0.524	0.607	0.728	0.564	0.636	0.622	0.650	29174	0.609	0.685	29174	29174
Negative	0.678	0.780	0.726	0.689	0.765	0.725	0.686	0.770	0.726	0.669	0.838	0.744	28959
Positive	0.676	0.760	0.716	0.690	0.769	0.727	0.739	0.712	0.725	0.761	0.738	0.749	28935
Accuracy			0.688			0.699			0.701			0.728	87068
Macro avg. F_1	0.692	0.688	**0.683**	0.702	0.699	**0.696**	0.702	0.701	**0.700**	0.737	0.728	**0.726**	87068

Table 3.5: Performance metrics of the deep neural networks DNN_1-4 on the validation dataset.

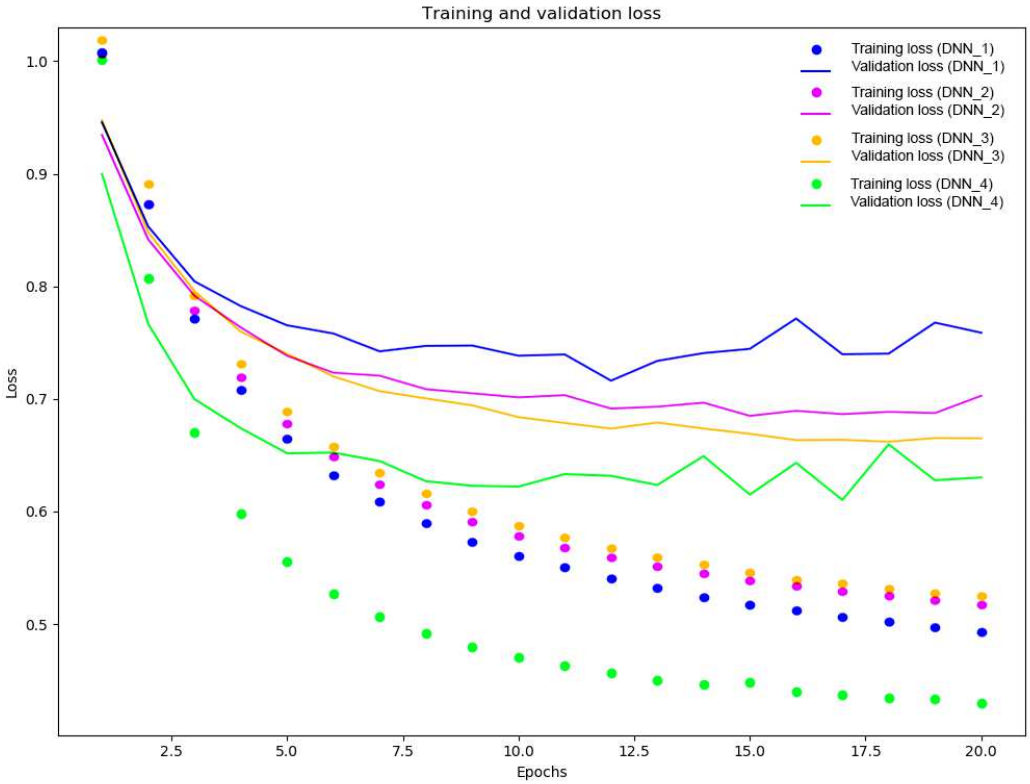


Figure 3.3: Comparison of the training and validation loss of the deep neural networks DNN_1-4

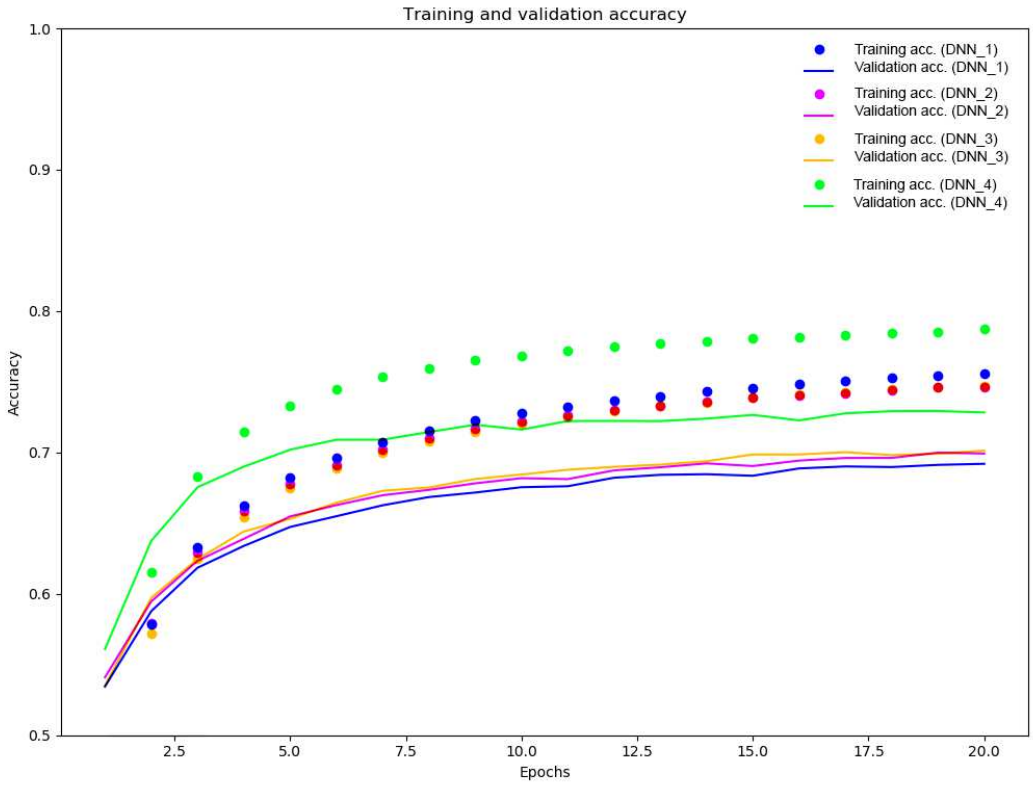


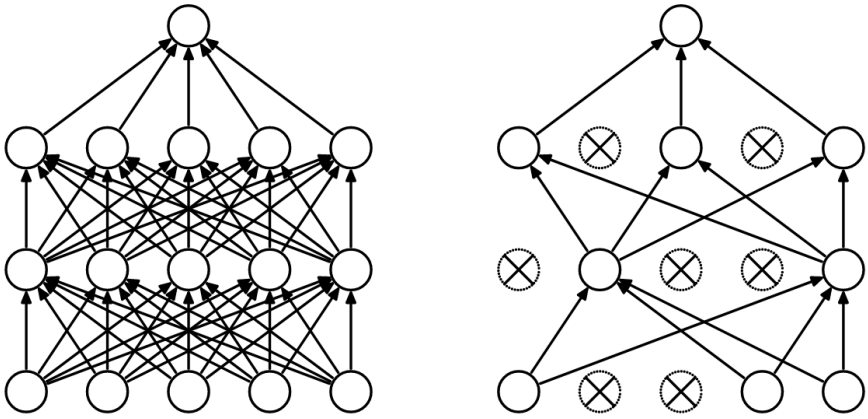
Figure 3.4: Comparison of the training and validation accuracy of the deep neural networks DNN_1-4

3.2.1 Dropout

Large neural networks that contain several hidden layers can learn very complex mappings. However, they also tend to overfit and are difficult to tune in high dimensional settings with a high level of background noise as Gentzkow, Kelly, and Taddy (2017) point out. This is also the case in this work. Overfitting occurs when a model learns statistical noise of the training data. It thus learns to consider features as relevant which actually have no predictive power. As a consequence, this leads to poor performance when the model is evaluated on unseen data, such as the validation or test data. This behaviour can be observed in Figure 3.6 where the validation loss of the blue curve rapidly increases after the second epoch due to overfitting.

According to Nitish Srivastava et al. (2014), theoretically the best way for regularization, if computational power is unlimited, is to average predictions of multiple different models. Dropout is a technique that prevents overfitting by providing a method to efficiently combine large quantities of different networks. The term "dropout" is used, because a distinct number of units is dropped out in each iteration. This means that these nodes and all their connections are temporarily disabled. The selection of those units is random. Figure 3.5 shows dropout on a small neural network. The application of dropout on a network with n

nodes results in 2^n different "thinned" networks, where each is composed of the remaining active nodes (Nitish Srivastava et al., 2014). Furthermore, the effects of different dropout rates can be observed in Figure 3.6. Without dropout, the model begins to overfit in the second epoch. An increasing dropout rate reduces the tendency to overfit the data and results in a better generalisation. This leads to a lower validation loss, which is accompanied by a higher training loss.



(a) Standard Neural Net (b) After applying dropout.

Figure 3.5: Neural network with two hidden layers. (a) No dropout is applied. (b) Dropout is applied: several units are deactivated (Nitish Srivastava et al., 2014).

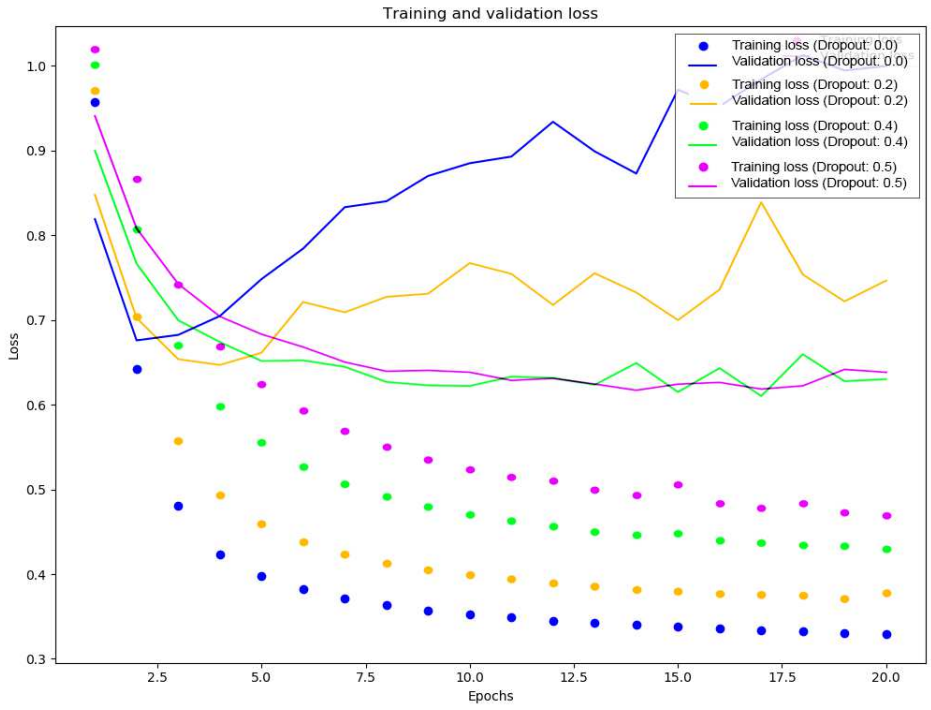


Figure 3.6: Different dropout rates applied on model DNN_4. A dropout rate of 0.40 shows the best results after 10 epochs of training.

3.2.2 Deep vs. Shallow Networks

Shallow neural networks have only a few hidden layers. Table 3.6 summarizes the properties of the neural network NN_1. It is made up of three hidden layers with 800 nodes in the first two hidden layers and 400 nodes in the third. Figure 3.7 shows the training and validation loss of this model once with dropout and once without dropout. It can be observed, that the shallow networks needs less epochs to achieve a similar validation loss, than its deeper counterparts. After seven epochs, the model with dropout reaches a macro-F_1-score of 0.727. Without dropout the model quickly overfits. Table 3.7 contains the performance metrics of this model with dropout, when trained for seven epochs. The model achieves good results, especially for positive and negative classes. The precision values are high for both, positive and negative predictions.

Input features		**Neural network (NN_1)**	
Stemming	No	# hidden layers	3
N-grams:	unigrams + bigrams + trigrams	# nodes in the hidden layers	800, 800, 400
# input features	25000	Dropout	0.40, 0.40, 0.25
Return normalization	Rolling window	Learning rate	0.001
		Beta	0.9

Table 3.6: Summary of the neural network NN_1 and its input features

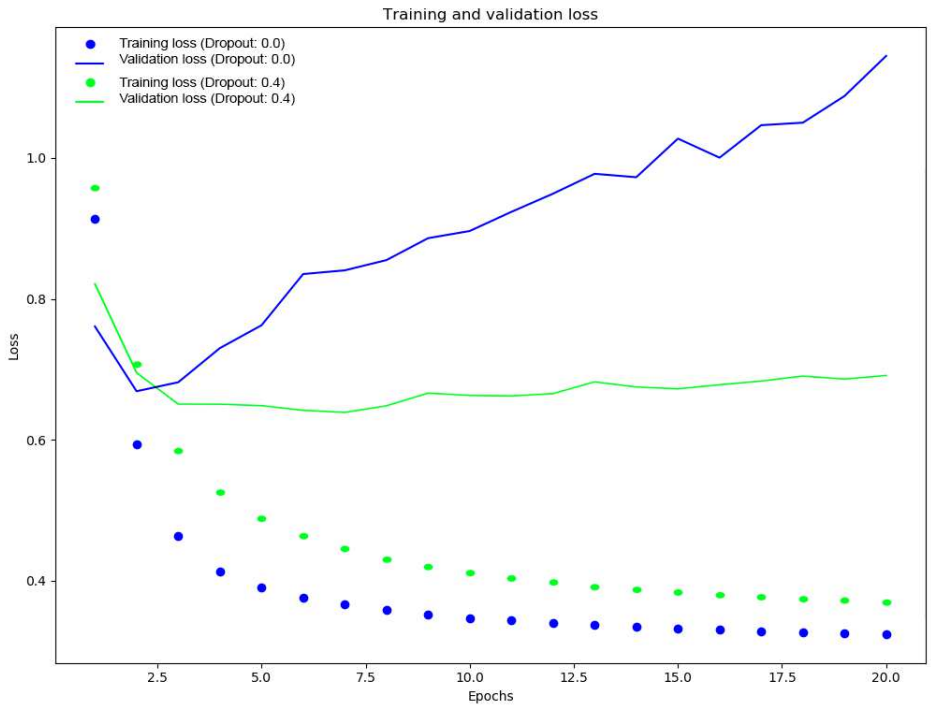


Figure 3.7: Training and validation loss of the shallow neural network NN_1 with three hidden layers. The blue lines show the loss when trained without dropout. The green lines show the loss with three dropout layers and dropout rates of 0.40, 0.40 and 0.25.

	NN_1			
Class	Precision	Recall	F1-score	# of observations
Neutral:	0.746	0.628	0.682	29174
Negative:	0.710	0.795	0.750	28959
Positive:	0.733	0.763	0.748	28935
Accuracy			0.728	87068
Macro avg. F_1	0.730	0.729	**0.727**	87068

Table 3.7: Performance metrics of the NN_1 on the validation dataset

3.3 Word Embedding Models

3.3.1 Feedforward Neural Network

The neural networks in this section use word embeddings as input features. Three feedforward neural networks with the architecture, illustrated in Figure 2.26, are summarized in Table 3.8 with different parameters. The NN_E_2 uses labels that were normalized by the market return instead of the rolling window mean return. Furthermore, the news data used for the neural network NN_E_3 is stemmed.

Settings and parameters	**Neural network NN_E_1**	**Neural network NN_E_2**	**Neural network NN_E_3**
Pretrained word vectors	word2vec	word2vec	word2vec
Stemming	no	no	**yes**
# input features	25,000	25,000	25,000
Return normalization	Rolling window	**S&P 500**	S&P 500
input length (s)	1000	1000	1000
Embedding dimension (d)	150	150	150
Trainable word vectors	True	True	True
# hidden layers	3	3	3
# nodes in the hidden layers	256, 256, 256	256, 256, 256	256, 256, 256
Dropout	0.40, 0.40, 0.40	0.40, 0.40, 0.40	0.40, 0.40, 0.40
Activation	ReLU	ReLU	ReLU
Learning rate	0.001	0.001	0.001
Beta	0.9	0.9	0.9

Table 3.8: Summary of the neural networks NN_E_1, NN_E_2 and NN_E_3

	NN_E_1			**NN_E_2**			**NN_E_3**		
Class	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Neutral:	0.617	0.641	29214	0.618	0.760	0.682	0.657	0.715	0.685
Negative:	0.692	0.726	0.709	0.749	0.755	0.752	0.730	0.806	0.766
Positive:	0.699	0.716	0.708	0.835	0.634	0.721	0.834	0.671	0.744
Accuracy			0.686			0.717			0.731
Macro avg. F_1	0.686	0.687	**0.686**	0.734	0.716	**0.718**	0.740	0.731	**0.732**

Table 3.9: Performance metrics of the neural networks, with word embeddings as input features, on the validation dataset

Training and testing of several models has shown that the best results are achieved with a limitation of 25,000 input features and an input length of 1000 words. All three models are trained over 30 epochs. It can be observed that using the market return for normalization in model NN_E_2 results in a higher precision and accuracy compared to the model NN_E_1 that uses the rolling window mean return. Furthermore, the model NN_E_3 which utilizes stemming further increases the performance in terms of the macro-F_1-score. However, the precision for negative classes is lower, compared to model NN_E_2 while the recall improved. Figures 3.8 and 3.9 show the training and validation loss and accuracy after training over 30 epochs. It can be observed, that the third model has the steepest learning curve.

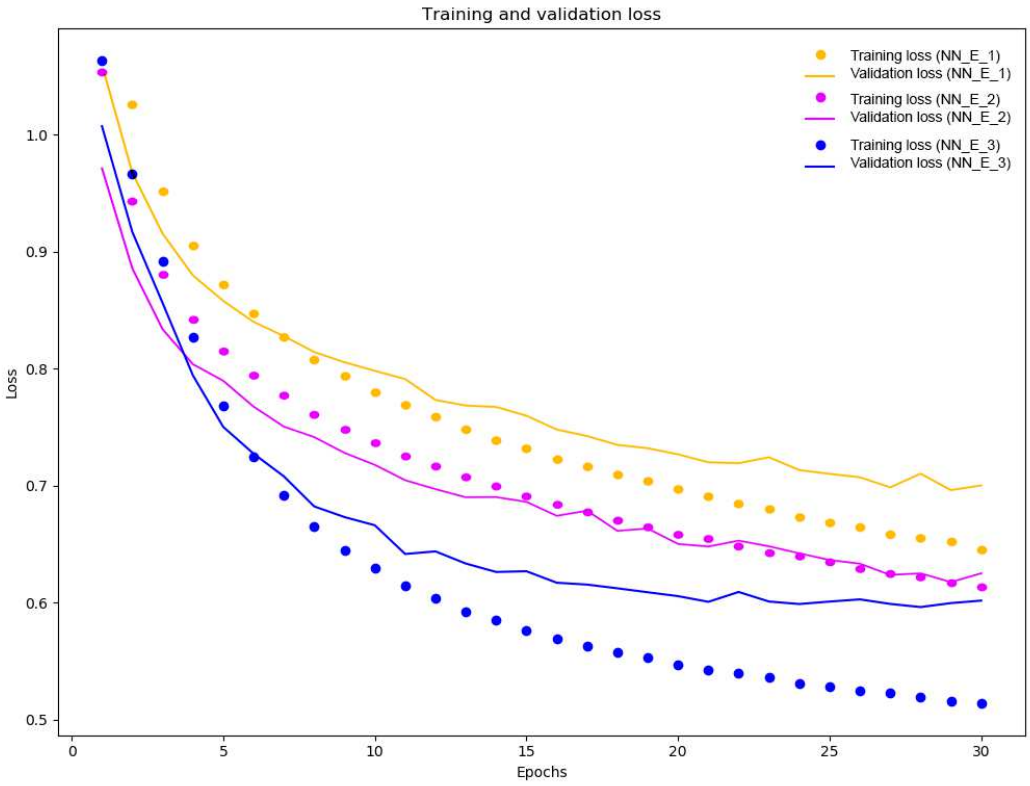


Figure 3.8: Comparison of the training and validation loss of the neural networks NN_E_1-3

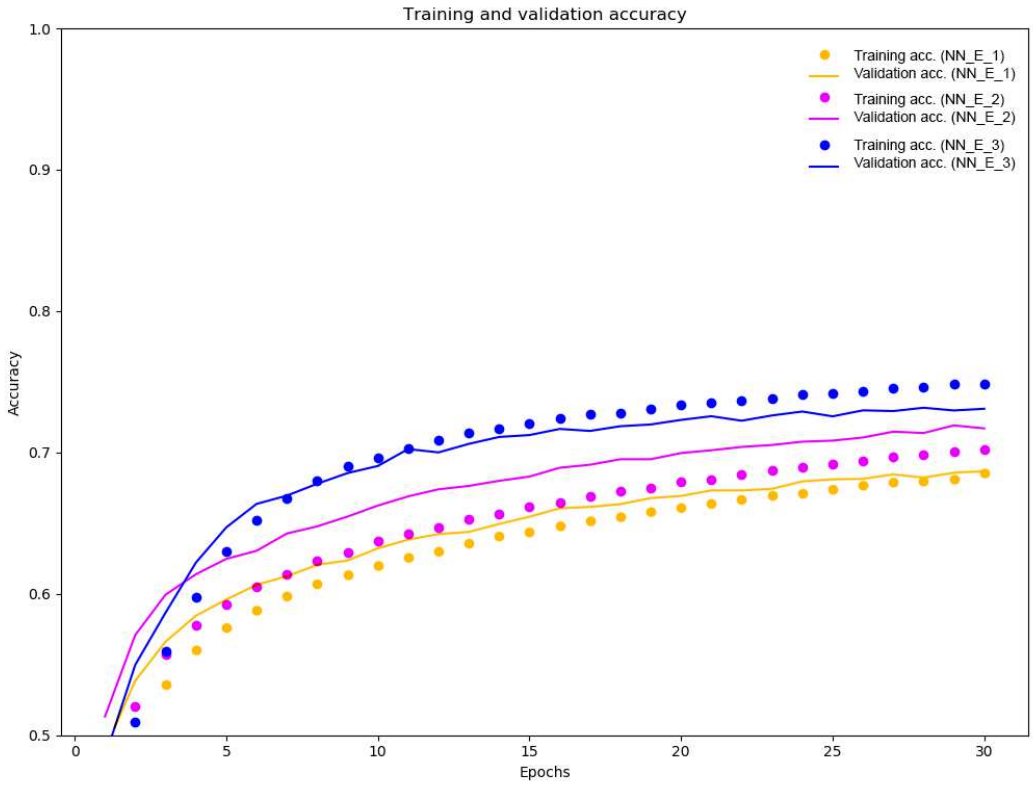


Figure 3.9: Comparison of the training and validation accuracy of the neural networks NN_E_1-3

3.3.2 Convolutional Neural Network

As described in Section 2.2.3.5, a refinement of the word embeddings was carried out to bring embeddings of same polarities closer together. CNN_1 uses refined word embeddings as an initialisation of the embedding matrix while CNN_2 uses embeddings that were directly derived from the word2vec model. The architectures of both models are shown in Figure 2.27.

Settings and parameters	**CNN_1**	**CNN_2**
Pretrained word vectors	Refined word2vec	word2vec
Stemming	no	no
Trainable word vectors	True	True
Return normalization	S&P 500	S&P 500
# input features	25,000	25,000
# filters	150	150
filter sizes	3, 4, 5	3, 4, 5
Learning rate	0.001	0.001
Beta	0.9	0.9

Table 3.10: Summary of the convolutional neural networks CNN_1 and CNN_2

Both models are trained over 9 epochs. The performance metrics are summarized in Table 3.11. It can be observed that CNN_1, which makes use of the refined word embeddings performs much better than CNN_2. The values for both, precision and recall increases for all classes. Furthermore, Figures 3.10 and 3.11 show the training and validation loss and accuracy after training over 9 epochs. The positive effect of the refined word vectors can be clearly seen immediately after the first epoch due to the offset between the two models.

	CNN_1			**CNN_2**		
Class	Precision	Recall	F1-score	Precision	Recall	F1-score
Neutral:	0.697	0.610	0.651	0.597	0.583	0.590
Negative:	0.736	0.799	0.766	0.706	0.735	0.720
Positive:	0.735	0.764	0.749	0.673	0.661	0.667
Accuracy			0.724			0.659
Macro avg. F_1	0.722	0.724	**0.722**	0.659	0.660	**0.659**

Table 3.11: Performance metrics of the CNNs on the validation dataset

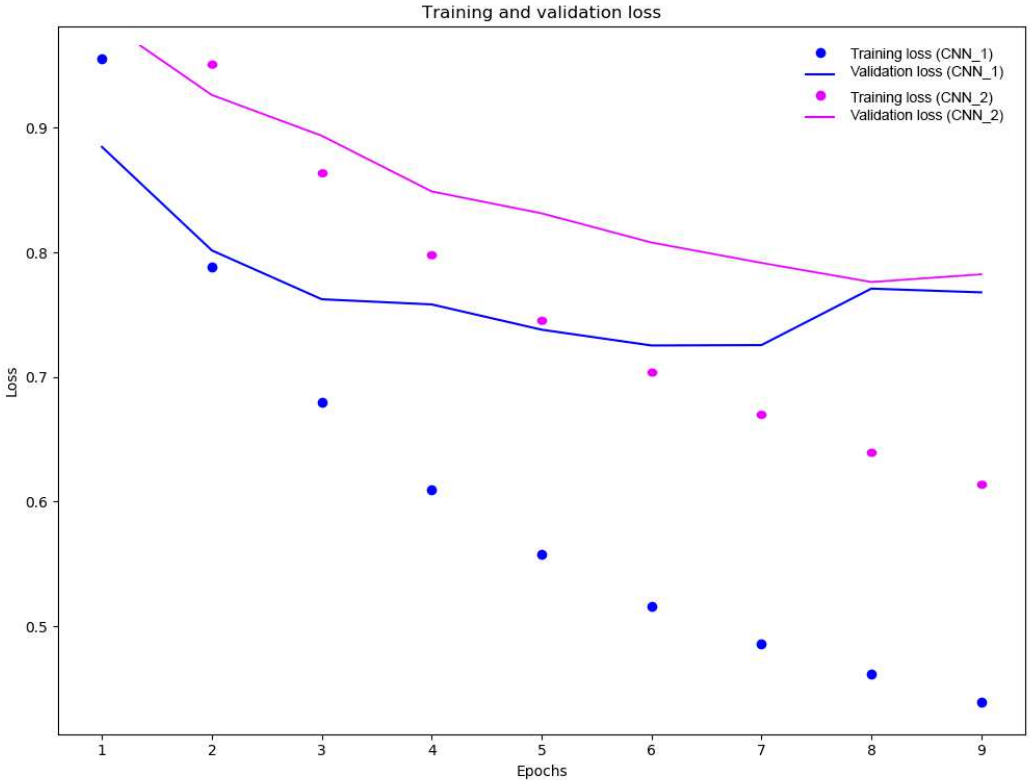


Figure 3.10: Comparison of the training and validation loss of the convolutional neural networks CNN_1 and CNN_2

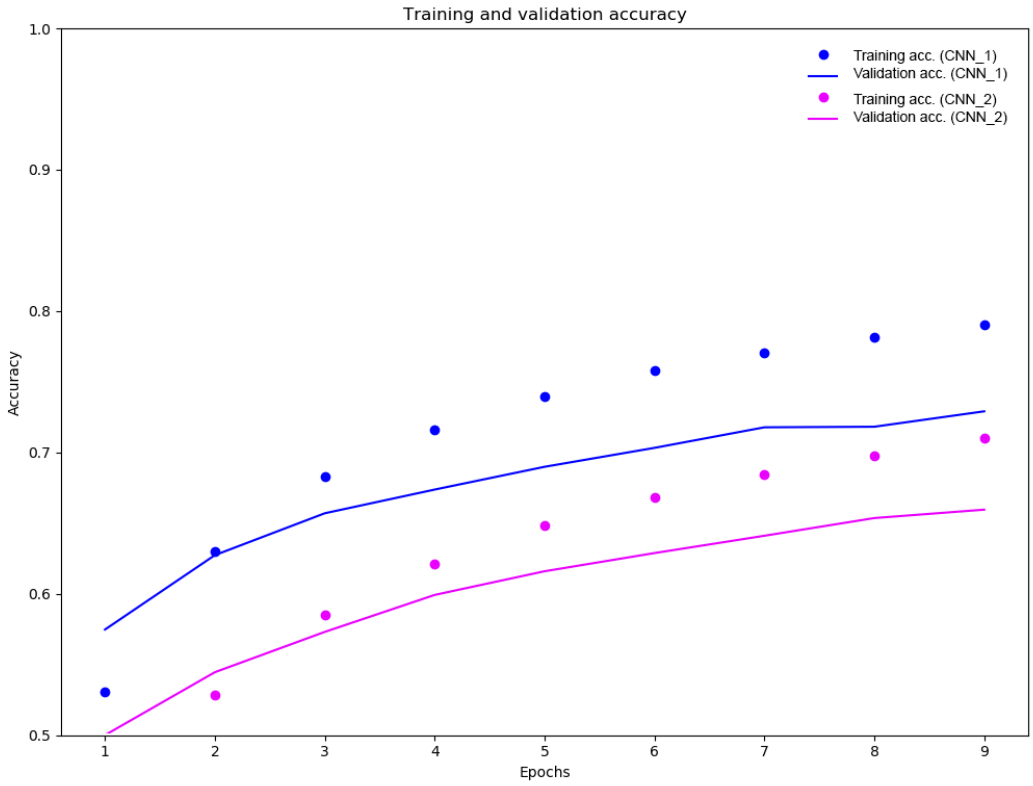


Figure 3.11: Comparison of the training and validation accuracy of the convolutional neural networks CNN_1 and CNN_2

3.4 Test Set Performance

The test set ranges from 01.01.2018 until 31.12.2019. In total, it consists of 9,057 positive, 38,405 neutral and 7,878 negative articles. After testing multiple models, the best performance on the test set was achieved with the neural network NN_E_2. Table 3.12 summarizes the performance metrics for two *min_confidence*[1] levels. The left side of the table contains all predictions since the *min_confidence* level is set to a minimum of 0.33. On the right side, the *min_confidence* level is set to 0.70 which means that only predictions that contain one class with a probability above or equal 0.70 are considered. As a consequence, the number of observations is reduced from 55,340 to 20,185 while the weighted accuracy increased from 0.626 to 0.683. Due to the fact, that the test dataset is strongly imbalanced, all other measures than the weighted average accuracy are misleading.

The actual predictions are shown in the confusion matrix 3.13 for a *min_confidence* level of 0.70. Furthermore, the confusion matrix shows that a large portion of positive and negative

[1]The minimum confidence level is a threshold value that determines whether a prediction is trustworthy or not. Each prediction of the model consists of three probabilities of the classes positive, neutral and negative, which sum up to 1. If the probability of one class exceeds the *min_confidence* level, the prediction is considered as relevant, otherwise it is discarded.

labelled articles are predicted as neutral. This may have several reasons. First, for the task of deriving sentiment labels, the arithmetic mean between the return on the publication date and the subsequent date was calculated which increased the number of neutral labelled news articles. Subsequently, the unbalanced training and validation data set was balanced by randomly duplicating positive and negative labelled articles. Since all neutral articles are unique, the model has more features to learn that predict a neutral article. This is probably the reason why the algorithm predicts so many articles on the test set as neutral. However, this does not affect the performance of the trading algorithm since it only considers news articles predicted as positive or negative. Consequently, the actual performance in the stock price prediction task can only be determined using the trading algorithm, which is discussed in the next Sections.

	NN_E_2 (min_confidence = 0.33)				**NN_E_2 (min_confidence = 0.70)**			
Class	Precision	Recall	F1-score	# sampels	Precision	Recall	F1-score	# sampels
Neutral:	0.740	0.822	0.779	38405	0.757	0.910	0.827	14057
Negative:	0.321	0.260	0.287	7878	0.525	0.293	0.376	2908
Positive:	0.334	0.231	0.273	9057	0.486	0.251	0.331	3220
Accuracy			0.645	55340			0.716	20185
Macro avg.	0.465	0.438	0.446	55340	0.589	0.484	0.511	20185
Weighted avg.	0.614	0.645	**0.626**	55340	0.680	0.716	**0.683**	20185

Table 3.12: Test set performance of the neural network NN_E_2 compared between min_confidence levels of 0.33 and 0.70

	Neutral	Negative	Positive
Predicted as neutral	12795	1851	2255
Predicted as negative	613	851	158
Predicted as positive	649	206	807
Total	14057	2908	3220

Table 3.13: Confusion matrix of the neural network NN_E_2 with a min_confidence level of 0.70

3.5 Performance Evaluation

3.6 Trading Algorithm

In order to evaluate the performance of different models in a real life scenario, a simple trading algorithm was developed. The trading strategy is similar to the one that is used by Kelly, Ke, and Xiu (2019). Each day, the algorithm buys stocks when news articles are predicted to be positive and shorts stocks when related news articles are predicted to be negative. The trading algorithm has three tuning parameters. One of them is the minimum confidence level which is a threshold value that determines whether a prediction is trustworthy or not. Each prediction of the model consists of three probabilities of the classes positive, neutral and negative, which sum up to 1. If the probability of one class exceeds the *min_confidence* level, the prediction is considered as relevant, otherwise it is discarded. For example, if the minimum confidence level is set to 0.70, an article with the predictions positive: 0.75, neutral: 0.20 and negative 0.05 is considered as positive. Otherwise, if the

predictions are: positive: 0.50, neutral: 0.30 and negative: 0.20, the article would not taken into account. This restriction gives more control over the trading scheme and reduces the risk of misclassification. The second parameter determines the maximum portfolio size. A large portfolio size means that companies with lower prediction probabilities are added to the portfolio while a smaller portfolio size restricts the portfolio to companies with higher prediction probabilities. In case there are only a few relevant news articles published, the number of portfolio constituents may be below the maximum portfolio size. In addition, the third parameter is the sentiment window. It determines the maximum number of days a stock is held in the portfolio after it has been added to the portfolio due to the news event. If the sentiment window is one day, than all stocks in the portfolio are sold after one day. In this thesis, the clock time of news publication is not considered. Therefore, the trading algorithm is limited in its ability to react quickly on published news. What is done, is that all news articles published on day t are analysed. Those companies related to the news with the highest sentiment scores are then traded on the day t+1. Depending on the sentiment window, stocks are hold for a minimum of one day in the portfolio. The algorithm always holds the companies with the highest prediction probabilities in the portfolio. This means that companies with low prediction probabilities can be exchanged with companies of higher probabilities even before the end of the sentiment window is reached.

3.6.1 Risk Adjusted Portfolio Benchmark

According to the Capital Asset Pricing Model (CAPM), the expected return of an investment is determined by its beta with the efficient market portfolio. Within CAPM, it is assumed that all investors act rationally and have homogeneous expectations. As a consequence, all investors will choose the same portfolio with the highest Sharpe ratio - the tangent portfolio. This further implies that the combined portfolio of all investors must also equal the tangent portfolio. Furthermore, due to the fact that each security is owed by an investor, the sum of all portfolios must equal the market portfolio. Consequently, the efficient tangent portfolio must equal the market portfolio (Berk and DeMarzo, 2014).
The expected return of an investment is calculated with Formula 3.9 and the factor β is calculated by Equation 3.10.

$$E^{\text{CAPM}}[R_i] = r_f + \beta_i * (E[R_m] - r_f) \tag{3.9}$$

with:

$E^{\text{CAPM}}[R_i]$	... expected return of asset i according to CAPM
$E[R_m]$	... expected return of the market portfolio
$E[R_m] - r_f$	... market risk premium
$\beta_i * (E[R_m] - r_f)$	... risk premium for security i
r_f	... risk free rate (2% p.a.)

$$\beta_i = \frac{Cov(R_i, R_m)}{Var(R_m)} \tag{3.10}$$

In real life scenarios, however, the market portfolio is not always efficient, as already argued in the introduction. One reason for this is that investors do not always act completely rationally and unbiased. The second reason is that new information about the financial markets can change the expected returns $E[R_i]$ of stocks and thus also their prices on the stock market. The objective of the trading algorithm is to spot those inefficiencies by analysing financial news. Figure 3.12 shows a market portfolio which is not in equilibrium. Suppose new information is published that raises the expected return of Exxon Mobil and GM and lowers the expected return of Anheuser-Busch and IBM for the case that stock prices remain unchanged. Consequently, the market portfolio is no longer efficient. Different portfolios than the market portfolio offer higher expected returns and lower volatilities (Berk and DeMarzo, 2014). The aim of the trading algorithm is to identify these inefficiencies and adjust the portfolio allocation accordingly.

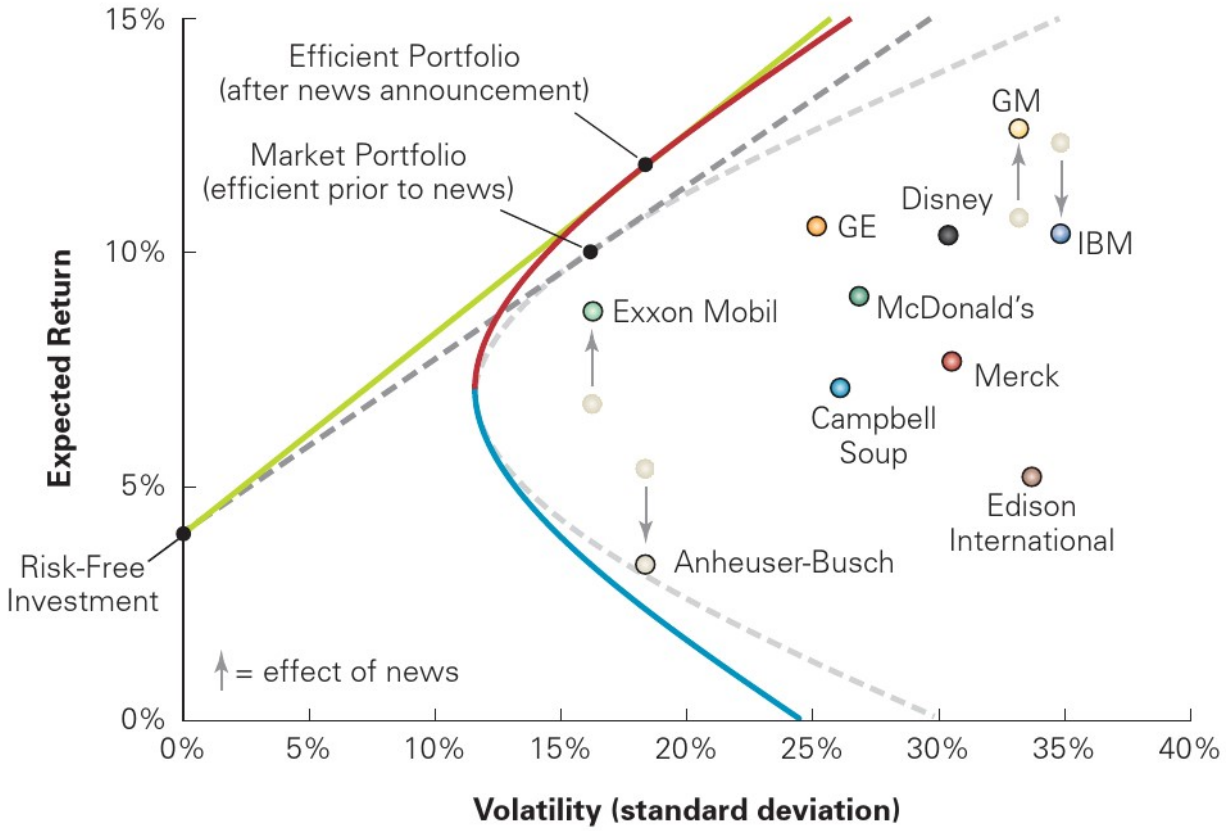


Figure 3.12: After the announcement of financial news, the market portfolio is not equal to the efficient portfolio. Thus, the market portfolio is not in the CAPM equilibrium (Berk and DeMarzo, 2014).

The deviation of the stock's expected return from the return required by the CAPM (which is on the security market line (SML), see Figure 3.13) is the stock's alpha (α_i):

$$\alpha_i = E[R_i] - E^{\text{CAPM}}[R_i] \tag{3.11}$$

with:

$E[R_i]$ ... expected return of asset i

In the case of an efficient stock market, all securities are on the security market line and thus have an alpha of zero. When the stock market is not efficient, investors can profit by buying stocks with positive alphas and selling stocks with negative alphas (Berk and DeMarzo, 2014). This strategy is implemented by the trading algorithm.

The hypothesis is that the proposed trading algorithm is able to detect market inefficiencies and generate positive alpha by making trading decisions based on financial news. Therefore it buys stocks based on positive news, and sells stocks with negative news.
A confirmation of this hypothesis would consequently provide evidence that the financial news from Thomson Reuters contains predictive information about future stock returns.

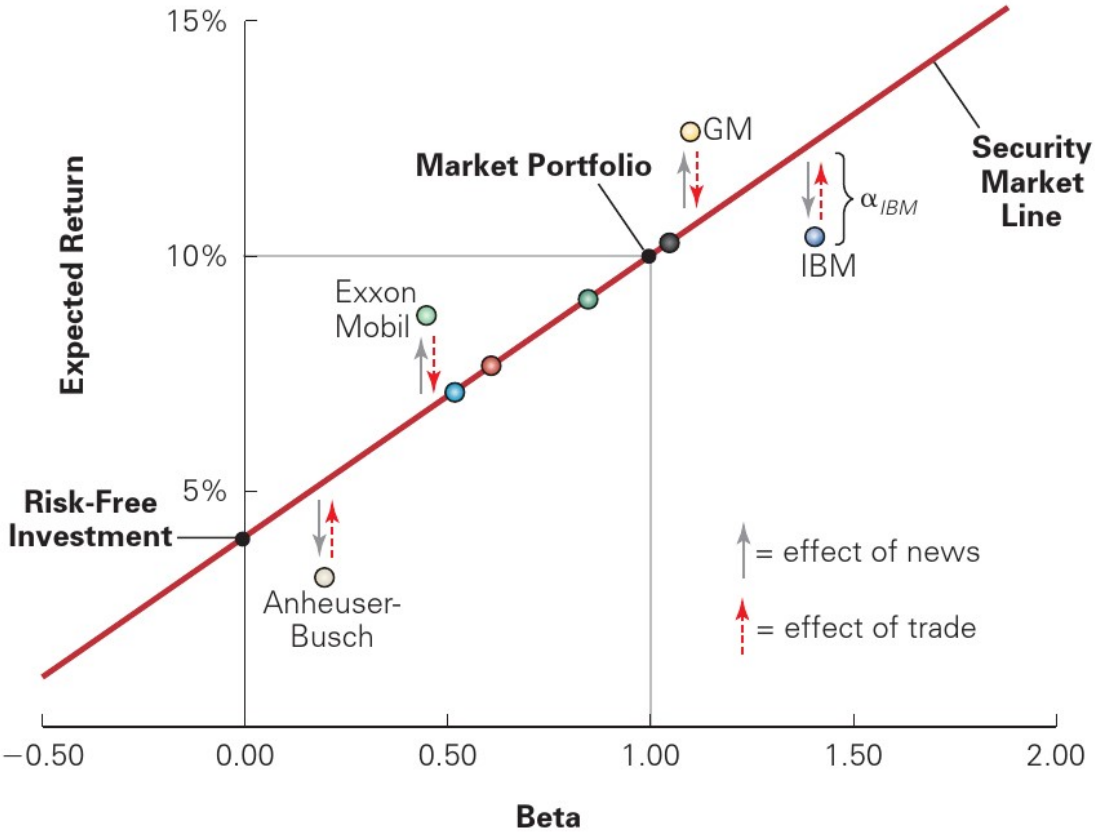


Figure 3.13: In an efficient market, all risk adjusted securities lie on the security market line (SML). However, if the market is not efficient securities deviate from the SML. The distance between the SML and an individual stock is the stock's alpha (Berk and DeMarzo, 2014).

In order to determine the performance of the trading algorithm, the CAPM-implied expected conditional return $x_{b(t)}$ is taken as a benchmark. Therefore, equation 3.9 can be written as:

$$E[R_P] - r_f = \beta_P * (E[R_m] - r_f) \tag{3.12}$$

To calculate the risk adjusted outperformance $y_{(t)}$ the CAPM-implied expected conditional return $x_{b(t)}$ and the excess portfolio return $x_{p(t)}$ need to be calculated in each time step t.

$$x_{b(t)} = \beta_{p(t-1)} * (r_{m(t)} - r_f) \tag{3.13}$$

$$x_{p(t)} = r_{p(t)} - r_f \tag{3.14}$$

with:

$x_{b(t)}$	... CAPM-implied expected conditional excess return at time step t (risk adjusted benchmark)
$x_{p(t)}$	... excess portfolio return at time step t
$r_{p(t)}$	... portfolio return at time step t
$r_{m(t)}$	... return of the market portfolio at time step t
$\beta_{p(t-1)}$	... portfolio beta at time step t-1

Since the portfolio allocation changes in every time step, the portfolio beta needs to be calculated on a daily basis with Formula 3.15. Furthermore, the covariance matrix $Cov(R_i, R_m)$, as well as the variance $Var(R_m)$ is calculated over a 2 year rolling window. The market return is the value weighted S&P 500 total return index.

$$\beta_{p(t)} = \sum w_{i(t)} * \beta_{i(t)} \tag{3.15}$$

with:

$w_{i(t)}$	... weight of the security i at time step t

The difference of the excess portfolio return and the risk adjusted benchmark return must be zero if the market portfolio is in the CAPM equilibrium. However, if the market is not efficient, the risk adjusted outperformance $y_{(t)}$ is defined as:

$$y_{(t)} = x_{p(t)} - x_{b(t)} = \alpha_{(t)} + \varepsilon_{(t)} \tag{3.16}$$

The resulting outperformance need to be tested for statistical significance. Therefore, a t-test (3.17) is conducted with the null hypothesis:
The mean value of alpha is equal to the mean value of the error term ε which equals zero.

$$t = \frac{\bar{y_T} - \varepsilon_T}{\frac{\sigma_T}{\sqrt{n}}} \tag{3.17}$$

with:

$\bar{y_T}$	... mean of the risk adjusted outperformance at time step T over the window of size n
σ_T	... standard deviation evaluated at time step T over the window of size n
n	... number of observations

Table 3.14 shows the two sided quantiles of the standard normal distribution. The resulting t-values are shown in Table 3.15 for different settings. These t-values are calculated over a window of size n, where n is equal to the total number of trading days in the two-year test period.

P	0.9	0.95	0.98	0.99	0.995	0.998	0.999
z	1.645	1.960	2.326	2.576	2.807	3.090	3.291

Table 3.14: Quantiles of the standard normal distribution

3.6.2 Performance of the Trading Algorithm

The objective of the trading algorithm is to maximize the risk adjusted portfolio return which is measured by the Sharpe ratio. The initial investment sum is set to 100,000 USD. Furthermore, the portfolio development is observed over a timespan of two years, ranging from 01.01.2018 until 31.12.2019. Multiple tests with many different models and settings were conducted. As a result, the trading algorithm achieves the best performance with the neural network NN_E_2. Figure 3.14 shows the performance of the excess portfolio return in comparison to the risk adjusted benchmark return. Each day, the trading algorithm forms a new portfolio based on the predictions of the trained model. It buys stocks with positive predicted sentiment and sells stocks with negative predicted sentiment. Thus, the portfolios are held for one day. As a result, it can be seen that the excess portfolio return of the trading algorithm outperforms the risk adjusted portfolio benchmark. The mean of the risk adjusted outperformance α over the two year period is 5.17 bps per day and the corresponding t-value is 1.973. (see Table 3.15). Thus, the null hypothesis can be rejected with a significance level of 5%. Furthermore, the second trading strategy is the long only strategy. The corresponding excess portfolio return is shown in Figure 3.15. As a result, this strategy achieves a risk adjusted outperformance α of 12.35 bps per day with a t-value of 3.478. Thus, these results are significant on a 0.1% level.

Moreover, the total performance (including the risk-free rate) of the trading algorithm is illustrated in Figure 3.16. The plot shows the performance of the long & short trading strategy as well as the long only trading strategy in comparison to the value weighted and equally weighted total return market portfolio. It can be observed that the long portfolio clearly outperforms all other portfolios. However, it turns out that the model has a weakness in predicting true negatives. Therefore, it performs badly compared to the long portfolio if there is an upward trend in the market. The two year return of the long portfolio is 103.20% while the long/short portfolio has a return of 36.19% and the value weighted S&P 500 shows a two year return of 18.73%. The standard deviations are 24.81%, 14.40% and 21.02% and the Sharpe ratios are 2.827, 1.943 and 0.57 respectively. Thus, a risk averse investor would

select the long/short strategy while a more risk seeking investor would choose the long portfolio.

The disadvantage of this dynamic trading strategies are the high transaction costs which are not considered in the simulations. Since the portfolios change every day, the daily asset turnover is close to 100%. In average, the long portfolio comprises of 6.565 constituents while the long & short portfolio consists of 10.13 assets. The portfolio allocation of both strategies over the full period is shown in Figure 3.17. Therefore, these trading strategies are only applicable for institutional investors or investors who pay low transaction costs.

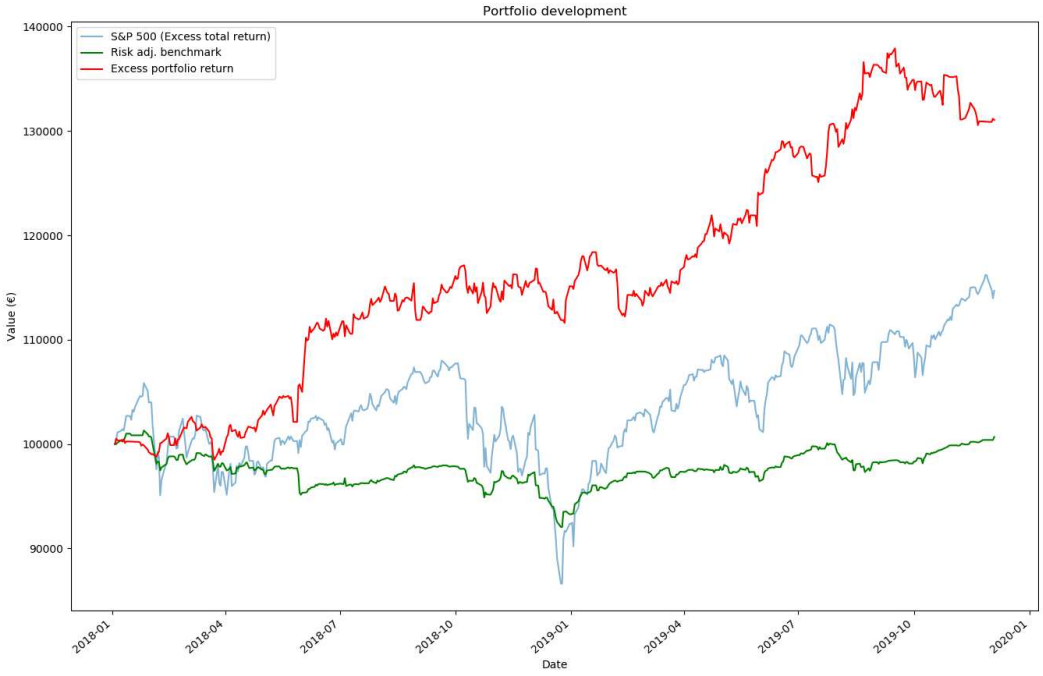


Figure 3.14: Excess portfolio return of the long& short trading strategy with the model NN_E_2 compared to the risk adjusted benchmark return and the S&P 500 excess total return.

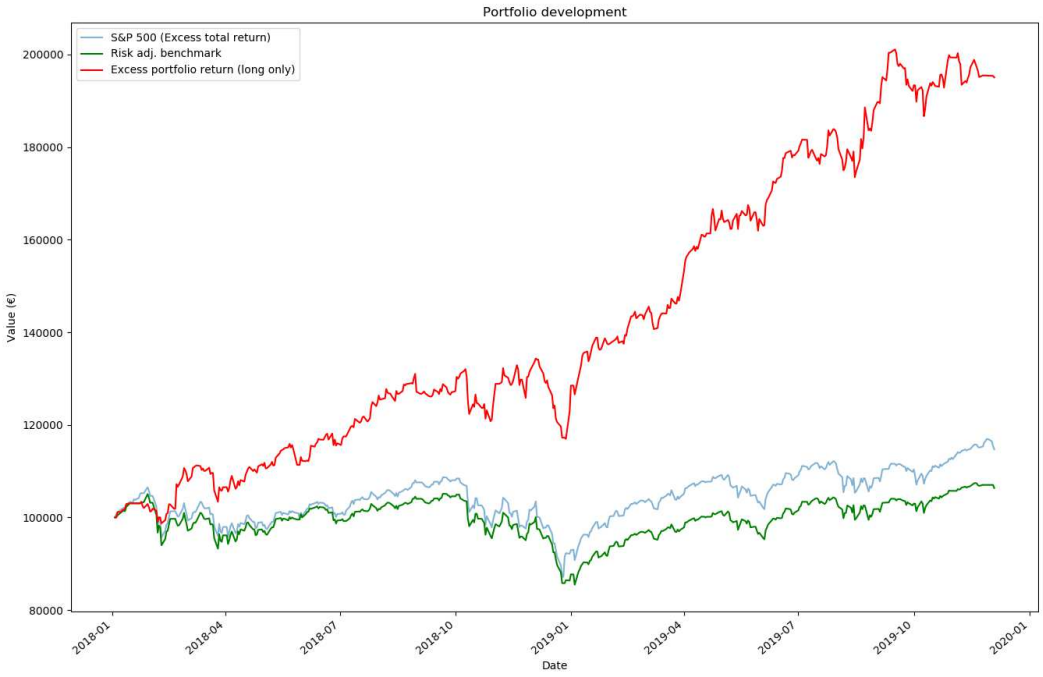


Figure 3.15: Excess portfolio return of the long only trading strategy with the model NN_E_2 compared to the risk adjusted benchmark return and the S&P 500 excess total return.

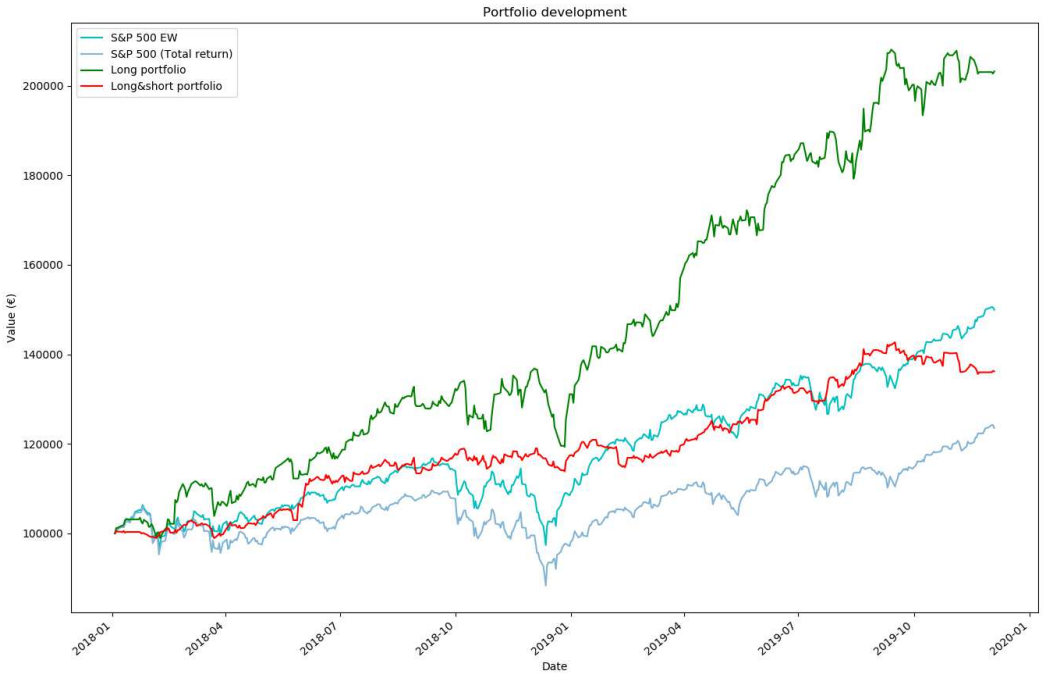


Figure 3.16: Portfolio development of the long portfolio and the long/short portfolio compared to the equally weighted total return market portfolio and the value weighted total return market portfolio of the trading algorithm with model NN_E_2.

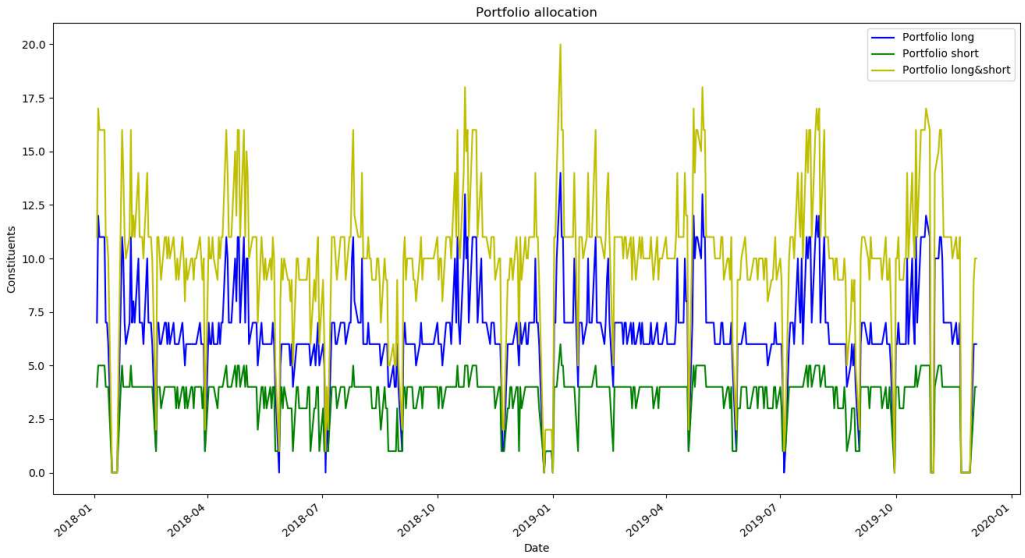


Figure 3.17: Portfolio allocation of the long/short and the long only trading strategies of the trading algorithm with model NN_E_2.

Additionally, the values of alpha and beta are plotted over the full period for the long & short strategy in Figure 3.18. Furthermore, Figure 3.19 shows the t-values of both trading strategies for a 60 day rolling window and the entire 2 year period. It can be observed, that the t-values are low after the market drops in the winter of 2018 and also in the the end of 2019. In those phases both strategies are not able to outperform the risk adjusted benchmark which is why α and therefore also the t-values are low at this time.

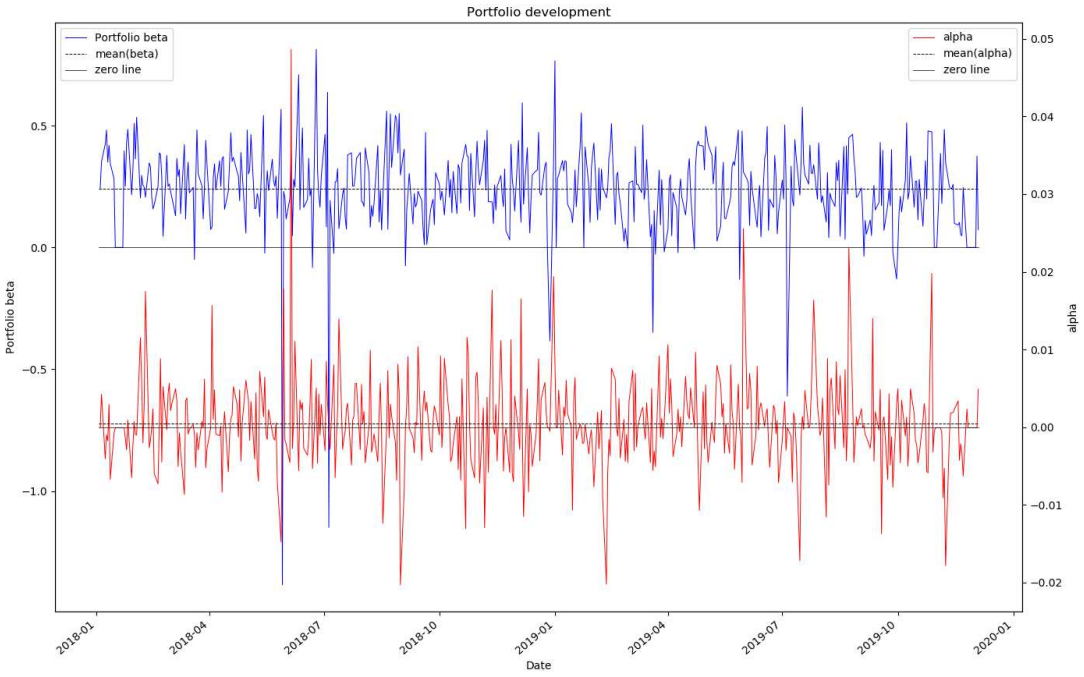


Figure 3.18: The portfolio beta and alpha received by the trading algorithm with model NN_E_2 and the long & short investment strategy

Figure 3.19: T-values of the long & short and the long only trading strategies over a 60 day rolling window and the entire period

3.6.3 Influence of the Depreciation of Information

Below is a comparison of the trading algorithm performance for different lengths of the sentiment window. The parameter *sentiment_window* determines the time interval of news over which the sentiment is computed. I.e., if the *sentiment_window* is set to 5 days, the sentiment at a given point in time is generated from news published over the past 5 days. In addition, the assets are ranked according to their prediction probabilities and the maximum portfolio size is limited to 20 assets. So if a news article causes a prediction with a high level of confidence, and the portfolio already consists of 20 assets, the new asset will replace the asset in the portfolio with the lowest prediction confidence. Table 3.15 gives an overview of the trading algorithm performance of different sentiment window sizes. It can be seen that the Sharpe ratio declines with an increasing window size. These results are consistent with the findings of Kelly, Ke, and Xiu (2019) who showed that it takes one day for large stocks to fully reflect new information. As a consequence, no value is created by considering news that is older than three days. However, the asset turnover decreases with larger window sizes, which leads to lower transaction costs. In addition, the standard deviation also decreases with larger window sizes due to the higher number of constituents in the portfolio. Lastly, the decision for the best trading strategy depends on the risk aversion of the investor and the transaction costs. Figures 3.20 and 3.21 show the excess portfolio return of the trading algorithm for sentiment window sizes of two and three days.

Sentiment window	1 day		2 days		3 days		S&P 500 total return index
Trading strategy	long/short	long	long/short	long	long/short	long	
Max. portfolio size	20	20	20	20	20	20	
min_conficence level	0.70	0.70	0.70	0.70	0.70	0.70	
Avg. portfolio size	10.13	6.57	17.90	12.74	19.14	17.21	
2 year performance	36.19 %	103.20 %	16.73 %	73.02 %	16.39 %	52.62 %	18.73 %
Sharpe ratio	1.943	2.827	1.134	2.432	1.227	1.886	0.570
StdDev	14.40 %	24.81 %	10.63 %	21.94 %	9.49 %	21.58 %	21.02 %
Alpha (risk adj. outperformance)	5.17 bps/d	12.35 bps/d	2.44 bps/d	7.99 bps/d	1.78 bps/d	5.54 bps/d	
t-value (2 year mean)	1.973	3.478	1.245	3.401	1.005	2.886	
Beta	0.240	0.903	0.201	0.920	0.124	0.927	
Daily asset turnover	96.73%	96.44%	48.25%	47.51%	33.67%	32.03%	
Trade count	4909	3172	4326	3032	3229	2762	

Table 3.15: Performance metrics of the trading algorithm with the neural network NN_E_2 for different sentiment window sizes.

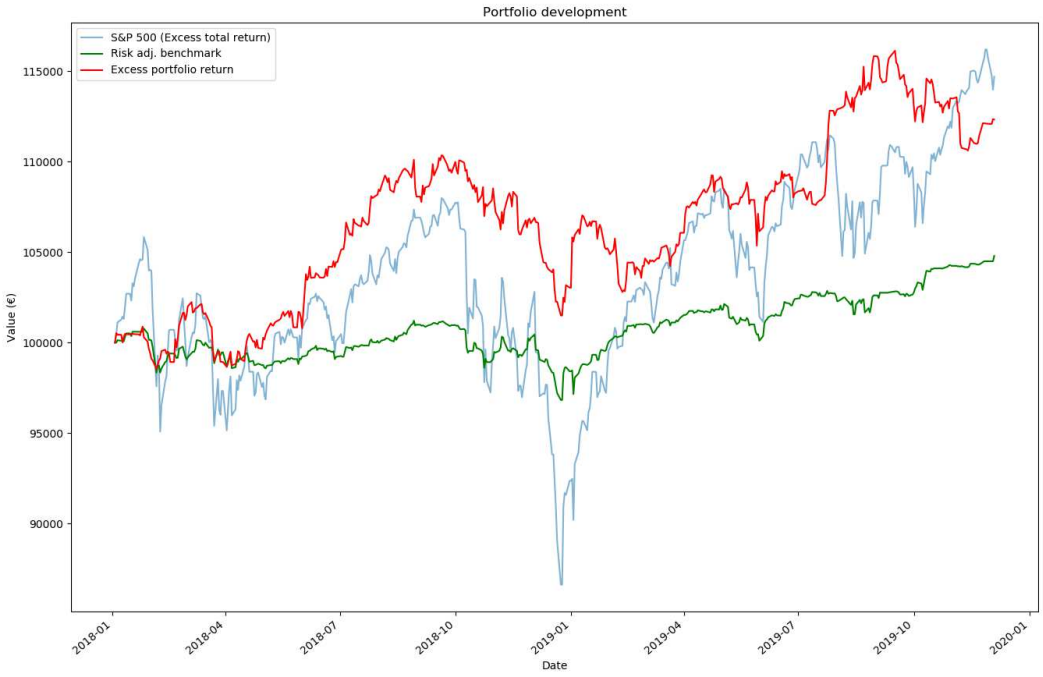


Figure 3.20: Excess portfolio return of the trading algorithm with the model NN_E_2 compared to the risk adjusted benchmark return and the S&P 500 excess total return for a sentiment window of two days.

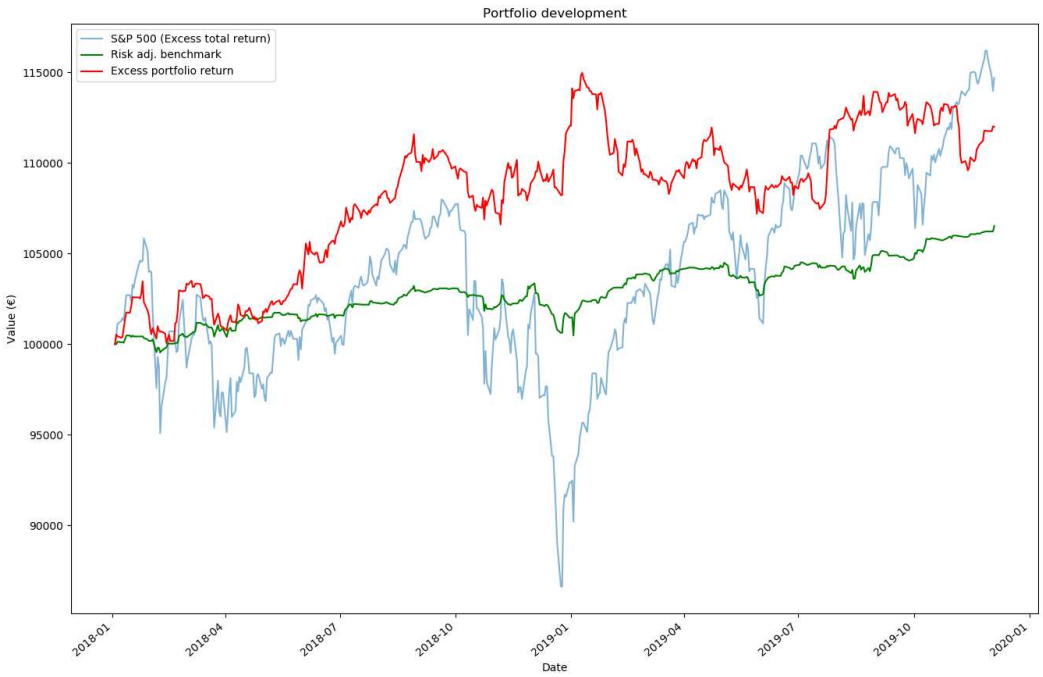


Figure 3.21: Excess portfolio return of the trading algorithm with the model NN_E_2 compared to the risk adjusted benchmark return and the S&P 500 excess total return for a sentiment window of three days.

Chapter 4

Conclusion

The aim of this thesis is to find an answer to the research question whether it is possible to implement an event-driven trading strategy that benefits from an inefficient stock market and generates a positive alpha by using financial news as a predictor for stock price movements. The results show that the implemented trading algorithm is able to benefit from the market inefficiencies caused by financial news. These results indirectly answer the second research question, whether financial news contains predictive information about future stock returns. As the trading algorithm is able to generate a significant positive alpha over the entire two-year evaluation period, it can be concluded that financial news contains predictive information about future stock returns. In addition, the implemented trading strategies were also able to significantly outperform the risk-adjusted benchmark. The long-only strategy achieved a Sharpe ratio of 2.827 over the two year testing period which is almost five times as high as the S&P 500 Sharpe ratio of 0.570. Furthermore, the long & short strategy of the trading algorithm achieved a Sharpe ratio of 1.943. This strategy performed well in declining market phases, but lagged behind the performance of the long portfolio in rising market phases. Additionally, the trading algorithm confirms the findings made by Kelly, Ke, and Xiu (2019) who states that new information is fully reflected in the share prices within one day for large stocks. For a sentiment window of one day, the algorithm achieves the best performance. The larger the sentiment window, the weaker the performance, as Table 3.15 shows.

Among the various models and feature representations tested, the neural network NN_E_2 achieved the best results on the validation and test sets as well as for the task of stock price prediction with the trading algorithm. It has three hidden layers with 256 nodes in each hidden layer and takes word embeddings as input features. After training over 30 epochs the model learns the sentiment polarity of the word embeddings which further increases performance. Moreover, the training of different models shows that all models tend to overfit due to the high noise in the test data. Limiting the number of input features to 25,000 and using dropout mitigates the problem of overfitting. In addition, although stemming increases the F_1-score, precision does not improve. Subsequently, no performance gains of the trading algorithm can be achieved with stemming. Despite implementing different models with different sizes and architectures, the resulting F_1-scores are all in the range of 0.70 ± 0.03. A closer look reveals that precision and recall differs much more between the different models. Therefore, the F_1-score is a relatively weak measure for the actual model performance. For the purpose of this thesis, precision is of much greater importance.

Chapter 5

Outlook

Today, we live in exciting times with rapid technological progress. In the last decade, researchers in the field of machine learning and artificial intelligence had many breakthroughs due to the increasing computational power and the vast available amount of data. I am personally convinced that this progress will further accelerate. Especially advances in the field of natural language understanding (NLU) has a high potential to boost the ability of stock price prediction by analysing textual data.
The most powerful network architectures that exist today are large Transformer networks that have billions of parameters. Those networks are trained on very large datasets by the most powerful supercomputers. One popular model in this category is called BERT, which stands for Bidirectional Encoder Representations from Transformers and was developed by Google in 2018 (Devlin et al., 2018).
Using the parameters of those large pre-trained networks and applying it to the dataset used in this thesis is likely to further improve the stock price predictability. Furthermore, reinforcement learning algorithms are also a powerful tool in finance to create advanced trading systems. Using those intelligent algorithms and combining it with different predictive signals as it was done by Gu, Kelly, and Xiu (2020) and with textual analysis of financial news and reports may result in very powerful models for asset management.

Furthermore, since today's financial institutions employ computers for trading on a large scale, prices in liquid markets react quickly to new information. Using data of shorter time intervals (e.g. hourly or minutely intervals) would enable the algorithm to react more quickly to financial news and consequently generate a higher alpha.

Bibliography

Adventures in Machine Learning, ed. (2018). *The vanishing gradient problem and ReLUs - a TensorFlow investigation*. URL: https://adventuresinmachinelearning.com/vanishing-gradient-problem-tensorflow/ (visited on 03/28/2020).

Antweiler, Werner and Murray Z. Frank (2004). "Is all that talk just noise? The information content of internet stock message boards". In: *The Journal of Finance* 59.3, pp. 1259–1294. ISSN: 00221082.

Baker, Malcolm and Jeffrey Wurgler (2007). "Investor Sentiment in the Stock Market". In: *Journal of Economic Perspectives* 21.2, pp. 129–152. ISSN: 0895-3309. DOI: 10.1257/jep.21.2.129. URL: https://www.aeaweb.org/articles?id=10.1257/jep.21.2.129.

Berk, Jonathan B. and Peter M. DeMarzo (2014). *Corporate finance*. Third edition. The Pearson series in finance. Boston, Mass.: Pearson. xxxii, 1104 páginas. ISBN: 9780132992473.

Bollen, Johan and Huina Mao (2011). "Twitter Mood as a Stock Market Predictor". In: *Computer* 44.10, pp. 91–94. ISSN: 0018-9162. DOI: 10.1109/MC.2011.323.

Cambria, Erik et al. (2017). "Sentiment Analysis Is a Big Suitcase". In: *IEEE Computer Society*.

Chollet, Francois (2016). *Using pre-trained word embeddings in a Keras model*. Keras. URL: https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html (visited on 03/17/2020).

Chung, Yu-An, Hung-Yi Lee, and James Glass (2017). *Supervised and Unsupervised Transfer Learning for Question Answering*. To appear in NAACL HLT 2018 (long paper). URL: http://arxiv.org/pdf/1711.05345v3.

David L. Olson, Dursun Delen (2008). *Advanced Data Mining Techniques*. 1st ed. Springer. ISBN: 3540769161.

DeepAI (2019). *Feature Extraction*. DeepAI. URL: https://deepai.org/machine-learning-glossary-and-terms/feature-extraction (visited on 05/12/2020).

— (2020). *Harmonic Mean Definition*. URL: https://deepai.org/machine-learning-glossary-and-terms/harmonic-mean (visited on 03/16/2020).

Devlin, Jacob et al. (2018). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. URL: http://arxiv.org/pdf/1810.04805v2.

Ding, Xiao et al. (2015). "Deep Learning for Event-Driven Stock Prediction". en. In:

Downey, Lucas (Nov. 18, 2003). "Efficient Market Hypothesis (EMH)". In: *Investopedia*. URL: https://www.investopedia.com/terms/e/efficientmarkethypothesis.asp (visited on 04/17/2020).

Elite Data Science, ed. (2017). *How to Handle Imbalanced Classes in Machine Learning*. URL: https://elitedatascience.com/imbalanced-classes (visited on 03/26/2020).

Gentzkow, Matthew, Bryan T. Kelly, and Matt Taddy (2017). "Text as Data". In: *National Bureau of Economic Research*. URL: https://www.nber.org/papers/w23276.pdf.

Géron, Aurélien (2017). *Hands-On Machine Learning with Scikit-Learn and Tensorflow. Concepts, tools and techniques to build intelligent systems*. O'Reilly Media, Inc.

Gu, Shihao, Bryan T. Kelly, and Dacheng Xiu (2020). "Empirical Asset Pricing via Machine Learning". In: *The Review of Financial Studies* 33.5, pp. 2223–2273. ISSN: 0893-9454. DOI: 10.1093/rfs/hhaa009. URL: https://academic.oup.com/rfs/article/33/5/2223/5758276.

Hearst, Marti (2009). *What Is Text Mining?* URL: http://people.ischool.berkeley.edu/~hearst/text-mining.html (visited on 03/11/2020).

Herschberg, Miguel (2012). "Limits to arbitrage: an introduction to behavioral finance and a literature review". In:

Ioffe, Sergey and Christian Szegedy (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. URL: https://arxiv.org/pdf/1502.03167.

Jegadeesh, Narasimhan and Di Wu (2013). "Word power: A new approach for content analysis". In: *Journal of Financial Economics* 110.3. PII: S0304405X13002328, pp. 712–729. ISSN: 0304405X. DOI: 10.1016/j.jfineco.2013.08.018.

Junxi Feng et al. (2019). "Reconstruction of porous media from extremely limited information using conditional generative adversarial networks". In: *PHYSICAL REVIEW E* 3. ISSN: 2470-0045. DOI: 10.1103/PhysRevE.100.033308. eprint: 31639909.

Kathuria, Ayoosh (2018). "Intro to optimization in deep learning: Gradient Descent". In: *Paperspace Blog*. URL: https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/ (visited on 03/29/2020).

Kelly, Bryan T., Zheng Tracy Ke, and Dacheng Xiu (2019). "Predicting Returns with Text Data". In: *Yale ICF Working Paper* 2019-10, p. 54.

Kenton, Will (2003). "Behavioral Finance". In: *Investopedia*. URL: https://www.investopedia.com/terms/b/behavioralfinance.asp (visited on 04/17/2020).

Kim, Ricky (2017). "Another Twitter sentiment analysis with Python — Part 3 (Zipf's Law, data visualisation)". In: *Towards Data Science*. URL: https://towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-3-zipfs-law-data-visualisation-fc9eadda71e7 (visited on 03/14/2020).

Kingma, Diederik P. and Jimmy Ba (2014). *Adam: A Method for Stochastic Optimization*. Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. URL: http://arxiv.org/pdf/1412.6980v9.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". eng. In: *Nature* 521.7553. Journal Article Research Support, Non-U.S. Gov't Research Support, U.S. Gov't, Non-P.H.S. Review, pp. 436–444. DOI: 10.1038/nature14539. eprint: 26017442.

Lee, Heeyoung et al. (2014). "On the Importance of Text Analysis for Stock Price Prediction". In: *LREC*. Vol. 2014, pp. 1170–1175.

Loughran, Tim and Bill McDonald (2011). "When is a liability not a liability? Textual analysis, dictionaries, and 10-Ks". In: *The Journal of Finance* 66.1, pp. 35–65. ISSN: 00221082.

Malkiel, Burton G. and Eugene F. Fama (1970). "Efficient capital markets: A review of theory and empirical work". In: *The Journal of Finance* 25.2, pp. 383–417. ISSN: 00221082.

Manning, Christopher and Richard Socher (2017). *Natural Language Processing with Deep Learning. Lecture 2: Word Vectors*. Ed. by Stanford University. URL: https://www.youtube.com/watch?v=ERibwqs9p38 (visited on 03/18/2020).

Md Tayeen, Abu Saleh et al. (2019). "Comparison of Text Mining Feature Extraction Methods Using Moderated vs Non-Moderated Blogs". In: *Proceedings of the 9th International*

Conference on Digital Public Health - DPH2019. the 9th International Conference (Marseille, France). Ed. by Patty Kostkova et al. New York, New York, USA: ACM Press, pp. 69–78. ISBN: 9781450372084. DOI: 10.1145/3357729.3357740.

Mikolov, Tomas et al. (2013). *Efficient Estimation of Word Representations in Vector Space*. URL: http://arxiv.org/pdf/1301.3781v3.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom (2014). "A Convolutional Neural Network for Modelling Sentences". In:

Nitish Srivastava et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56, pp. 1929–1958.

Oppermann, Artem (2019). *Activation Functions in Neural Networks*. Ed. by DeepLearning Academy. URL: https://www.deeplearning-academy.com/p/ai-wiki-activation-functions (visited on 03/28/2020).

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Petrov, Christo (2019). *16+ Big Data Statistics - The Information We Generate [2020]*. URL: https://techjury.net/stats-about/big-data-statistics/#gref (visited on 04/27/2020).

Powers David M, W (2008). *Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation*. Vol. 2.

Ramos, Juan, ed. (2003). *Proceedings of the first instructional conference on machine learning. Using tf-idf to determine word relevance in document queries*. 242nd ed.

Rechenthin, Michael, W. Nick Street, and Padmini Srinivasan (2013). "Stock chatter: Using stock sentiment to predict price direction". In: *Algorithmic Finance* 2.3-4, pp. 169–196. ISSN: 21585571. DOI: 10.3233/AF-13025.

Severyn, Aliaksei and Alessandro Moschitti (2015). "Twitter Sentiment Analysis with Deep Convolutional Neural Networks". In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '15*. the 38th International ACM SIGIR Conference (Santiago, Chile). Ed. by Ricardo Baeza-Yates et al. New York, New York, USA: ACM Press, pp. 959–962. ISBN: 9781450336215. DOI: 10.1145/2766462.2767830.

Starmer, Josh (2018). *Machine Learning Fundamentals: The Confusion Matrix - YouTube*. URL: https://www.youtube.com/watch?v=Kdsp6soqA7o (visited on 06/10/2020).

Statman, Meir (1995). "Behavioral finance versus standard finance". In: *Behavioral Finance and Decision Theory in Investment Management*, pp. 14–22.

Sullivan, Dan (2001). *Document warehousing and text mining: techniques for improving business operations, marketing, and sales*. John Wiley & Sons, Inc.

Tang, D. et al. (2016). "Sentiment Embeddings with Applications to Sentiment Analysis". In: *IEEE Transactions on Knowledge and Data Engineering* 28.2, pp. 496–509. ISSN: 1558-2191. DOI: 10.1109/TKDE.2015.2489653.

Tetlock, Paul C. (2007). "Giving Content to Investor Sentiment: The Role of Media in the Stock Market". In: *The Journal of Finance* 62.3, pp. 1139–1168. ISSN: 00221082. DOI: 10.1111/j.1540-6261.2007.01232.x.

Thaler, Richard H. (1999). "The end of behavioral finance". In: *Financial Analysts Journal* 55.6, pp. 12–17.

Tomar, Ankur (July 30, 2019). "A math-first explanation of Word2Vec". In: *Analytics Vidhya*. URL: https://medium.com/analytics-vidhya/maths-behind-word2vec-explained-38d74f32726b (visited on 03/18/2020).

Vargas, M. R., B. S. L. P. Lima, and A. G. Evsukoff (2017). "Deep learning for stock market prediction from financial news articles". In: *2017 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)*. 2017 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA), pp. 60–65. ISBN: 2377-9322. DOI: 10.1109/CIVEMSA.2017.7995302.

Yih, W. et al. (2011). "Learning discriminative projections for text similarity measures". In: pp. 247–256.