



Large Language Model-driven Sentiment Analysis for Exploring the Influence of Social Media and Financial News on Stock Market Development

Master Thesis

Master of Science in International Software System Science

Pascal Cezanne

December 1, 2024

Supervisor:

1st: Prof. Dr. Christian Ledig

2nd: Sebastian Dörrich, M.Sc.

Chair of Explainable Machine Learning

Faculty of Information Systems and Applied Computer Sciences

Otto-Friedrich-University Bamberg

Abstract

News about financial events is often a driving factor for stock prices. Therefore, leveraging Large Language Models (LLMs) to create a profitable trading strategy appears promising. However, traditional sentiment analysis approaches fall short when applied to stock market predictions. This is due to the complexity of the market, which is influenced by various factors, and news having contradictory outcomes depending on the involved companies.

This research addresses the critical gap between textual sentiment and actual market movement by introducing an innovative approach to predicting market sentiment. The study involves crawling a dataset containing financial news titles from various publishers. Instead of relying on human-annotated labeling, the research focuses on historical price data to connect news events with price developments. By examining approaches from the financial domain, such as Excess Return, it attempts to isolate external market factors from the impact of news. The dataset contains news involving multiple stocks, where the same news event can have divergent market sentiment outcomes depending on the specific company perspective, making it ready for stock-aware market sentiment predictions.

To overcome data imbalance challenges inherent in financial market datasets, the research utilizes augmentation methods. This includes evaluating various textual augmentation techniques, ranging from traditional approaches to advanced LLM-assisted methods. By applying these techniques and using the carefully constructed dataset, a model was trained that yields comparable results in predicting stock price development directions, matching the performance of other machine learning-assisted trading approaches.

Abstract

Nachrichten über Finanzereignisse sind oft ein treibender Faktor für Aktienkurse. Daher erscheint die Nutzung von Large Language Models (LLMs) zur Entwicklung einer profitablen Handelsstrategie vielversprechend. Herkömmliche Ansätze der Stimmungsanalyse greifen jedoch zu kurz, wenn sie auf Vorhersagen für den Aktienmarkt angewendet werden. Dies liegt an der Komplexität des Marktes, der von verschiedenen Faktoren beeinflusst wird, und an den Nachrichten, die je nach den betroffenen Unternehmen zu widersprüchlichen Resultaten führen.

Diese Forschungsarbeit beleuchtet die kritische Lücke zwischen den textuellen Sentiment und der tatsächlichen Marktbewegung, indem sie einen innovativen Ansatz zur Vorhersage der Marktstimmung einführt. Die Studie umfasst das Crawlen eines Datensatzes, der Finanznachrichten von verschiedenen Herausgebern enthält. Anstatt sich auf menschlich annotierten Zielvariablen zu verlassen, konzentriert sich die Forschung auf historische Kursdaten, um Nachrichtenereignisse mit Kursentwicklungen zu verbinden. Durch die Untersuchung von Ansätzen aus dem Finanzbereich, wie z. B. den Excess Return, wird versucht, externe Marktfaktoren von den Auswirkungen der Nachrichten zu isolieren. Der Datensatz enthält Nachrichten, die mehrere Aktien betreffen, wobei ein und dasselbe Nachrichtenereignis je nach der spezifischen Unternehmensperspektive zu unterschiedlichen Ergebnissen in Bezug auf die Marktstimmung führen kann, so dass er sich für aktienspezifische Marktstimmungsvorhersagen eignet.

Zur Behebung von Datenungleichgewichten in Finanzmarktdatensätzen werden in der Studie Augmentierungsmethoden eingesetzt. Dazu gehört die Auswertung verschiedener Textanreicherungstechniken, die von traditionellen Ansätzen bis zu fortgeschrittenen LLM-gestützten Methoden reichen. Durch die Anwendung dieser Techniken und die Verwendung des erstellten Datensatzes wurde ein Modell trainiert, das vergleichbare Ergebnisse bei der Vorhersage von Aktienkursentwicklungen liefert und der Leistung anderer, durch maschinelles Lernen unterstützter Handelsansätze entspricht.

Acknowledgements

I am profoundly grateful to Prof. Dr. Christian Ledig for providing me with the opportunity to write my thesis at his chair.

I would also like to express my deepest gratitude to my supervisor, Sebastian, for his exceptional support and guidance throughout this thesis. He was always approachable and available for questions, and our regular discussions were not only insightful but also instrumental in shaping my work.

I am also grateful to Aaron, who spent countless hours in the library with me. His company and unwavering motivation were a constant source of support.

Special thanks go to Marie and Nicolas for their thoughtful advice and their willingness to help me navigate the challenges I faced during this journey. Their openness and understanding meant a great deal to me.

Finally, I am deeply appreciative of everyone who supported me in completing this thesis.

Contents

List of Figures	vii
List of Tables	viii
List of Acronyms	ix
1 Introduction	1
1.1 Related Work	2
1.1.1 Early Approaches to Sentiment Analysis	2
1.1.2 Advancements in Word Embeddings: Word2Vec and GloVe	3
1.1.3 BERT	4
1.1.4 FinBERT	4
1.1.5 Predicting Stock Price Movements	5
2 Background	7
2.1 Financial Domain	7
2.1.1 Stock Market	7
2.1.2 Stocks and Assets	7
2.1.3 Stock Market Index	8
2.1.4 Price Trends	8
2.1.5 Slippage	9
2.1.6 Excess Return	9
2.1.7 Efficient-Market Hypothesis	11
2.2 Technical Background	12
2.2.1 Sampling Strategies	12
2.2.2 Transformers	13
2.2.3 BERT	13
2.2.4 Pre-Trained Transformers	15
2.2.5 Overfitting	15
2.2.6 Data Augmentation in NLP	15

3	Data	18
3.1	Data Acquisition	18
3.1.1	Scraping of Social Media Posts	18
3.1.2	Scraping of Yahoo News article	19
3.2	Data Preprocessing	23
3.2.1	Extracting Ticker Names	23
3.2.2	Preparing Column Names and Values	24
3.2.3	Exploding Targets	25
3.2.4	Calculation of Sentiment Scores	25
3.3	Data Cleaning and Sampling	29
3.3.1	Data Quality	30
3.4	Sampling	31
3.4.1	Splitting the Data	31
3.4.2	Creating Sampled Datasets	32
3.5	Characteristics of the Dataset	33
4	Methods	35
4.1	Data Augmentation	35
4.1.1	Integrating Data Augmentation into the Data Pipeline	38
4.1.2	Integrating Data Augmentation into the Training Pipeline	39
4.2	Fine Tuning	40
4.2.1	WandB as MLOps Platform	42
4.2.2	Hyperparameters	43
4.3	Metrics	46
4.3.1	DirectionAccuracy	46
4.3.2	SentimentAccuracy	47
4.3.3	ProfitSimulation	47
4.3.4	AvgProfit	48
5	Experiments	49
5.1	Hyperparameter Tuning Results	49
5.2	Setup	51
5.3	Scaling	51
5.4	Sampling Methods	52
5.5	Time Period for Price Development	53

5.6	Data Augmentation Methods	54
5.7	Quality over Quantity	54
5.8	Excess Return	55
5.9	Social Media	56
5.10	Performance	56
6	Discussion	57
6.1	Interpretation of Results	57
6.2	Future Work	57
6.2.1	Backtesting Framework	57
6.2.2	Further Social Media Evaluations	58
6.2.3	Informational Pre-Training	58
6.2.4	Stock-Aware Predictions	58
7	Conclusions	60
A	Appendix	61
	Bibliography	62

List of Figures

1	Example of two Candlesticks representing a decreasing and increasing Trend of an Interval.	9
2	The Transformer - model architecture by Vaswani et al. (2017)	14
3	Visualization of the three stages of the Crawling of News Articles.	21
4	Visualization of generate_dataset.py preprocessing the observations and generating their target values.	23
5	Replication of posts form each associated assets.	25
6	Visualization of data_split.py filtering, splitting and sampling the observations.	29
7	Training Data	32
8	Predictions of the Model	32
9	Imbalanced Dataset Problem	32
10	System and Task Prompts for the Data Augmentation. Modified version of Møller et al. (2024) prompts.	37
11	Implementation of regression functionality in BERT's Sequence Classification	41
12	Overview of the Hyperparameters and their results	50
13	Comparison between Training with and without Scaled Target Values	52
14	Evaluation on different Sampling Approaches	52
15	Comparison between Different Periods for the Target Values	53
16	Performance Comparison between Selected Publishers and All Publishers	55
17	Attempt on Training on small Dataset of X Posts	56
18	Approaches for Incorporating Stock Information as Context to the Model	59
19	Visualization of the whole Data Processing	61

List of Tables

1	Distribution of the Publication Years	33
2	Various text augmentation methods on the data space with examples at different levels for the example sentence <i>Apple is facing delivery issues for the new iPads.</i>	35
3	Hyperparameter Search Space	49
4	Mean Metrics for each Augmentation Type	54
5	Comparison between RoR and Excess Return as Target Variable. . .	55
6	Final Metrics	56

List of Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CAPM	Capital Asset Pricing Model
CSV	Comma Separated Values
CUDA	Compute Unified Device Architecture
DB	Database
DBMS	Database Management System
DDGS	DuckduckGo Search - https://github.com/deedy5/duckduckgo_search
EMH	Efficient Market Hypothesis
GDP	Gross Domestic Product
GPU	Graphics Processing Unit
LLM	Large Language Model
MAE	Mean Absolute Error
MLM	Masked Language Modeling
MSE	Mean Squared Error
NMT	Neural Machine Translation
NLP	Natural Language Processing
R^2	R-squared
RNN	Recurrent Neural Network
ROR	Rate of Return
RTT	Round-trip Translation
SEO	Search Engine Optimization
URL	Uniform Resource Locator
US	United States
WandB	Weights & Biases (MLOps Framework https://wandb.ai)

1 Introduction

When it comes to outperforming the market, approximately 90% of traders fail to achieve this goal¹. A frequently cited reason for this failure is the influence of emotions, which often lead traders to deviate from strict trading rules (Fairchild, 2014). To mitigate this issue, trading companies like hedge funds employ algorithmic trading strategies. The movement of stock prices is influenced by several factors, including the current state of the economy, represented in macroeconomic indicators such as GDP, national income, unemployment, and inflation, as well as microeconomics like price movements of the certain stock and technical indicators such as the Simple Moving Average (SMA) and Relative Strength Index (RSI). All these factors can be quantified numerically, excluding emotional influences from the market. Training complex regression models on extensive historical datasets of these numerical indicators helps hedge funds overcome the emotional biases that can affect human decision-making.

While this might help with avoiding humane error sources based on emotions, emotions can often be a driving factor for the stock price. The sentiment of a newly published news can have a big impact on the value of a company, if its information is crucial. Whether published on news platforms or social media portals, these texts can depict a different strength of sentiment and emotions. Understanding and analyzing this textual information, without being fooled by its emotions, is essential for successful stock market predictions.

When trying to make decisions about the price movement based on news articles, it stands to reason that the task of Sentiment Analysis is being used. This involves usually classifying the sentiment of a text into categories such as *positive*, *neutral*, and *negative*. In the terminology of financial markets, sentiment is often referred to as *bullish*, *sideways*, and *bearish*. This leads to the first challenge when developing a sentiment model for the financial domain: good sentiment does not necessarily translate to a bullish market. Some news articles may have an overall positive sentiment but could still have a direct negative impact on the stock prices.

"ECB increases interest rate to 5%"

For example, consider the scenario where the European Central Bank (ECB) increases the interest rate to 5%. The average person might perceive this message positively, as it implies they will earn more interest on their savings. For companies, on the other hand, such a change will most likely lead to a decrease in stock prices, since it results in a more expensive refinancing.

Even when assuming that sentiments unrelated to the stock markets can be disregarded, there remain instances where the meaning can be ambiguous. When asking

¹<https://www.spglobal.com/spdji/en/documents/spiva/spiva-us-year-end-2019.pdf> - Research showing that 89% of fund managers were outperformed by the S&P 500 index over a decade

the question "What impact will this news have on the stock market?", it is necessary to define what is meant by the stock market in this context. Considering the following news title, it seems evident that it may influence the stock market (Chee, 2024). Whether the effect will be bullish or bearish depends on which company is the focus of interest.

Spotify wins lawsuit against Apple

This shows it is important to take additional cautious steps when wanting to utilize an existing sentiment model for the financial domain. The textual sentiment of financial news and the subsequent stock price development are not always aligned. Therefore, the aim of this work is not to extract the sentiment of the texts but to predict subsequent price changes triggered by a news event, a concept we refer to as *market sentiment*.

For this purpose, several posts about financial news will be acquired for a new dataset. Its target values will be generated based on the subsequent price development, while also considering the excess return of the stock to isolate it from other factors and influences. Different augmentation methods for text data will be evaluated to overcome overfitting.

1.1 Related Work

This section provides a concise overview of sentiment analysis and its historical development. Additionally, it examines a work applying sentiment analysis in the context of financial news. Since the focus of this work is more on the market sentiment than the textual sentiment, various studies are reviewed that aim to predict the market movement, using news as well as alternative inputs such as historical price data.

1.1.1 Early Approaches to Sentiment Analysis

Sentiment Analysis is a task in Natural Language Processing (NLP) that involves extracting the emotional tone conveyed by a text (Pang et al., 2002). In its early years, it gained popularity in analyzing customer reviews for opinion mining. With advancements in NLP, it has also become widely applied in other areas, such as news and social media, which introduce additional challenges like complex texts and irony (Hamborg et al., 2021; Zhang et al., 2019).

In the first years of Sentiment Analysis, the focus was primarily based on the polarities of single words in the text rather than the semantics of the full text. One of the pioneering works in this area was by Hu and Liu (2004), who published with their approach a top-cited sentiment scoring method with the aim of summarizing customer reviews. By applying lexical categorizing, a subset of relevant words is determined consisting mainly of adjectives. Using a lexicon, mapping each of these

words to a predetermined sentiment score, an overall sentiment score will be calculated by its sum, where negative words are annotated as negative numbers. Eight years later the author of the same work published the book Liu (2012), gathering the then-known state-of-the-art methods of sentiment analysis. Despite the introduction of new techniques such as considering word negations and utilizing SVM, word counting of polarities continued to play a crucial role in sentiment classification. This approach did not warrant a comprehensive semantic understanding of entire sentences but rather focused on the polarity of individual words. While this method delivers good performance for simple texts like social media posts, it is still insufficient for the complex texts of financial news. These kinds of texts often tend to have a more complex sentence structure and a dry nature.

1.1.2 Advancements in Word Embeddings: Word2Vec and GloVe

A way of having a more enhanced contextual representation of a word in a sentence can be achieved through *word embeddings*, which provide dense vector representations of words in a continuous vector space. These vectors are constructed such that words with similar meanings are located close to each other, capturing the context in which words frequently occur (Jurafsky, 2000). The theory behind this is the context of a word is defined by the set of words that appear nearby, following the idea of (Firth, 1957, page 11) famous quote: "You shall know a word by the company it keeps". Using this approach, the comprehensibility of sentiment analysis can be enhanced by capturing subtle meaning and semantic relationships in complex texts.

One framework that marked a significant advancement in fitting documents to such word embeddings is Word2Vec (Mikolov, 2013). It learns word associations from large text datasets, based on the center words in the documents and their relationship with its outside words. Based on the model variant, there are different approaches to computing the probability of these words relative to each other. For *Skip-grams* the probability of the surrounding words given a specific center word is calculated. While when optimizing the model using the *Continuous Bag of Words* approach, the center word is predicted based on a bag of outside words. The goal is to maximize the probabilities of the likelihood function, effectively placing similar words close to each other in the vector space. Using gradient descent, the objective function, which is the average negative log-likelihood, is minimized. Though *Word2Vec* enhances the understanding of complex texts through contextual representation, it does not scale well with large corpora and relies solely on the local context.

A more sophisticated approach leveraging global statistical information from the entire corpus is *GloVe* (Pennington et al., 2014). Inspired from other count-based approaches (Deerwester et al., 1990; Bullinaria and Levy, 2007) it constructs a co-occurrence matrix, representing how frequently a pair of words appear together in a defined window length. Otherwise than *Word2Vec*, the matrix of *GloVe* provides a comprehensive view of word relationships across the entire dataset. This matrix will be factorized to predict the logarithm of the probabilities of word co-occurrences.

By doing this, it combines the advantages of count-based and prediction-based approaches, enabling efficient training on large corpora, while also capturing complex patterns exceeding word similarity.

Building on these word embeddings and their approaches paved the way for making more context-aware predictions in sentiment analysis for complex texts (Rakshit and Sarkar, 2024; Wang et al., 2016; Tang et al., 2015b,a).

1.1.3 BERT

These word embeddings were an important step for the foundations of many transformer models (Vaswani et al., 2017). The input of the model is first transformed to word embeddings before it is passed to the transformer in combination with a positional encoding. Using this positional encoding and its attention mechanism enables the model to have an even more enhanced textual understanding of sentences, even in longer documents.

One implementation of such a transformer is done in the paper *Bidirectional Encoder Representations from Transformers* (BERT) (Devlin et al., 2019). It, along with its variations, is widely used in the field of sentiment analysis and is still delivering state-of-the-art results on multiple benchmarks (Csanády et al., 2024; Xie et al., 2020; Heinsen, 2022).

1.1.4 FinBERT

An important landmark study in the financial field of sentiment analysis is the work of Araci (2019). They build their model based on Google’s bidirectional transformer *BERT* (Devlin et al., 2019). The specialization involved three components: pre-training, classification, and regression. For each of these tasks, a different dataset was used.

For additional pre-training of the model, the TRC2-financial dataset was employed. This dataset is a subset of Reuters Corpora², filtered by financial keywords. This financial selection from the TRC2 dataset includes 46,143 news articles ranging from 2008 to 2010. The purpose of this dataset in the training process is to enhance the BERT model’s literacy in reading texts within the financial domain. However, they could not conclude that domain pre-training has a significant advantage for the downstream task performance. The training for the sentiment analysis downstream task was conducted and evaluated as both a classification and a regression task.

For the classification task, the *Financial PhraseBank* (Malo et al., 2014) dataset was used. It consists of 4,845 sentences randomly picked from financial news found in the LexisNexis database. The target data is human-annotated by 16 persons with a financial background. They had to label each sentence based on their expertise regarding how they believed the information might affect the stock price of the involved company. These labels are expressed as **negative neutral** and **positive**.

²<https://trec.nist.gov/data/reuters/reuters.html> - needs to be requested

The dataset was published in multiple subsets based on the levels of agreement among the authors. On the datasets without any sentences with contradictory annotations, the FinBERT model was able to reach an accuracy of 97%, outperforming other state-of-the-art models available at that time.

The regression task was trained on the dataset for Task 1 of the *WWW '18 conference financial opinion mining and question answering challenge* (Maia et al., 2018). Its instances consist of a mix of 1,174 financial news headlines and tweets. The target, expressing the sentiment score, is a continuous number ranging between -1 and 1. In comparison to the previous state-of-the-art results, FinBERT outperforms them in both MSE and R^2 .

Overall, this paper demonstrates that BERT is not data-hungry for training on downstream tasks and is already capable of delivering state-of-the-art results using only 1,000 to 5,000 observations. Unfortunately, there was no evaluation of how the model would perform on the stock market executing trading decisions based on its prediction outcomes. Its training was indeed conducted on a dataset designed to resemble potential stock market movements (Malo et al., 2014). However, it was not evaluated how well the estimate of the experts correlates with the actual subsequent price movements.

1.1.5 Predicting Stock Price Movements

Although *FinBERT* is specialized in working with financial texts and determining their sentiment, it is not optimized for predicting stock price movements. While an accuracy of 97% is impressive, there is no evaluation of how these predictions translate into assumptions about the stock market development. To gain a better understanding of possible results through AI-assisted trading methods, a few works are presented here. For the sake of comparability, studies that use a direction accuracy as an evaluation metric were specifically chosen. Contrary to the accuracy used in FinBERT, this accuracy assess whether the model predicted the right direction of the subsequent price movement, rather than the sentiment class. Hereby not only models interacting with textual data are considered, but also those incorporating other data types, such as historical price movements.

With *Galformer* Ji et al. (2024) presents a model based on Vaswani et al. (2017)'s Transformer architecture. Different than its original use cases, it does not handle NLP but is trained on a time series of daily adjusted close prices. Galformer forecasts over multiple days, which is optimized by a hybrid loss function, aimed at achieving both precise and trend-following predictions. This approach has led to an improvement in the direction accuracy of traditional transformers from 50.86% to 52.69%. Ismail et al. (2020) focused in their approach primarily on predicting the direction of the next day. They evaluated a variety of approaches, including Random Forests, logistic regression, SVM, and artificial neural networks. For each model, they calculated the accuracy for the years between 2001 and 2017. Utilizing persistent homology, they achieved long-term accuracies of up to 67.15%.

A multi-modal approach that combines textual information with price information as input is demonstrated by Sawhney et al. (2020). After processing both inputs through their respective decoders, a *Graph Attention Network* is employed to predict the movement of the stock price. With this approach, they achieved to surpass the previous baseline of the StockNet benchmark dataset³ with an accuracy of 60.8%. Another study focused on predicting stock price movements using financial news by employing a fine-tuned contextual embedding recurrent neural network (FT-CE-RNN) (Chen, 2021). They utilized BERT to train a contextual embedding on Bloomberg’s proprietary News dataset⁴. Passing these embeddings to a RNN, enabled them to achieve accuracy results between 56.6% and 74.5% depending on the proportion of the test set.

³<https://paperswithcode.com/sota/stock-market-prediction-on-stocknet> - Benchmarks on the StockNET dataset (Xu and Cohen, 2018)

⁴<https://www.bloomberg.com/professional/product/event-driven-feeds/>

2 Background

2.1 Financial Domain

This section provides background knowledge about financial markets to help the reader understand how stock prices are determined. It also explains the characteristics of stocks, including how they are grouped and identified. Furthermore, it presents an approach for isolating external factors in the price development of a stock.

2.1.1 Stock Market

The concept of a *stock market* describes the process of different actors arranging the exchange of shares of companies or other securities between two parties (Teweles and Bradley, 1998). For such a process, a minimum of two actors are necessary, including a buyer and seller of the asset. This trade can be executed solely between these two participants, however, most trades involve a stock exchange as a second party. *Stock exchanges* are regulated platforms for managing and executing orders from traders. Originally, orders were settled face-to-face on the trading floor through a process called *open outcry*. Although most of these systems nowadays operate online and automated, there are still market hours restricting when trades are possible, which are generally between 9:30 AM to 4:00 PM. Some of the most prominent stock exchanges include the *New York Stock Exchange*, the *NASDAQ*, the *London Stock Exchange*, and the *Frankfurt Stock Exchange*. Due to regulatory requirements, acting on these exchanges is restricted to licensed brokers or authorized participants. To also enable individual investors access, brokers act as intermediaries between the investors and the market against a commission.

One task of stock markets is the *price discovery*, where the market value of an asset is determined by considering the willingness of the traders to sell or buy for a certain price. These prices are called *bid price* on the buyer's side and *ask price* on the seller's side. If both of them align, the exchange's *order matching system* will fulfill the corresponding orders. All remaining offers are kept in what is known as an *order book*, which keeps track of the prices and sizes of unfilled orders. This system is used to identify the current best price for buying (lowest ask) and best price for selling (highest bid). The gap between them is referred to as *spread*, whose size varies depending on the liquidity provided by *market makers*.

2.1.2 Stocks and Assets

A stock represents the share of the ownership of a company. The sum of the value of all shares, also called *market capitalization* depicts the overall value of the asset. In common parlance, stocks refer to publicly traded companies, which are listed on stock exchanges. Companies issue shares primarily to secure flexible funding without having to pay interest. In return, the investors profit from the growth of

these companies and may receive dividends as a share of the company's earnings. Depending on the company's choice, stocks can also grant the shareholder voting rights on corporate decisions.

To have a short and unique identifier for addressing securities on stock exchanges, *ticker symbols* are used. These symbols consist of 1 to 5 uppercase letters, often abbreviating the company's name, and are unique within each exchange. In some cases, companies can be represented with two tickers on the same stock exchange. Alphabet Inc., the parent company of Google, chose to offer shares in two different stocks, which primarily differ in voting rights. This structure enables the company to issue new shares of *GOOG* to raise capital, without diluting control over the company, which is managed by the also publicly traded *GOOGL* shares.

2.1.3 Stock Market Index

Stock market indices are used as a measure of the performance of a certain segment of the stock market. In this way, the overall direction of the market can be tracked and compared to historical trends. It does not only serve as a metric but can also be invested in through *Exchange Traded Funds* (ETFs). This financial instrument is realized by fund managers, who determine the index's composition based on a factor like the stock's market capitalization. To reflect the price change of the index, the institutions replicate the index by holding the stocks that make up the index.

The composition of an index can be guided by various factors, such as regional-based selection, like the *DAX* or *MSCI Emerging Markets*, or exchange-based, as in the *NASDAQ-100*. With including the 100 largest non-financial companies listed on the NASDAQ stock exchange, NASDAQ-100⁵ represents one of the currently most important sectors of the stock market. Given that the tech companies featured on this index have proven growth potential and high media presence, the selection of stocks for consideration is focused on this index.

2.1.4 Price Trends

Price charts are usually displayed using candlestick charts. Each candlestick represents an interval, which can range from seconds to hours (referred to as intraday candlesticks), or from days to months. The price characteristics within one candlestick are represented in a box-plot similar structure, which can be seen in Figure 1. The range of the box is defined by the *OPEN* and *CLOSE* prices, while the *LOW* and *HIGH* prices are represented by the extreme values of the whiskers. The coloring of the candlestick is determined by the relation of the *CLOSE* price to the *OPEN* price.

When working with historical price charts it is common to use an adjusted closing price, which considers dividends but also other corporate actions. Especially for

⁵<https://www.nasdaq.com/solutions/nasdaq-100> - NASDAQ-100 Index

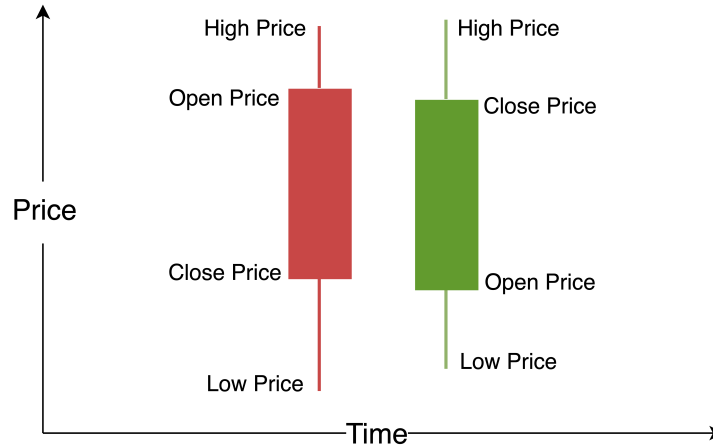


Figure 1: Example of two Candlesticks representing a decreasing and increasing Trend of an Interval.

stock splits it is important, where the amount of shares is modified while maintaining the same market capitalization. In a two-for-one split, for example, every shareholder would receive twice as many shares, while also the price is reduced to its half value. When not using an adjusted close price, such events could appear like a sudden significant change in the value of the stock.

Price movements indicating a positive trend are termed *bullish*, whereas negative trends are described as *bearish*.

2.1.5 Slippage

Slippage is a financial term, describing the difference between the expected price and the actual realized price. Slippage can occur due to various factors, including the rapid changes in prices, which means that the price at the time of decision-making can already be outdated. Another factor is the spread between the bid and ask prices, representing the prices on the buyers and sellers side. Besides that, there are also other factors, like high volume trades during low liquidities and possible trading fees.

2.1.6 Excess Return

The net gain or loss of an investment can be expressed as the *Rate of Return* (RoR). It expresses the entire value change of the position over its period as a percentage of its initial amount (see Eq [1](#)).

$$\text{RoR} = \frac{\text{Final Value} - \text{Initial Value}}{\text{Initial Value}} \quad (1)$$

Often, a company's price change is not only impacted by information about the company itself but also by other extrinsic factors. Some of these factors can overshadow

the event of the news to such an extent that information, which would otherwise have been positively priced in, leads to a stock price decrease. Since our selection of the stocks is determined by the NASDAQ 100 index, these factors can be involved by events regarding US politics or the technology sector. A relevant event in US politics could be a monetary policy change, such as adjustments to the interest rates by the *Federal Reserve System* (Sellin, 2001). Another macroeconomic influence can be the announcement of rising inflation, as it often leads to reduced consumer spending. Events regarding the technology sector could include regulatory changes, such as stricter data privacy laws, which affect a large amount of tech stocks.

A common practice in the field of financial markets to evaluate investments isolated from other market factors is to consider the excess return. This metric represents the return of an investment exceeding the expected return of a benchmark, effectively separating the actual total return from external factors. A popular implementation of this concept is Jensen's Alpha (Jensen, 1968), which is traditionally used for measuring the difference between the return of a portfolio in relation to the overall market. In this way, it assesses the performance of a portfolio while accounting for the current risk of the market. This isolated risk can be viewed as the external factors influencing the stock price. For a given asset, it calculates the return of this security (Eq 5) and subtracts the corresponding CAPM value.

$$\text{MarketRiskPremium}(t_i) = \text{Return}(\text{QQQ}, t_i) - \text{RiskFreeRate}(t_i) \quad (2)$$

$$\text{CAPM}(a, t_i) = \text{RiskFreeRate}(t_i) + \beta_a \times \text{MarketRiskPremium}(t_i) \quad (3)$$

$$\text{ExcessReturn}(a, t_i) = \text{Return}(a, t_i) - \text{CAPM}(a, t_i) \quad (4)$$

CAPM The foundation of the *Capital Asset Pricing Model* (CAPM) formula was introduced by Sharpe (1964) and Lintner (1975). CAPM is a commonly used framework in financial analysis for determining whether a security is fairly priced based on its systemic risk. It calculates the expected return of an asset by combining the return of a *risk-free* investment with an adjustment for the asset's risk relative to the market, based on its *beta* and a *market risk premium* (Eq 3).

Beta The beta value of a stock describes how its price is expected to fluctuate in relation to overall market movements. When the beta value is higher than 1, the stock's price is more volatile than the overall market. For example, a stock with a beta value of 1.5 would be expected to experience 50% stronger movements than its benchmark index. This applies in both directions, meaning that when the market rises by 5%, the stock's value is expected to increase by 7.5%; similarly, a market decline of 5% would result in a 7.5% decrease in the stock's price. In contrast, a beta value of less than 1 indicates that the stock's price movements are slower and less volatile than the overall market. A higher beta is generally associated with higher risks. The beta value is derived from statistical measures of historical data.

Market Risk Premium The *market risk premium* represents the additional expected return over the *risk-free rate* (Eq 2). It quantifies the compensation investors expect for taking on additional market risk compared to investing in a bond considered risk-free.

Risk Free Rate A risk-free rate represents the annual rate of an investment that is considered free of financial risk (Weil, 1989). Typically, these are modeled by bonds with an assured interest rate, which is paid regardless of market conditions. For fixed-rate bonds, this rate is set at issuance and remains guaranteed until maturity. In the case of floating-rate bonds, the interest rate is periodically adjusted to reflect current market conditions, approximating the rate of a bond issued at that moment. In the domain of financial markets, US Treasury Bonds are predominantly used as a proxy for the risk-free rate, due to their perceived safety and reliability (Damodaran, 2008). Depending on the period of the planned investment, different Treasury Bills may be appropriate. For short-term investment – what our intention for trades based on news article is – it is common to use a 1 Year US Treasury Bond. It is important that the investment period does not exceed the bond’s maturity to ensure the expected risk-free return is maintained.

2.1.7 Efficient-Market Hypothesis

The *Efficient Market Hypothesis* (EMH) (Fama, 1970) implies that all available information is fully and immediately reflected in an asset price, as soon as it becomes publicly known. This includes information about the current state of the company, macroeconomic factors, expected risk, as well as anticipated events and future cash flows. This is due to the high competition among traders and investors, who constantly analyze prices and information, driving prices toward equilibrium. The EHM does not claim, that it is impossible to make profits by trading, but rather that, in an efficient market, it is not possible to consistently achieve excess returns.

It is distinguished between public information, which can be accessed by outsiders, and private information, reserved for insiders of the company. Which information is incorporated by the market, depends on the degree of market efficiency. In its *weakest form*, it is still possible to profit from public information such as news, as the efficiency regards only the reflection of historical price movements. Consequently, long-term profitable trading strategies assisted by regression models trained solely on historical price data are not possible. The *semi-strong form* assumes that public information is rapidly priced in, to an extent that trading based on these events becomes unprofitable in the long term as well. The *strongest form* covers all kinds of information, including both public and private, into asset prices. In this scenario, no investor could consistently outperform the market, regardless of having insider information. In reality, private information is rather priced in slowly, which results from its confidentiality and strict regulations for insiders.

The form which is most applicable to real-world markets is the semi-strong form. Even though it has its flaws and is not always consistent (Malkiel, 2003; Tiŕan, 2015),

it can be observed that particular publicly known information is quickly priced in. This does not completely eliminate the possibility of profiting from this information, but it requires prompt action to compete with the market.

2.2 Technical Background

This section outlines the fundamental concepts and techniques of sampling, augmentation, and transformers, providing a basis for the subsequent sections, which will delve into their implementation and application.

2.2.1 Sampling Strategies

Sampling describes the process of selecting observations for a subset from a larger set of all available observations. The goal may either be to represent the original population as closely as possible, or to shift the imbalance to a more favorable distribution for training.

There are multiple strategies available for sampling. One of the most basic is *Simple Random Sampling (SRS)*. By randomly selecting, every observation has the same chance of being picked, reducing the selection bias. The subset resulting from SRS is representative of the population.

Stratified Sampling offers a more structured approach. As a prerequisite step, the population is divided into subgroups, based on common criteria. From each of these strata a sample is drawn, ensuring that all parts of the population will be considered. Stratified sampling methods themselves can vary in their implementation, including how strata are defined and the sample sizes are drawn from each group. The definition of the strata could be categorical labels for classification tasks or defined ranges of a target variable in regression tasks. These ranges could be pre-defined by borders or divided into n equal-sized intervals. The sampling of these groups could be targeted to achieve a uniform distribution or to represent the original population.

Oversampling is a technique used to address class imbalance, particularly in situations where traditional sampling methods would result in insufficient representation of the minority classes. Its implementation can be as simple as randomly duplicating instances from the minority class. While this method helps mitigate the bias towards the majority class, it also carries the risk of overfitting the model, most notably when the maximum allowed ratio is chosen too generously. This is because inserting duplicate instances does not introduce any new variance to the minority class, leading to poorer generalization performance during the training. Approaches like *SMOTE* (Chawla et al., 2002) and *ADASYN* (He et al., 2008) are developed to overcome this challenge. They create synthetic data points by interpolating between existing observations of the minority groups. However, these methods are not applicable for text data unless it has been transformed into embedding, which comes with limitations, such as the need to decide on a model up front.

2.2.2 Transformers

The approach of *transformers* (Vaswani et al., 2017) offers a different strategy than *Recurrent Neural Networks* (Hochreiter, 1997; Chung et al., 2014) for implementing complex sequence modeling tasks.

Since transformers operate without the need for recurrence, they require an alternative method to consider the position of the tokens. This is realized by combining the word embeddings with a positional encoding. Through a sinusoidal function, consisting of a sine and shifted cosine function, the positional encoding creates a unique identifier for each position, providing the means to distinguish the sequential order of tokens.

A technique contributing to transformers' ability to parallelize computation and long-term textual understanding is the *attention mechanism*. Using it the model is able to determine relevant parts of the text to focus on, even over longer sequences. The computation of the attention mechanism consists mainly of three components: **query**, **key**, and **value**. The **query** represents the information the mechanism seeks to focus on. The **keys** are representations of the input elements, indicating their relevance to the query. Every *key* has an assigned **value**, which holds corresponding semantical information for the token for further processing. By calculating the dot product of the query with the key, the importance of each token is determined, indicating whether its corresponding value should be considered. The resulting attention scores, when multiplied with the values, produce a weighted sum that provides a contextually aware representation of the input sequence.

Multiple of these attention layers are stacked to a multi-head attention layer, enabling each layer to concentrate on a different part of the text. The computation of their attention scores can be done in parallel.

The model architecture, which is shown in Figure 2, consists of N encoder and decoder layers. The encoder layer, having the task of extracting the information of the text to create a more abstract representation, consists of a multi-head attention and feed-word layer. Each of these layers has its own fitted fully-connected feed-forward network for feature extraction, which independently processes each position to prepare the input for the next layer. For its original purpose of text generation, their output is passed to the decoder network of the transformer, to generate an output sequence. The decoder layers are built similar to the encoder layers, with the exception of having an additional multi-head attention layer interacting with the output of the encoder, thereby adding an *encoder-decoder attention* mechanism, besides the *self-attention*. For other down-stream tasks, like sentiment analysis, the decoder network can be omitted.

2.2.3 BERT

BERT (Bidirectional Encoder Representations from Transformers) is a language representation model widely used in the field of NLP, which was introduced by

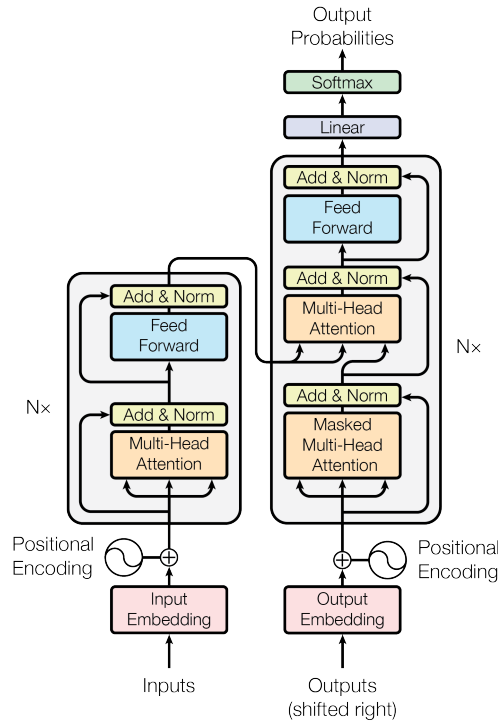


Figure 2: The Transformer - model architecture by Vaswani et al. (2017)

Google researchers in 2018 (Devlin et al., 2019). One of its contributions is the innovative bidirectional pre-training of the encoder. BERT's architecture is based on the transformer model, which uses self-attention mechanisms to process input text efficiently and capture intricate dependencies between words. Contrary to its original approach, BERT only uses the encoder network.

Former traditional models used to be trained on predicting the next token, based on the previous tokens to the left. Using a so called *Masked Language Modeling* (MLM) the model learns to predict masked tokens within a sentence, while also considering the tokens to the right as context. This facilitates BERT to a significant advantage in generating contextual word embeddings of high quality.

Like other pre-training techniques is MLM also an unsupervised approach. The masking of the words within the documents is applied randomly during the training, giving the model the task of predicting the recently masked tokens. The training corpus included the entire English Wikipedia and the BooksCorpus (Zhu et al., 2015) dataset, totaling 3.3 billion words. This broad dataset enables BERT to develop a deep understanding of language context and structure.

They also demonstrated using the GLUE (General Language Understanding Evaluation) benchmark (Wang et al., 2019) that BERT outperforms back then state-of-the-art models in various collections of NLP tasks, including sentiment analysis, semantic similarity, question answering, and others. As a method for the training on the downstream task, they only fitted one additional classification layer. Due to its architecture and extensive pre-training BERT's rapid convergence is capable

of delivering these results already after three epochs. Even the smaller-sized model $BERT_{base}$ with 110 million parameters, achieved better results than the competitor models. This makes BERT's base model especially attractive for small research teams, as it can be trained using relatively few resources, while still delivering state-of-the-art results.

2.2.4 Pre-Trained Transformers

Pre-Trained Transformer is a framework where models are first pre-trained on a large data set, before being specialized on a specific task. Often they are also referred to as *Generative Pre-Trained Transformer* (Radford, 2018), which however does not apply to BERT, as it misses the decoder network required for generative tasks.

The process of training (generative) pre-trained transformers consists of two essential stages, involving an initial general pre-training on a large corpus, followed by fine-tuning on a specific downstream task. Most models of this kind are provided with already pre-trained weights, significantly reducing the resources required for downstream task training. While fine-tuning is crucial for the actual task, further pre-training can help to make it more affine on a specific domain. The pre-training does not require any target values and is conducted as an unsupervised learning process.

2.2.5 Overfitting

Overfitting describes the phenomenon where a model becomes so closely adjusted to its training data that it loses the ability to generalize to unseen data. It can have various causes, which can either be related to the model or to the properties of the data. Models that are too complex or trained for too long tend to exhibit overfitting. However, datasets lacking sufficient training data or having an imbalanced distribution can also contribute to it.

2.2.6 Data Augmentation in NLP

A popular approach in the field of Computer Vision for avoiding overfitting is Data Augmentation. While traditionally associated with image-related task, its application has also gained attention in Natural Language Processing recently. Data Augmentation is an approach for expanding the existing instances by slightly different versions of their originals. The reason why it mitigates overfitting is because a bigger dataset is gained by it and constantly seeing new instances in every epoch, will distract the model from learning just by memorizing.

The practices used to realize data augmentations differ across various fields. For the field of *Natural Language Processing* Bayer et al. (2022) compares and provides an overview of different methods for augmenting textual data. The implementations

surveyed in this work range from conventional techniques to first approaches using AI-assisted methods. They categorize the different strategies into a taxonomy that distinguishes between different spaces and levels.

Most of them occur in the data space, meaning that the data has not yet been transformed into a feature space as it would happen during the tokenization. In these approaches, manipulations are applied directly to the text, which can happen on different levels of granularity.

At the most granular level, **Character Level** text augmentation involves adding, removing, or altering individual characters within the sentence. It can be used to add noise by randomly inserting characters, but it can also be systematically applied to imitate typos, such as swapping random characters with their neighbors or with characters that are close to the original on the keyboard. This can make the model less prone to user-introduced errors, although such issues should ideally be addressed during the pre-training stage for LLMs.

Moving up to the next level of granularity, **Word Level** text augmentation alters entire words, often including the usage of synonyms. This can affect the position of the altered word, or it result in its removal or replacement with another word. By introducing these variations, the model is exposed to a wider range of linguistic expressions, improving its robustness and ability to generalize.

Another level mentioned by Bayer et al. (2022) is the **Phrase Level**, which operates on smaller segments within a sentence. This includes operations such as paraphrasing, reordering the sentence structure, or adding subordinate clauses.

The least granular level, applied to the entire text, is the **Document Level**. Here the transformation either applies to the entire body of text or at least considers it as context. This level is divided into the two task types of *translation* and *generative*, usually aided by ML models, especially for the generative tasks.

Augmentations using translation are generated in a two-stage process referred to as **BackTranslation**. Hereby the original text will be translated to another language and back to its source language. The result will ideally retain the same meaning, but may still differ in wording and structure due to nuances and variations introduced by the intermediate language. This technique, also known as Round-trip Translation (RTT) first gained popularity as an evaluation measure for machine translations, indicating that sentences being similar to their original are of high quality (Chan, 2004). Aiken and Park (2010) assessed the efficacy of RTT as an evaluation metric and concluded that, while this method has predictive power, it cannot be considered a bulletproof measure due to variations in the translations, which still maintain the original meaning of the sentence. Edunov et al. (2018), which employed back-translation for further training of NMT models, found it to be a highly effective method for data augmentation in the field of NLP.

According to Longpre et al. (2020) most conventional data augmentation techniques fail to yield significant improvements when applied to LLMs. Therefore a more sophisticated approach - using another LLM for augmentation - is employed to address the challenge more effectively, essentially like 'fighting fire with fire'. Ding et al.

(2024) describes different methods of how LLM can be used for data augmentation in the field of NLP. This includes prompting LLM models to generate similar examples of given instances (*Data Creation*), annotate unlabeled datasets (*Data labeling*), and create contradictory examples to the original instances (*Data Reformation*).

The other space addressed in Bayer et al. (2022) operates in the feature space. In the case of *Transformer* models, this refers to modifications made to the output of the embedding, before it is passed to the encoder of the transformer. This can either be done by interpolating between two instances or adding some noise. Building on this idea, *NEFTune* has demonstrated significant effectiveness in enhancing model performance by employing noise to the embedding, encouraging generalization by introducing variations in the feature space (Jain et al., 2023). Evaluated across various instruction fine-tuning datasets, this approach achieved performance boosts up to 34.9 percentage points (Alpaca (Taori et al., 2023)) and an average improvement of 15.1 percentage points.

3 Data

The goal of this work is to have a textual dataset containing information about events regarding traded companies so that a model can learn the impact of this information on the market. The model should be able to predict the subsequent price direction on the stock market, rather than merely making assumptions regarding the emotional tone of the text. As the outcome depends on the subject of the news, every observation should be linked to the relevant asset.

Such a dataset must include posts about topics, that address recent events relevant to the pricing dynamics of an asset. To enable learning from this data a target variable is required, representing the subsequent stock price development of the involved company. In the optimal case, this also includes news containing references to multiple tickers, so that models could potentially be trained to capture the ambiguous outcomes of a single event across different companies. For this, the dataset must either already include the described target variable or contain the relevant columns to derive the market’s subsequent reaction, such as the names of the involved companies and the publication time.

Existing datasets either focused on the sentiment of the text, which does not necessarily align with the market sentiment, or were limited to posts that each involved only a single stock (Maia et al., 2018; Xu and Cohen, 2018). Therefore, we decided to create a dataset that fulfills all the aforementioned requirements. The following sections outline the criteria and methods used to collect the posts, as well as the process for generating the target variable. Additionally, they detail the sampling and splitting procedures employed to divide the dataset into the training and testing versions.

3.1 Data Acquisition

As sources, we considered news articles from information portals and social media posts on the platforms *Reddit* and \mathbb{X} . During the data acquisition process, we specifically targeted content related to stocks included in the index outlined in Section 2.1.3.

3.1.1 Scraping of Social Media Posts

Social Media posts can serve as a valuable source of information for anticipating market movements since the instantaneous form of micro-blogging provides a faster way to react to recent events. For this purpose posts from the social media platforms *Reddit*⁶ and \mathbb{X} ⁷, formerly known as *Twitter*, were gathered.

⁶<https://www.reddit.com>

⁷<https://x.com>

To retrieve data from *Reddit*, the python library *PRAW*⁸ is utilized, which provides an interface to communicate with the official *Reddit* API. It is implemented in compliance with *Reddit*'s API rules and also automatically makes further requests, to retrieve the full pagination. To ensure financial relevance, the posts are gathered from specific subreddits, which are groups focusing on financial topics. With *PRAW*'s *Subreddit* class and its methods, a capped number of posts with each request can be retrieved. Different search queries are employed for each asset to maximize the scope.

A possible way to acquire posts from *X* is through their official API. However, the free plan of it mainly covers the usage for posting tweets and the cheapest plan offering search functionality starts at \$5,000 per month⁹. Alternative options had to be evaluated, as the associated costs are economically not viable for a master's thesis. Additionally, the previous academic research access program has been discontinued, except in cases mandated by Article 40 of the Digital Services Act¹⁰.

X offers a service known as *Syndication*, which enables the embedding of individual tweets or timelines of users into websites. While still not giving the possibility to search for tweets, it is able to obtain tweets based on a post identifier or username. When requesting it for a specific user, up to 100 posts can be received. The tweets are not sorted in chronological order but by popularity. This is useful for our case since those tweets are probably more relevant and offer a mix of posts from a broader time range. A user on *GitHub* described a way how tweets from this tool can be programmatically extracted¹¹, which we used in our implementation. In this way, a dataset of relevant tweets can be assembled, when provided with a list of financial influencers.

3.1.2 Scraping of Yahoo News article

As a source of our data, *Yahoo*'s service *yahoo!news* was chosen. This service works as an internet-based news aggregator, enabling us to fetch news articles from various publishers on a single platform. This simplifies the data acquisition process, as the solution for retrieving the articles has to be only implemented for a single source. Additionally, the variety of publishers includes previews or even full text for articles, which would otherwise have been exclusive to subscribers on the publisher's page. *Yahoo*'s finance sector also incorporates the *yahoo!news* service with added features. One of these features is the "In this article" display, showcasing stocks mentioned in the article along with their daily price changes. When a news article involves multiple listed companies, this widget will display them all. That information is crucial, as it is necessary for the dataset to be used for *Aspect-Based Sentiment Analysis*.

⁸<https://praw.readthedocs.io/> - The Python *Reddit* API Wrapper

⁹<https://developer.x.com/en/products/x-api>

¹⁰<https://devcommunity.x.com/t/academic-research-access/221102>

¹¹<https://github.com/zedeus/nitter/issues/983#issuecomment-1678942933>

The most essential data that needs to be retrieved for the dataset are the news titles and their related assets. As first method to retrieve the articles published on *yahoo!news* without overhead was using Yahoo's API. Yahoo's finance search endpoint ¹² contains besides all available quotes for the search, also relevant news articles. While not containing the full text, the response contains all other relevant information like the title, publisher, publication time, URL, and all related assets of each article. However, the results are limited to the latest eight articles, which makes this endpoint unsuitable for our purpose. Over the duration of a longer period, it would still be possible to gather a larger dataset, which would nonetheless not contain any historical articles, introducing a bias towards the current market situation.

Since the API could not be used due to its limitation of limited results, we had to resort to scraping. For this a scraper was implemented, which retrieves all available data present on the Yahoo News page regarding the article based on a URL. In order to retrieve these URLs search engines were utilized, as the homepage only displays recent articles.

All the data regarding the scraping of the news articles will be persisted and processed in an SQL database. As a database engine, SQLite¹³ was selected for its ability to operate without any dependent server or service. Despite being a lightweight database solution, its capabilities are more than sufficient for this task. The content of SQLite databases is stored in a single file, making them highly portable. This enables an easier publication of the data since the file can be easily copied and shared across different systems. Furthermore, utilizing a database system like SQLite over CSV files offers several advantages. One benefit is the possibility to specify schemas with unique constraints for better integrity, ensuring that no duplicate articles will be added. Furthermore, using a DBMS enables a more efficient updating of specific rows, which is essential during the second stage of scraping the content. It also supports concurrent processing, enabling parallelization of some tasks across the multi-stage scraping workflow.

The whole scraping process involves three stages, which workflow is visualized in Figure 3. First, the *URL Crawler* saves bare URLs of Yahoo News articles to the DB, with each entry annotated with `scraped=0` to indicate it has not yet been scraped. In the next stage, the *Selenium Scraper* will pick up the URLs, which are not yet scraped or marked as errors. After successful scraping all retrieved fields will be added to the entry of the given URL in the DB. On failures, this row will be marked in the database, so that such URLs will not be tried again. To enhance the efficiency of this stage, a number of workers, each having their own *ScrapeWorker* can be defined. The task for each URL will then equally be distributed over each worker, using Python's built-in *ThreadPoolExecutor*. The final stage involves processing the scraped data, to make it suitable for the training tasks. This includes fetching all successfully scraped data from the DB along with

¹²<https://query2.finance.yahoo.com/v1/finance/search?q> - Yahoo Finance's Search Endpoint

¹³<https://sqlite.org/>

some preprocessing and generating the target value for each news-asset pair. The resulting information is then exported to a CSV file.

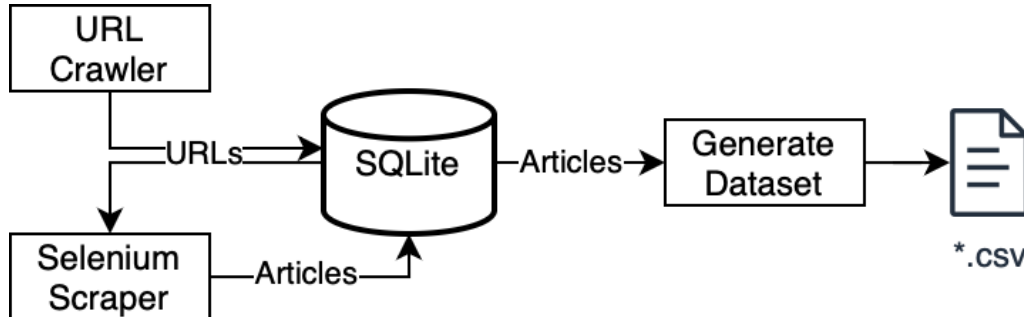


Figure 3: Visualization of the three stages of the Crawling of News Articles.

During the development, various search engines were tested to accomplish the scraping of the URLs. Therefore it is possible to interchangeably select different platforms for the `news.url.Scraper`. In order to achieve high modularity all the implementations of these services have their query be set with `set_search(query)` and will return their results incrementally with `next_page()` until no more hits are available.

One of the used services is the Google search itself. Google Cloud Platform offers a *Programmable Search Engine*, which can be used to create a custom search engine based on Google’s index. When creating such an engine, you can specify which websites will be included in the search scope of the engine. For our purpose, this was set to `finance.yahoo.com/news/*`. As an outcome, you will receive a custom Google Search engine that delivers results exclusively from financial Yahoo News articles^[14]. Google’s *Custom Search JSON API* provides an endpoint to get results from this previously registered *Programmable Search Engine*^[15]. Its free contingent includes 100 requests for each day. Given that one query can contain up to ten URLs, this allows retrieving up to 1,000 news articles each day. Starting from the 101st request, each further call will cost half a cent, totaling 5\$ for 1,000 additional requests. Since this service is bound to a contingent, credentials are required for its usage, which can be passed as environment variables.

To simplify the workflow with this API, we developed the wrapper `GoogleCSE`. A single request to the endpoint will only return the results of one page. Therefore, the method `next_page` will return the results of the current page while automatically setting the URL for the upcoming request to the next page. Hence, by incrementally calling this method, all available URLs can be retrieved. However, the results are still limited to a maximum of 100 results for each query.

The open source project *duckduckgo_search* (DDGS) offers a Python interface to DuckDuckGo’s search engine, without requiring any API key or credentials^[16]. Its

¹⁴Programmable Search Engine created for this thesis only returning financial Yahoo News articles: <https://cse.google.com/cse?cx=51052ab44096d4dda>

¹⁵<https://developers.google.com/custom-search/v1/reference/rest/v1/cse/list>

¹⁶https://github.com/deedy5/duckduckgo_search - GitHub Repository for a DuckDuckGo search interface

implementation already handles the pagination, therefore the `next_page` of our implemented wrapper will already return the complete search result on the first call. Depending on the search term DDGS can deliver up to 2000 results, exceeding the maximum possible 100 of Google's *Custom Search API* by far. The method name was still retained, to ensure modularity of the search engine classes. As opposed to Google's custom search engine, DDGS will search the whole web. In order to also only retrieve here financial articles from Yahoo News the following custom search query is being used:

```
site:finance.yahoo.com/news/ {query}
```

An encountered limitation of this library is *RateLimit* exceptions, even when obeying a modest usage. We documented this issue on its GitHub repository, but despite an initial fix, the problem persists sporadically and seems to be location-dependent¹⁷.

Before starting a crawling process for URLs, the latest stocks gathered within the last day are retrieved from the database to avoid wasting resources on already scraped URLs. For each asset, that is not already covered in this list, a search process is started with the selected search engine platform. The query is set to search for the specific asset, with the option to also specify a publisher to target articles from desired resources. Using the method `next_page` the URLs will incrementally be persisted to the DB, while the table schema will prevent duplicate entries.

For the scraping of the articles, Selenium was used, which is a framework for browser automation¹⁸. Using its Python package, the process of scraping the news pages was implemented into a class called `ScrapeWorker`. Before the worker invokes a requested URL, sanitization is applied first. Some URLs might include a language code inside their subdomain. This part will be removed from the URL, as some localizations have a different website structure. After visiting the Yahoo News page for the first time, a cookie disclaimer will be displayed. In order to retrieve the content of the page, it has to be accepted or declined by the scraper. Since this disclaimer will not appear anymore on subsequent visits, a method `click_if_exists(class_name)` was implemented to handle it. After some delay to ensure that the page is completely loaded, its relevant content will be scraped. This is done by localizing the elements of interest using defined class or path identifiers. In this way, the title, publisher, publication date, and a list of related assets are extracted. For the full text of the article, all `<p>` elements inside the body class will be concatenated, after clicking a potential *Read more* button to collapse the content. The worker returns this information as a *DTO*, enabling more convenient processing by the database module.

¹⁷https://github.com/deedy5/duckduckgo_search/issues/213 - Documentation of RateLimit issue

¹⁸<https://www.selenium.dev/>

3.2 Data Preprocessing

Data Preprocessing describes the procedure in this work of transforming the crawled observations to a state, where they can be used for training according to our task specifications. This process is implemented by the `generate_dataset.py` script and involves various steps to organize the data and generate target values for them. The subsequent section provides a detailed description of each step in this process, as illustrated in Figure 4.

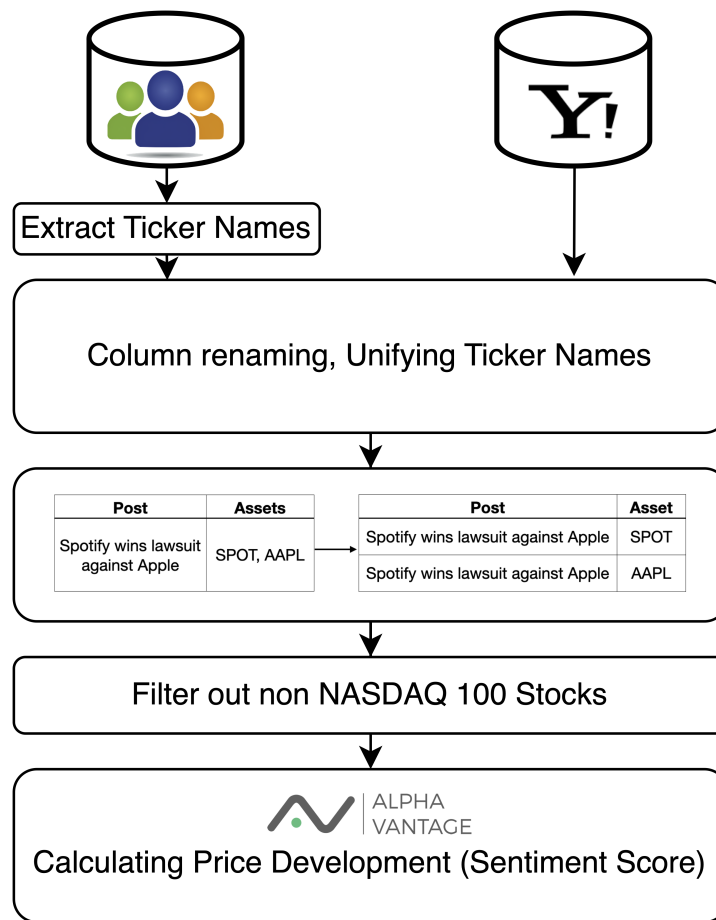


Figure 4: Visualization of `generate_dataset.py` preprocessing the observations and generating their target values.

3.2.1 Extracting Ticker Names

A majority of news articles come annotated with the stock companies they mention. For all other articles or texts from sources without such annotations, like social media posts, this information has to be extracted during the preprocessing, as it forms the foundation for calculating the market sentiment score. Such posts often mention the ticker names within their text, usually represented by 2-5 uppercase characters. When searching for them based on these criteria, it would also include

terms that do not represent stock names. As a requirement, searches could target terms with a \$-prefix, which is typically common in microblogging. However, during the initial evaluation of this method, it was found that only 20% of the processed posts contained at least one match. To achieve higher coverage, an approach utilizing a search-word map is employed, containing the stock selection mentioned in Section 2.1.3. This map assigns each search word to the ticker name of the according company. The search words are the full company and ticker names of the selected stocks. Each text of the observations is scanned for these search words. When a word is found, its corresponding ticker name is added to a list of identified tickers. This list is utilized to avoid redundant searches for words, which tickers are already identified, and is ultimately returned as the function's output. To minimize false positives, such as instances where the ticker name appears as part of an unrelated word, the search is constrained by the following regular expression:

```
\b + re.escape(search_word) + \b
```

This ensures that the name is matched as a complete word and not part of any other word, while it is still allowed to be preceded or followed by special characters since it is common to use writings like \$AAPL, #META. Using the flag `re.IGNORECASE` allows the pattern to also match words regardless of their original case. The search word is escaped using the `re` library before being inserted into the pattern to prevent any special characters from disrupting the regular expression.

3.2.2 Preparing Column Names and Values

The following step prepares the inputs to ensure they are in the correct format and state for subsequent processing. Since the script works with different sources of datasets, including news portals and social media, it ensures that all required columns are present and follow the correct naming conventions. Besides that, certain format conversions are applied, such as converting the `created` column to a datetime format. Since the flat structure of the CSV file does not support nested objects, the strings representing the assets in each row need to be parsed into arrays.

The values of these arrays need to be checked for companies having multiple ticker names, as described in Section 2.1.2. Although these represent different stocks with different prices, it can be assumed that the reflected percentual price change on events is almost identical across them. Hence, to maintain consistency, such ticker names are renamed, using a dictionary, to a unified symbol name. It is recommended to select the ticker with higher liquidity for the unified name, ensuring smaller spreads. While this criterion probably does not play a significant role here, for the only company affected in our stock selection, *Alphabet Inc.*, `GOOGL` was chosen as the ticker. To make this approach expandable for future changes, the dictionary is read from a JSON file, whose path can be passed as an environment variable. Finally, any possible duplicate ticker names within each list are removed.

3.2.3 Exploding Targets

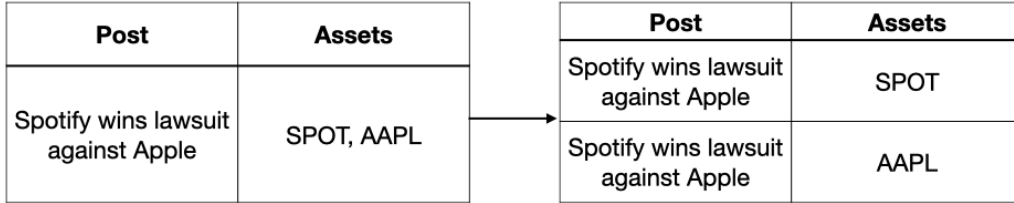


Figure 5: Replication of posts form each associated assets.

As previously mentioned, the content of the news may involve multiple companies. In some cases, the market sentiment scores may align, but often the price movement of the affected stocks differ or even contradict each other. Such an example showing opposing sentiments for each company can be seen in Figure 5. The lawsuit mentioned here resulted in a \$2 billion fine for Apple, significantly impacting their cash flow, and allowing Spotify to advertise their own prices for their services in the app (Chee, 2024).

In order to reflect the sentiment of each asset, such observations must be split into multiple news-asset pairs. This is done by using *pandas* explode functionality, which replicates each element of a list column to a new row, preserving the values of the other columns. For training models, which only consider a single asset for each entity, it can be selected if only posts involving one company should be selected. Instances that do not have a value assigned for the ticker will be removed, as the required input for the computation of the sentiment score is missing. Furthermore, all news-asset pairs not belonging to our selection of stocks will be excluded.

3.2.4 Calculation of Sentiment Scores

In addition to the scraped news articles, a set of target values is required to fine-tune a model on it. The development of the market price, triggered by the news, will be utilized to determine the market sentiment. This approach is automatable, eliminating the need for tedious manual labeling of the instances. It also removes potential human emotional bias, focusing solely on the observed stock price movements.

A potential downside of using price development as an indicator for the sentiment score is that not every positive news title necessarily results in a significant price change. For example, headlines like *"Disney Park welcomes local artists for a weekend for creative workshops"* or *"Apple unveils new eco-friendly packaging initiative"* are surely positive messages, but unlikely to cause measurable changes in stock prices. Considering this, the previously mentioned method for calculating the sentiment score may seem unfitting, as such observations could lead to an inconsistent dataset. Nevertheless, we still decided to utilize this method because the purpose of the model is to benefit from price changes in the stock market, where market sentiment plays a bigger role than the sentiment of the text. It is important to

expose the model to such instances, as they will also appear in real-world scenarios. Therefore, it was decided to keep such instances in the dataset.

The score is expressed as the percentage difference in price development between the time periods t_i and t_{i+1} , based on the closing price at the publication time t_i . The formula used in the implementation relies on the *Rate of Return*, as shown in Equation 5. The function $Close(asset, t_i)$ ideally returns the last known price of the requested *asset* at the exact minute of t_i . However, due to the opening hours of the stock exchange, there will be instances where minute-level price data is missing. To address these gaps, the functions $Close_{Last}$ and $Close_{Next}$ are employed. The $Close_{Last}$ function returns the last known price at the publication time, while the $Close_{Next}$ function provides the next known price of the stock after a certain time period.

$$Return(asset, t_i) = \frac{Close_{Next}(asset, t_{i+1}) - Close_{Last}(asset, t_i)}{Close_{Last}(asset, t_i)} \quad (5)$$

Different lengths for these time periods were evaluated, ranging from one hour to days. When a post contains genuinely new information, it is highly likely that the market, following the EMH (see Section 2.1.7), will reflect this almost immediately in the stock price. Considering this, it seems rational to keep these time periods relatively short. For other types of articles, such as in-depth company analyses and future prognoses, the sentiment of the post may be reflected over a longer duration, potentially up to a year, making it challenging to trace the effect back to the original post.

Another drawback of considering the return of the asset as the only factor for the target value is that this completely ignores other market factors, as described in Section 2.1.6. In order to prevent the models from being influenced by these factors, a variation of the return, known as *excess return* is employed, which isolates the asset-specific performance. After its role model the excess return will be calculated by our `PriceService` as shown in Equation 4.

During the calculation of the CAPM (see Eq 3), values such as the risk-free rate, the beta of the company, and the market risk premium, needed to be determined. The following paragraphs outline the sources and the rationale behind the decision made for obtaining these values.

As the source for the *beta* of a company, *yfinance* is used, instead of calculating the prevailing *beta* at the specified time ourselves. This comes with the limitation that, for historic observations, the current *beta* will be used, which might be outdated, especially for companies that have undergone significant changes.

The benchmark index used for the calculation of the *market risk premium* is configurable, where `QQQ`¹⁹ is set as a default since this resembles the NASDAQ-100 Index[®], which is the same index as we used for the selection of our assets. Its

¹⁹<https://finance.yahoo.com/quote/qqq/> - QQQ ETF resembling the NASDAQ-100 Index[®]

return will be computed over the same period and in the same manner as the stock return (Eq 5).

$$\text{RiskFreeRate}(t_i) = (1 + \text{AnnualRate}_{t_i})^{\left(\frac{\text{period}}{365 \times 24}\right)} - 1 \quad (6)$$

Also for the risk-free rate, the period of the investment needs to be considered, which is done by Equation 6. For this, the annual rate of a bond which is considered risk-free is broken down to the duration of the investment. The rate considered here is the one that was current at the time of the news publication. As a source for the risk-free rate, either a static CSV file containing the 4 weeks coupon equivalent treasury bill rate from 2002 until November 2024, provided by the U.S Department of Treasury²⁰, or the *13 Week Treasury Bill* (^IRX) bond accessed via *yfinance*. The `PriceService` attempts both options as fallbacks, with the execution order specified in a list of its constructor. Both of these bonds have a maturity of less than one year, making them suitable for our short-term investment approach, where trades are not expected to exceed a holding period of one day.

In *Classification of financial markets influencers on Twitter* Almeida and Sabino (2022) also relied on the market return for analyzing the sentiment of tweets of potential influencer and their correlation to the price development. Unlike the previous approach that considers excess return, this study focuses solely on the total return of the stock, without accounting for external factors. For their calculation, they used the timeframe of one day. This choice is likely restricted due to the limitations of the data provided by *yfinance*²¹, which offers *1day* as the most specific interval for historical data. This also has the consequence of not being able to use the exact time of publication for their calculations.

Missing the exact publication time of a news event can significantly affect the accuracy of the market sentiment score. It is important to capture the moment of publication as closely as possible since news tends to reflect quickly in price movements (see Section 2.1.7). The largest price shifts often occur shortly after publication. Therefore, in order to isolate the effect of the news from other factors, it is crucial to have a data source providing fine-granular price data. This data, covering prices within one trading day, are called intraday prices and is often structured in so called candles, containing the first (OPEN), highest (HIGH), lowest (LOW), and last (CLOSE) price and traded volume during an interval. Hence, an ideal source of data for this purpose should meet specific requirements, including price data at a *1min* granularity and access to historical intraday prices.

The service *AlphaVantage*²² offers a developer-friendly and enterprise-grade API for financial markets. By offering free API access to a majority of its endpoints, the Boston-based company has gained popularity, especially under academic researchers. They are one of the only services providing historic intraday data, up to the year

²⁰<https://home.treasury.gov>

²¹Python library offering an interface to Yahoo!®finance's market data <https://pypi.org/project/yfinance/>

²²<https://www.alphavantage.co/>

2000. With available intervals of *1min* up to *60min* it is feasible to get the exact minute as long the market was not closed during that time. The regular trading times are between 9:30 AM to 4:00 PM US Eastern time. By considering also trades during the pre-market and post-market, AlphaVantage extends this range from 4:00 AM to 8:00 PM.

The intraday endpoint returns the price data in chunks, covering all available minutes within a specified month for a given company. For calculating the sentiment score of a news article, only up to two of those minutes will be utilized. The remaining minutes are not required for the calculation, but potentially be of relevance for other instances. To optimize performance and reduce unnecessary requests, a cache is introduced, since the requests are the primary bottleneck in the calculation process. This approach is particularly important in the free tier, where the number of requests is capped.

The intraday endpoint returns the prices in chunks of all available minutes within a specified month for a company. For the calculation of the sentiment score of a news article, only a maximum of two minutes is required for this data. The remaining minutes will not be considered for this calculation, but will probably be of relevance for other instances. To avoid unnecessary requests, a cache is introduced, since the requests are the main bottleneck of the calculation. This approach is particularly important in the free tier, where the number of requests is capped.

In order to handle the delegation between making requests, caching, and retrieving items from the cache, the wrapper *AlphaVantageClient* was implemented around the required endpoint. Its cache is persisted in CSV files, where each asset has its own file, which will be loaded lazily on demand. The method *get_price* will fetch prices on demand if they are not cached yet. By calling *is_cached* the corresponding file of the asset will be read as a *pandas.DataFrame*, if it exists and hasn't been loaded yet. As the cache is indexed by the date, a quick search for the minute of interest should be possible. Keeping in mind that there are not prices for every minute and the function *Close_{Next}* and *Close_{Last}* will also consider alternative minutes, the search selection will be extended by the duration of a weekend, which is the maximum usual time period where data will be missing. When minutes within this selection can be found, the cached data could be used. However, if none of the retrieved candles fall within the same month as the requested minute, making an additional request might still be useful, to gather data from the requested month.

New price data will be obtained by requesting *1min* intervals from AlphaVantage's *TIME_SERIES_INTRADAY* endpoint. After reformatting the retrieved data to the same format as the cache, the updated cache will be persisted to the disk. Since the returned prices are adjusted close prices, the *PriceService* does not need to account for stock splits. However, when using the cached intraday prices over a longer time, it is important to ensure that no recent stock splits have occurred for stocks with previously saved price data. In such cases, the cached prices must either be manually adjusted for the split or fetched again.

After taking those steps, it can now be assumed that a candle close to the requested point in time t_i can be found in the cache. For this the closest row to t_i and its time difference Δt is determined. If Δt exceeds a defined tolerance, a `ValueError` will be returned. For cases where Δt is still tolerable, an alternative point in time will be determined. As defined in Equation 5, the last known price will be identified for t_i and the next known price for t_{i+1} . If the index of these exceeds the boundaries of the `DataFrame`, a request to AlphaVantage will be performed with the expectation of filling these gaps.

3.3 Data Cleaning and Sampling

After the previous steps, the data are ready to be tested for the first training. To achieve more stable and reliable results, additional processing is required to ensure this. In these steps, the data will be purified from invalid values, but also selected by quality criteria. To produce more generalizable and balanced predictions, train-test splitting and different sampling strategies are applied. This workflow is executed by the `data_split.py`, which process is visualized in Figure 6.

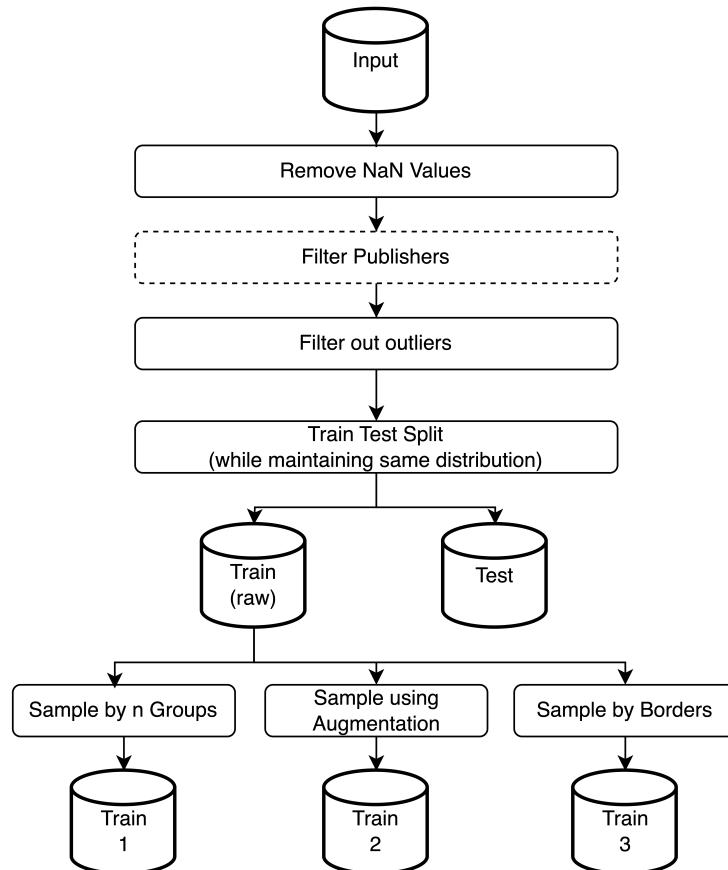


Figure 6: Visualization of `data_split.py` filtering, splitting and sampling the observations.

3.3.1 Data Quality

The quality of data plays a big role in how well a model is able to fit on a task. This goes so far that poor quality of data can prevent the model from learning the task altogether or introduce biases (Budach et al., 2022; Fiona Zhao et al., 2024). In order to ensure a good model convergence the used training data needs to fulfill certain quality criteria.

A key criterion for effective model convergence is the recency of the subject in the article. Since the sentiment score is based on the subsequent price development, it is substantial for the news to have an impact on the stock market. Hence, it is important that the topic of the news is about recent developments and not a report about how the stock price changed due to a prior event. Following the principle of the *Efficient-Market Hypothesis* (see Section 2.1.7), it can be assumed that the new relevant information will be reflected rapidly on the asset price, making the first hour the most significant. Subsequently, even a one-hour delay in publication compared to other sources can disrupt the sentiment score, as it fails to fully capture the price development anymore. Therefore it is advisable to prioritize publishers known for their prompt dissemination of news.

Another criterion relevant to our training purpose is that the meaning of the news should be already recognizable by the title. Since the predictions are based solely on the titles, these should ideally contain as much information as possible. Any information missing there can not be considered by the model. This can be especially fatal for news titles, that obscure, misrepresent, or overdramatize the information. A substantial number of publisher tend to have *clickbait titles* to adapt their articles for *Search Engine Optimization* (SEO). These titles intentionally omit important information and are designed to attract the reader's attention, motivating them to visit the entire article. Consider the following examples of titles that showcase this style of article:

- "10 Stocks You Can't Afford to Miss"
- "Is STOCK a Buy or Sell?"

Since the crawling process for the URLs relies on a search engine, it is prone to picking up these SEO-optimized articles. Additionally, the publisher of such articles often has a high output. As a consequence, this kind of news represents a high share of the crawled data.

When having a large volume of articles, manually evaluating the quality of news titles becomes challenging. To accelerate this process a model could be employed for categorizing the news type or assessing a quality measure. Due to time constraints, an alternative approach was chosen, relying on metadata saved alongside the gathered articles for the data quality selection process. One attribute of the metadata that is most likely to provide the highest information gain for distinguishing data quality, besides the text, is assumed to be the publisher of the article. It can be assumed that a publisher with high-quality articles will generally produce quality

content, and vice versa, suggesting little variation in quality within a publisher. When supplied with a list of publishers known for quality articles, this can serve as a basis for selecting a high-quality dataset. During the crawling process, articles from over 275 publishers were gathered. Evaluating the quality of their articles for each of these publisher can be tedious. Therefore, we prioritized publishers based on the number of articles gathered in this process. The suitability of a publisher was manually annotated by iteratively going through a list sorted by articles per publisher. For each publisher, several samples were viewed and rated by how well their titles represent the actual content. Another criterion for the exclusion of publishers was articles reporting on recent price movements and their origins, as these will no longer affect the price in most cases. This process was stopped after selecting 8 publishers, since all remaining ones had fewer than 200 gathered articles.

Additionally, quality control was conducted based on the calculated target values. For events where a price in tolerable distance could not be determined, NaN values were assigned. These instances are removed to avoid uncertainties in the market sentiment scores. To further enhance data quality and enable more stable predictions, extreme values were removed. This was achieved by calculating the z-score for all observations and excluding outliers with z-scores exceeding a specified threshold.

3.4 Sampling

Initial training results in the early development stage indicated a bias towards neutral observations. This is often the consequence of an imbalanced dataset, as shown in Image 9. The imbalance of this task directs the model to learn that a substantial share of the error can easily be minimized by predicting values around 0.5. As a consequence, the model tends to avoid exploring more extreme values, given that these represent only a small fraction of the dataset, contributing minimally to the overall error. This will lead to a distribution of the prediction values, as it can be seen in Image 8.

To address the issue of imbalanced data, a version of the dataset is created with a more balanced distribution of target values. This was achieved by employing various sampling strategies with the aim of giving minority classes a weight comparable to the majority class. However, as these strategies reduce the overall size of the dataset, additional techniques are applied to augment samples for the minority group.

3.4.1 Splitting the Data

Before any sampling, augmentation, or training is done, it is advisable to put a test set of the data aside. The test does not require to be uniformly distributed, it should rather represent the original population of the dataset. For such a case it might seem sufficient to use the *Simple Random Sampling* method. Due to the high frequency of neutral observation, this might lead to a deviating population for the corner cases. To ensure a similar as possible population, a stratified approach will

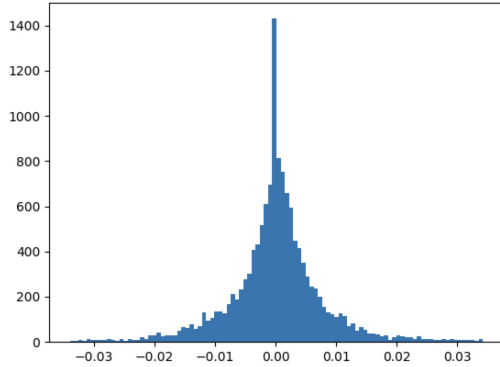


Figure 7: Training Data

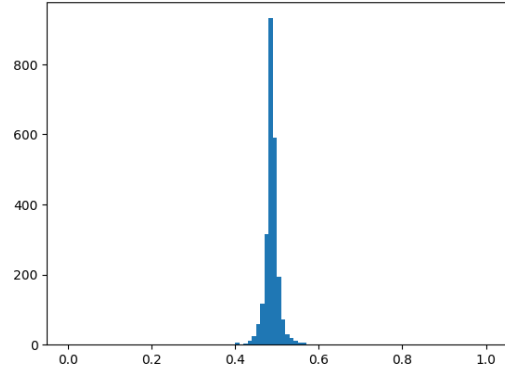


Figure 8: Predictions of the Model

Figure 9: Imbalanced Dataset Problem

be applied here. The range of the target column will be divided into 10 strata, each with equal intervals. Out of every of these strata 10% will be randomly drawn for the test to accurately represent the distribution as much as possible. In addition to the test set, a raw version of the training set will also be saved to the disk before any further sampling is applied.

3.4.2 Creating Sampled Datasets

The stratification is achieved by programmatically defining boundaries based on the target column, which can be specified as a parameter. This process involves establishing truth conditions, determining which x_i from D has a value for column c between two defined boundaries for each boundary group b (Eq 7).

$$\text{condition}_{i,b} = \mathbb{I}(x_i \in \mathbf{D} \wedge \text{lower}_b < x_{i,c} \leq \text{upper}_b) \quad (7)$$

These conditions will be used to annotate each instance with a label indicating each boundary group, allowing for later grouping by these labels. As stated in Section 2.2.1 the implementation of stratification strategies can vary. For this thesis, two different versions were integrated: (1) Automatically dividing the existing range of the target value into n equally sized interval groups, ensuring that the entire range of the distribution is represented. (2) Relying on boundary values that are predefined by the user, allowing a sampling approach based on the sentiment classes and their borders.

When a `sample_size` is defined, these methods will sample from the dataset; otherwise, all instances will be included in the strata, which can then be used for more complex sampling. For the sampling process, it can be specified whether oversampling should be allowed. In order to prevent excessive oversampling, a minimum length for a stratum to be considered, can be specified.

Drawing from the stratified groups can be approached using several strategies. One implemented approach determines the strata with the fewest instances and caps

all other groups to this size. The aim is to have as many observations as possible from minority groups, while not disproportionately representing the remaining. A minimum length can be specified to avoid that an underrepresented stratum, only containing a few instances, leads to a small dataset. A more simple strategy is to randomly draw across the whole raw training set, effectively reducing it to a smaller subset. In this way it can be used as a means of comparison, to evaluate a sampling strategy against a raw version of the dataset of the same size.

When a certain size for each stratum is required, this can be achieved by specifying a desired amount. Often, the minority groups might not have as many samples available as desired. In order to address this problem, oversampling can be used. However, its usage has to be chosen with care, since it can lead to overfitting, through duplicating existing instances multiple times without adding new information. Therefore, another method was integrated into the sampling process, which increases the dataset size. By employing data augmentation, modified instances based on existing data can be introduced, overcoming the limitations of oversampling. Using data augmentation modified instances based on existing can be introduced, thereby overcoming the limitation of oversampling.

To evaluate the most effective sampling strategy, the `data_split.py` script generates multiple dataset versions based on the described techniques. One version is grouped into ten equally sized intervals of the target variable, with samples drawn uniformly according to the size of the smallest group. Another version augments this approach by drawing 300 samples per group, compensating for missing data through augmentation. Additionally, a dataset is created using four predefined group boundaries for the target variable, again sampling uniformly based on the smallest group size. Finally, a randomly sampled dataset of comparable size serves as a baseline for comparison.

3.5 Characteristics of the Dataset

The raw version of the dataset contains over 59,000 news articles from 275 different publishers, including 9,554 articles without specified tickers, 32,438 articles associated with exactly one ticker, and 17,000 articles mentioning at least two tickers. These multi-ticker observations could be particularly valuable for training models to predict the impact of specific news on different assets. The dataset spans articles published between 2004 and 2024, with a focus on more recent articles. The distribution of publication years is shown in Table 1.

Year	2024	2023	2022	2021	2020	2019	2018–2015	2014–2004
Share	46.6%	24.4%	8.9%	5.6%	3.9%	2.9%	5.7%	1.9%

Table 1: Distribution of the Publication Years

The test version resulting from our data pipeline contains 2,860 observations. By employing an LLM-assisted sampling process its range of target values is uniformly distributed. Instances, which are augmented versions of others, are marked with an

augmented flag and can be traced to its original through its identifier. The test set consists of 308 instances and does not contain any augmentations. Its distribution represents the one of its original population. As target variables, the subsequent stock price developments are available over three different periods (1, 12, and 24 hours). They are either specified as *Rate of Return* or *Excess Return*.

4 Methods

This chapter outlines the processes and techniques used to train the model. Besides defining the goals and procedures during fine-tuning, it also covers the implementation and integration of augmentation methods to mitigate overfitting. Additionally, other ML operations such as hyperparameter tuning and the logging of metrics and models are examined. Finally, it introduces custom metrics, which were implemented for the evaluation of the model.

4.1 Data Augmentation

This section provides an overview of the implementation of the augmentation techniques, described in Section 2.2.6. Example outputs of the different techniques can be seen in Table 2. Additionally, the section examines how these methods can be integrated into the data or training pipeline, highlighting the specific advantages they offer at each stage.

Original	Apple is facing delivery issues for the new iPads	
Character level	Appel is facing delivery issuws for the new iPads	
Word level	Synonym replacement	Apple is confront delivery supply for the new iPads
	Random insertion	Apple is issuance facing delivery issues for the new iPads
	Random swap	iPads Apple facing delivery issues for the new is
	Random deletion	Apple is facing issues for the new iPads
Apple is facing delivery for the new iPads		
Document level	Back-Translation	Apple faces delivery problems for the new iPads
	Generative	iPad Delivery Delays Continue to Plague Apple Fans
		Supply Chain Crisis Hits Apple's Latest iPad Launch

Table 2: Various text augmentation methods on the data space with examples at different levels for the example sentence *Apple is facing delivery issues for the new iPads*.

The augmentation on the *character level* is implemented by introducing noise that mimics typical user mistakes while typing on a keyboard. Random words from the titles are selected, and individual letters are altered. This is achieved either by swapping the position of two neighbored letters to simulate typos from fast typing or by replacing letters with others located nearby on the keyboard. For replacement

a keyboard matrix is used, where a random vector is added to the index of the original letter, producing a nearby character.

For the *word level* augmentation, multiple methods were implemented, including *Random Deletion* removing arbitrary words, *Random Swap* changing the position of two random words, *Random Insertion* adding synonyms at random positions and *Synonym Replacement* substituting words with their synonyms. The synonyms are being retrieved from synsets provided by *WordNet*, which is a large lexical database (Miller et al., 1990).

For the phrase level, no explicit implementations are provided in this work, as the text data used here mainly concentrates on titles and short social media posts, which often do not include more than one sentence. This makes the *Phrase Level* for our aspect quite similar to the least granular level: *Document Level*. This level of augmentation is implemented using translation-based and generative approaches.

The process of the back-translation is automated using Neuronal Machine Translation (NMT) models. The University of Helsinki provides with the open-source machine translation *OpusMT* a high quality solution for such a problem (Tiedemann et al., 2023; Tiedemann and Thottingal, 2020). They trained several NMT models using *Marian*, offering over 150 different languages and their combinations. *Marian* is an efficient and free framework written by the Microsoft Translator team, which can be used for training and deploying NMT models (Junczys-Downmunt et al., 2018). Using the *MarianMT* from Hugging Face’s transformers library, these model can be converted to a PyTorch format, enabling their use within the Hugging Face ecosystem. Each of these models is trained for a specific translation direction between two languages. Since the *back-translation* requires both of these directions, two models need to be loaded. As a part of this project, a `BackTranslation` service was implemented that lets users choose the source language and an intermediate language. Based on this, the corresponding *OpusMT* models are automatically located and loaded, during the initialization of the service. These models are then used in the `augment` method to handle the two-step translation process.

It is important to note that the *back-translation* technique does not guarantee variations of the original sentence. In some cases, it can produce the exact same sentence as the original, influenced by the nuances of the text, languages, and translation models.

As described in Section 2.2.6, the generative approach for augmentation can be applied using various methods. Since the process of labeling our dataset is already automated and the missing instances are concentrated at the extremities, we are focusing on the **Data Creation** method to generate additional instances for the minority groups. Møller et al. (2024) addresses the method of *Data Creation* further to compare the performance of models being trained on augmented data with those trained on human-labeled data across various NLP tasks, including *Sentiment Analysis*. Following its approach, the class `LLMAugmenter` was implemented to automate this process. The user can specify which model to use for augmentation and the number of augmented versions to generate per instance. Any `PreTrainedModel`

suitable for *text generation* tasks, which is available on the Hugging Face Hub, can be selected as a model.

As a default, we decided to use `Llama-2-7b-chat-hf`, which is the smallest available size of Meta’s LLM (Touvron et al., 2023). It has a permissive license called the *Llama 2 license*, allowing both educational and commercial use. At the time of its release, it was one of the best performing open-source models, with its largest variant being competitive with ChatGPT as of the publication date.

In order to get this model to fulfill the task of augmentation, its instructions are described in the task prompt. This prompt, concatenated with the instance to be augmented, will be passed as a user message to the model. The task prompt also includes the sentiment score of the instance and a factor defining how many augmented versions should be created. The sentiment score is added to ensure that the model generates instances expressing a similar sentiment. Additionally, there is a system prompt, describing the desired behavior of the model and providing some guidelines, including the rule that the resulting text should differ from the input text. Both prompts are illustrated in Fig 10. After processing these prompts through the pipeline, the augmented versions of the instances will be extracted from the model’s answer using a regular expression. This process can be executed for multiple instances at once using the method `augment_batch`. The new instances will be labeled with the same target and post identifier as their original counterparts. Furthermore, an *augmented* flag will be added, indicating that these are augmented versions.

System Prompt:
 You are an advanced AI writer. Your job is to help write examples of text with a certain sentiment. The examples should have the aim to differ from the input text. Sentiment is represented as a score between 0 and 1, where 0 means it has a very negative effect on the stock market price and 1 means it has a very positive effect on the stock market price.

Task Prompt:
 Based on the following news article title which has a `{label}` sentiment score, write `{factor}` new similar examples in the style of a news title, that has the same sentiment. Separate the texts by newline.
 Text:
`{text}`

Figure 10: System and Task Prompts for the Data Augmentation. Modified version of Møller et al. (2024) prompts.

A technique that operates in the feature space by adding noise was introduced with *NEFTune*. An adaptation of this approach was also implemented into Hug-

ging Face’s `transformers.Trainer`, and can be activated by setting a value for `neptune_noise_alpha` in the `TrainingArguments`. This factor will determine the magnitude of the noise added to the embedding, normalized based on its dimensions. The generated noise will be spread evenly between the positive and negative limits specified by the magnitude.

4.1.1 Integrating Data Augmentation into the Data Pipeline

Data augmentation can be integrated at various stages of the process. By integrating the augmentation process into the data pipeline, a dynamic approach can be implemented, taking advantage of the knowledge of the entire dataset. Such a method enables filling data in certain ranges of the target value, achieving a more uniform distribution. For this reason, the augmentation step was embedded into the sampling process. In this process, our data is already present in stratified population groups based on the market sentiment score. Depending on the sizes of these groups, a desired size will be determined, ensuring consistency over all groups.

Algorithm 1 Get Samples By Augmenting

```

function GETSAMPLESBY AUGMENTING(groups, desiredSize, factor)
  augmenter = Augmenter(factor)
  samples = []
  for each group in groups do
    available = min(desiredSize, len(group))
    remaining = max(0, desiredSize - available)
    samples.add(group.sample(available))
    if remaining > 0 then
      amountToAugment = ceil(remaining/factor)
      amountToAugment = min(available, amountToAugment)
      samples = group.sample(amountToAugment)
      augmentedSamples = augmenter.augment(samples)
      amountAugmented = min(len(augmentedSamples), remaining)
      augmentedSelection = augmentedSamples.sample(amountAugmented)
      samples.add(augmentedSelection)
    end if
  end for
  return concatenate(samples)
end function

```

The algorithm shown in [1](#) will evaluate for each group if and how many more observations are needed to reach the desired size. For creating new versions of the instances an *Augmenter* was initiated with a defined factor f , which specifies how many new altered variations should be created for one instance. Should a subset require additional observations, a maximum of $1/f$ of the amount of remaining instances will be sampled of this group, in order to augment them.

Besides being able to exploit the global knowledge in a dynamic approach, it also has the advantage of being more efficient. Given that there is no training happening in parallel, no additional model needs to be loaded besides the one used for the augmenting. This leads to fewer conflicts in the usage of the available memory and a faster execution of the training steps, compared to integrating the augmentation into the training pipeline. It is also possible to generate multiple augmentations of one instance with a single prompt, which can later than be redistributed in the dataset. Once the pipeline is executed and the dataset saved, there will be no more augmentation overhead for the training, until another dataset shall be created.

The downside of this approach is, that the model will see the same instances with every epoch since the augmentations already happened in the data pipeline. This might eliminate one of the desired effects, which is preventing the model from memorizing the dataset, instead of generalizing.

Since a train-test split has not happened yet, it is important to ensure there will be no overlapping of the augmented instances and original versions between the train, validation, and test set. To guarantee that, a custom *train_test_split* method was implemented, which splits the data into two non-overlapping subsets, based on a specified column. For this case, a post identifier is being used, which is unique for every original instance, while the augmented versions also use the identifier of the original posts.

4.1.2 Integrating Data Augmentation into the Training Pipeline

When integrating the augmentation into the training pipeline, the process will happen Just-In-Time during the training, for every learning step. For this the augmentation needs to happen, between the processing of the training batches. Trying to intervene during this step in the `transformers.Trainer` class would not work for most methods, since the text is already tokenized at this stage. This would eliminate all methods that operate on sentence or word level and do not use this tokenization. To enable it to work for all augmentation methods, it needs to be implemented at a stage, where there was still no tokenization applied to the instances. Before implementing the data augmentation into the training pipeline, the tokenization already happened during the preprocessing of the loaded dataset. A stage during which intervention is still possible would be when the `DataLoader` retrieves the instances from the dataset. For this a custom class of `torch.utils.data.Dataset` was implemented, with the added functionality of augmenting and tokenizing its items, called `TextAugmentationDataset`. Besides the dataset and the name of the base model, which is required to ensure the correct tokenization, it takes an augmentation type and probability. It will be initialized with an yet not tokenized text dataset. The tokenization takes place at the end of the `__getitem__` method, after the original text was passed to the method `augment_text`. This method might return an alternated version of its input, depending on the specified probability threshold. Based on the selected augmentation type, the transformation will be delegated to the according function.

While this integration of the augmentation does not increase the amount of training instances per epoch, it still offers the advantage of generating different versions of the instances with every epoch. This forces the model to learn more generalized patterns rather than memorizing the original dataset since there can be always new feedback, based on unseen data, to adjust the weights. When the augmentation method is properly configured, its effect can persist indefinitely, ensuring that new, unseen data continues to be available at each epoch, even after countless iterations. The biggest drawback is that it can be rather resource-expensive, depending on the applied augmentation method, which slows down the training process. While conventional methods might only add a marginal computation effort, it is especially demanding for AI-assisted methods like BackTranslation and Data Creation by LLM Prompting. Most LLM-assisted methods may already conflict with the remaining available memory occupied by the weights of the loaded training model. Including the inference of the augmentation into the training process, would increase the training time significantly.

In order to fully leverage data augmentation’s ability to overcome overfitting, a hybrid approach was implemented, integrating augmentation into both the data and training pipeline. Considering that the disadvantages of the integration into the training pipeline do not apply to the most conventional augmentation methods, these strategies will be implemented into the training process. While the resource-intensive LLM-assisted methods already take place during the data pipeline.

4.2 Fine Tuning

In this subsection, all procedures and considerations involving the *fine-tuning* will be described. Fine-tuning is the part of the training, where a pre-trained model is specialized by further training on a *downstream task*. This downstream task often diverges from the text generation task learned during pre-training. Determining the stock development can either be done through a classification task, which categorizes the news to be followed by a bearish, bullish, or neutral market reaction, or a regression task aiming to measure the impact of the market movement.

The target value of our downstream task is a market sentiment score, which expresses the reaction of the market. The reaction is measured by investigating the subsequent price movements of the involved stocks. Even though the sentiment score could have been converted to classes by applying thresholds, it was decided to train the model as a regression task directly on the target value for several reasons. Firstly, the target value itself is a continuous number, providing the ability to differentiate between stronger and weaker price changes. Another challenge would have been finding the right thresholds for dividing the sentiment scores into the sentiment classes, as there is no clear boundary between neutral and price-affecting observations. Using a regression prevents cases falling into uncertain territory between being classified as bullish, bearish, or neutral from affecting the model convergence.

In order for *BERT* to be fine-tuned for a specific downstream task, adding a single additional layer is sufficient. For extending the model with the additional

```

class BertForSequenceClassification(BertPreTrainedModel):
    def forward(self, ..., labels=None):
        ...
        if labels is not None:
            if self.num_labels == 1:
                # We are doing regression
                loss_fct = MSELoss()
            ...

```

Figure 11: Implementation of regression functionality in BERT’s Sequence Classification. Source: https://huggingface.co/transformers/v3.0.2/_modules/transformers/modeling_bert.html#BertForSequenceClassification

layer, *Hugging Face* offers various classes for different task types in their *transformers* library (Wolf et al., 2020). The one that we employ for our training is the `AutoModelForSequenceClassification`²³. Using *Hugging Face*’s `AutoModel` classes enables a more modular implementation for future models. Inside the constructor, it delegates the initializing to the implementation of the specific model. While the name suggests a classification-only use, this class is also widely utilized for regression tasks. This mode is limited to univariate regression and can be enabled by setting the number of labels to 1. The `forward` method, as demonstrated in the Code Snippet 11, sets the loss to a *MSE* in that case, commented as regression.

Under the approach of scaling it is understood to transform columns of the dataset into a similar range. This method can be applied to the input data as well as the output variables. While the processing of the input text is already done by BERT’s tokenization and embedding, there are several reasons why scaling the target values can be advantageous. An advantage of scaling the target value is that it enhances the comparability of results across different models and datasets. When dealing with datasets that lack a defined maximum, the individual maximum can have a high impact on the magnitude of measures like the mean squared error. While having a scaled target value in the range of 0 to 1, the highest possible error is capped at 1, assuming the model’s predictions do not exceed the range of the scaler.

A more crucial reason to use a scaler is to align with the output activation of the models. Activation functions are used throughout the entire model to transform the output of each layer to a certain range, to ensure a more stable training process (Nair and Hinton, 2010). Several activation functions, commonly used for the logits of the model, fit within a 0 to 1 range. In order to fully exploit the available range and avoid having target values that are impossible to predict with the chosen activation layer, it is necessary to apply normalization to the targets. While `BertForSequenceClassification` does not apply any activation to its logits by default, it might be beneficial to apply one depending on the task type.

²³https://huggingface.co/transformers/v3.0.2/model_doc/auto.html

Additionally, there are other factors by which scaled target values can enhance the model’s convergence speed and accuracy (Bishop, 1995). One reason is that the normalization prevents the calculated gradients from becoming particularly large or small. It may also be favorable for the model layers to learn the concept of the target value when it is on a similar scale as the inputs. Consequently, we utilized the `MinMaxScaler` to scale our target values, with the goal of achieving optimal model performance and comparability across different datasets.

4.2.1 WandB as MLOps Platform

In order to monitor the variations in model performance during training, it is essential to track and visualize the results. One tool that offers such functionality is *Weights & Biases* (WandB)²⁴. *WandB* is a MLOps platform that can be utilized throughout the entire lifecycle of a model. This spans from tracking initial training experiments and hyperparameter tuning to deploying models for inference. *WandB* provides cloud storage for saving results, enabling the collection of training data from different machines without the need to set up a centralized service. This is particularly useful for our case, as it provides the ability to view the outcomes on the local machine via the web interface, while the training is executed on dedicated servers. On this web service, all runs can be listed, filtered by their configurations, and sorted by their metric values. Additionally, the metrics of multiple runs can be visualized together in a single graph, providing a useful tool for comparison.

In order to incorporate WandB’s experiment tracking into the training process, its Python library needs to be installed, which is also used for the authentication of the account. After setting the project name in the environment variables and configuring `report_to` in the `TrainingArguments`, the `Trainer` will automatically log all metrics which are returned by the `compute_metrics` method. When it is necessary to custom configurations along with the models and trainer’s metrics and configurations, this can be achieved by passing these configurations when initializing *WandB*.

We were particularly interested in analyzing the model’s performance on evaluation data in relation to its results on the training data to gain better insights into the impact of overfitting. However, by default, the `Trainer` class does not execute the `compute_metrics` method on the training data, effectively logging only the loss metric during training. Therefore, we created a `CustomTrainer` inheriting from Hugging Face’s `Trainer` class, which intercepts the training process. To avoid causing any overhead during inference, we manipulate the `compute_loss` method, where the predictions on the training set are generated. Since evaluations on the train batches would result in fluctuating measure values, we only gather the predictions and their ground truth values without computing any metrics at this stage. For the actual metric computation, we override the `evaluation_loop` method. Here the before-gathered values are passed to the `compute_metric` method and the results are

²⁴<https://wandb.ai/> - WandB’s MLOps platform

logged before calling the parent class’s method to perform the usual evaluation. To ensure consistency, the evaluation strategy is set to `epoch` in `TrainingArguments`, guaranteeing that each metric calculation includes the same set of instances.

After completing the training process, the updated weights are saved to *WandB*’s Model Registry, so that they can be deployed and re-used for additional experiments. This functionality is natively supported by the `transformers` library. However, starting with version `v4.46.0`, changes in the library inadvertently broke the integration of *WandB*’s callback for saving models at the end of the training. To address this issue, we submitted a Pull Request²⁵ adjusting the callback to be compatible with the updated `Trainer`. As a result, users can utilize this feature with versions of `transformers` either prior to `v4.45.0` or from `v.47` onward.

4.2.2 Hyperparameters

One of BERT’s contributions is that through its architecture and extensive pre-training often only one additional layer is required for the fine-tuning on the downstream task. That already eliminates the need for adjustments for a lot of parameters for the model structure. However, there are still several other parameters for the training process to be tweaked to optimize the results.

One such parameter is the **number of epochs**, which defines how many times the training set is passed through the model during the training. Increasing the number of epochs provides the model with more opportunities to learn from the dataset, but can also lead to overfitting (Murphy, 2012).

Each epoch consists of several batches. The amount of batches per epoch depends on the **batch size**, which defines how many instances are presented to the model before it executes the *backpropagation*, updating its weights. Due to the parallelization capabilities of GPUs, a larger batch size might result in faster training but also requires more memory. Smaller batch sizes on the other hand can potentially improve the model’s generalization by providing more regular updates.

Using the **learning rate**, the pace of the training can be controlled, defining how strong the model’s weights should be updated to the direction determined by the optimizer based on the loss function. Employing higher learning rates accelerates convergence but can also cause the model to overshoot the optimal solution. While lower learning rates mitigate this problem by making smaller, more precise updates, they still bear the risk of getting the model weights stuck in a local minimum, from which the momentum is too weak to escape. In order to benefit from both of these aspects, the learning rate can be applied dynamically. By default, the `TrainingArguments` set the scheduler of Transformer’s `Trainer` to decrease the learning rate linearly towards zero over the course of the training process. In this way, the initial training phase can be used for exploring global minima, before settling into a local minima using a more precise learning rate.

²⁵<https://github.com/huggingface/transformers/pull/34720>

Since the initial weights of the classification head are random, they may not be stable enough to make strong training adjustments based on them right away. Therefore, it is advisable not to start immediately with the specified learning rate. The warmup phase schedules the training to begin with smaller, more stable updates. Using the **warmup ratio**, the number of training steps used to build up the learning rate from zero to the desired level can be defined. While starting with smaller updates, the model has more time to generalize from the training data before making large adjustments. However, if the warmup ratio is set too high, the model might have trouble converging.

The **weight decay**, also called *L2 penalty*, is a regularization technique used to prevent overfitting (Krogh and Hertz, 1991). To achieve this, the *L2 norm* of the model weights, scaled with the weight decay, is added to the loss function. Through this way the optimizer is encouraged to keep the weights small, avoiding exploding gradients. This is a way of keeping the model simple, without the need to make cuts to the model’s parameter count. Since the complexity of the model is often the reason for overfitting, applying a weight decay to the loss function can enhance the model’s ability to generalize.

The **augmentation type** is a parameter in our `TextAugmentationDataset` that specifies whether and which data augmentation method should be applied during training. All listed methods in Table 2 can be selected, with the exception of the *Generative* approach, since it is too resource-intensive to utilize an additional LLM model during training. The probability of applying these augmentations during training can be controlled through a separate parameter **augmentation_prob**, allowing fine-tuning of the augmentation frequency.

The magnitude of the NEFTune augmentation on the feature space can be controlled using the **neftune_noise_alpha** parameter (Jain et al., 2023). This parameter, already implemented in Hugging Face’s `Trainer`, adjusts the level of noise added to the embeddings, helping the model to become more robust to variations in its learned instances.

For most of these parameters, there is no universally optimal or correct value. Therefore it is essential to determine which parameters are most suitable for a specific task and dataset. This process of finding the best parameters in order to optimize a specific metric is called *Hyperparameter Tuning*. Finding these parameters manually can be quite tedious, which is why various frameworks exist to automate this purpose. In this study, we decided to use `WandB`’s hyperparameter search functionality called *Sweeps*²⁶. One reason for the choice of this tool was to maintain consistency in our MLOps workflow since we are already using `WandB` for *Experiment Tracking*. Additionally, we found `WandB`’s interface and visualization capabilities to be more intuitive and informative compared to other MLOps tools we had previously explored, which further supported our decision to adopt it for hyperparameter tuning.

The search space can be defined by specifying **parameters** in the sweep config. There you can list every hyperparameter, which should be optimized, along with

²⁶<https://docs.wandb.ai/guides/sweeps/> - Sweep Docs of WandB

the ranges of values to be considered during the tuning process. These can either be specified by providing constant predefined values, or by defined ranges. The distribution of such a range can be either a normal distribution, defined by its mean and standard deviation, or a uniform distribution between two specified values, or some transformations of these distributions.

Besides the search space, it is also required to define a method, for choosing from the possible values of the parameters. Taking into account the size of the search space and the overall goal, multiple strategies can be considered.

One of the simpler approaches is called *Grid Search*. By iterating over all hyperparameters and their defined range of values, it ensures that every possible combination is evaluated. Depending on the size of the search space, the execution time can become exponentially large. Assuming we would like to explore six different values for each of the above stated parameters, this would already result in 262,144 training runs. Additionally, its iterative nature could result in the evaluation of optimal hyperparameter ranges occurring at a relatively late stage. Therefore, it should only be considered if the whole sweep can be expected to be executed in a reasonable time.

A more suitable method for larger search spaces is *Random Search*. This strategy will randomly draw from the specified distribution of the hyperparameters. In this way, it is able to explore many diverse combinations in a short time. It can be helpful for getting a quick overview of how each parameter might influence the performance of the model. However, it does not consider these results in order to choose parameters that achieve better training results.

An approach that makes informed decisions based on knowledge about its previous parameters and their results is known as *Bayesian Search* (Dewancker et al., 2015). While the parameters for the first run are still chosen randomly, it will use a probabilistic model to determine the parameters for subsequent runs. For these runs, it will balance the trade-offs between *exploration* and *exploitation* based on an acquisition function. During the exploitation phase, the goal is to maximize a specified metric by selecting the parameter values within regions, that have demonstrated a high likelihood of yielding promising training results. In order to prevent the strategy from settling on initial parameters that appear optimal but might not truly be the best, the exploration phase strives to discover new values. With the aim of maximizing the knowledge gained from these new points, its probabilistic model determines parameter values within regions with high uncertainty. In this way, the *Bayesian Search* is able to reduce the search time, while employing a more sophisticated approach of finding the best suitable parameters in fewer steps.

When using *Bayesian Search*, it needs to be defined in the sweep config, which metric is supposed to be optimized by the scheduler. For the name of the *metric*, every measure that will be logged can be specified. In order to prevent overfitting it is advisable to choose an *eval-metric*. Whether its value should be minimized or maximized is defined by the *goal* attribute. For example, minimizing metrics like *Mean Squared Error* is common, while metrics like *Accuracy* should strive for the

highest possible value. It is also possible to define a *target*, which functions as a stop criterion to not start any further runs once it is reached.

The integration of this process is done by passing the defined sweep configuration to a *WandB* agent along with an objective function. The task of the objective function is to evaluate a resulting score for a certain run and its defined parameters. The score must be logged in this method and should match the defined metric from the sweep configuration. Since Hugging Face’s `Trainer` comes with built-in *WandB* support for logging, there is no need to implement it manually. The only step necessary to integrate it was to retrieve the specific config of this run from *WandB*. This config is being used to dynamically set the parameters of the `TrainingArguments` and our `TextAugmentationDataset`. A frequent problem that occurred during testing was that runs consistently crashed after a while due to *out-of-memory* exceptions. When running a resource-intensive process over a long time it is important to take precautions to avoid memory leaks. Therefore, we ensure the CUDA cache is cleaned after deleting the previous model and before starting with the new run.

4.3 Metrics

The fine-tuning of the model happens on a regression task. Common metrics there are *MAE*, *MSE*, and R^2 , which were also used for the optimization of the model. When it comes to comparing the model, these metrics are often less suited, especially when using varying datasets. Depending on the task they can be less feasible to interpret. Hence, some custom created metrics were added, in order to make the results more comprehensible. All of the following listed metrics were implemented by inheriting from the `torchmetric.Metric` class. Its method `update(preds, target)` processes and stores relevant information out of the prediction and its ground truth values. The actual value will then be calculated by calling `compute`. Previously stored information can be discarded with `reset()`.

4.3.1 DirectionAccuracy

This metric describes how accurately the model can distinguish between falling and rising prices based on the corresponding posts. For the sake of simplicity, a post is classified as reflecting rising or falling prices, if its return is above or below 0%. When training models on a scaled version of the target data, the corresponding scaled value of 0 must be specified when initiating the metric. The measure is then calculated by counting the correct and total predictions and dividing them (Eq 8).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{\text{correct}}{\text{total}} \quad (8)$$

A prediction is counted as correct when it has the same true value for $\hat{y} > zero_{scaled}$ as its target for $y > zero_{scaled}$.

4.3.2 SentimentAccuracy

The *SentimentAccuracy* is similar to the previous one but considers an additional class representing neutral cases. Since the three classes bearish, neutral, and positive can not be represented anymore as a boolean value, the predictions and targets will be categorized into a numerical value, representing the three possible classes. The classification works by comparing each value in the tensor to specified boundaries (Eq 9). Instead of the zero-point, there are now two threshold values, defining the boundaries of the neutral cases. Unlike the fixed zero value used in *DirectionAccuracy*, the neutral boundary values for *SentimentAccuracy* do not have a fixed point and must be selected based on the specific context or dataset.

$$\text{classify}(t_i) = \begin{cases} 0 & \text{if } t_i < \text{threshold_left} \\ 1 & \text{if } \text{threshold_left} \leq t_i < \text{threshold_right} \\ 2 & \text{if } t_i \geq \text{threshold_right} \end{cases} \quad (9)$$

4.3.3 ProfitSimulation

The main purpose of the model is to support making profitable trading decisions, which is why the *ProfitSimulation* metric was introduced. This metric simulates trading actions based on the model's predictions and calculates the profit using the ground truth data.

If the model is trained on scaled values, the scaler needs to be passed during the initialization of the metric. It will then be used to inverse-transform the values, restoring them to their original state. Additionally, a threshold to determine when a trade should be considered and a base amount to define the trade size can be specified as parameters.

A prediction counts as bullish or bearish, when its absolute value exceeds the defined threshold, indicating a position should be opened (Eq 10). This threshold is specified as a percentual return and is set to 1% as a default. Theoretical profit when predicting the right direction, is calculated by multiplying the target value with a base amount (Eq 11). When the excess return is being used, it needs to be considered that it does not express the actual followed return, but rather the expected price movement isolated from other extrinsic factors. Whether this amount should be added as a profit or subtracted as a loss, will be determined by checking if the prediction direction is the same as the one of the target value (Eq 12 and Eq 13). The overall simulated profit is then evaluated by multiplying all these variables as shown in Equation 14. The metric itself is expressed as an absolute number, representing the total profit.

$$\text{shouldOpen} = |\text{preds}| > \text{threshold} \quad (10)$$

$$\text{theoreticalProfit} = |\text{target}| \times \text{baseAmount} \quad (11)$$

$$\text{rightDirection} = (\text{preds} \geq 0 \wedge \text{target} \geq 0) \vee (\text{preds} < 0 \wedge \text{target} < 0) \quad (12)$$

$$\text{profitMultiplier} = \begin{cases} 1 & \text{if rightDirection} \\ -1 & \text{if not rightDirection} \end{cases} \quad (13)$$

$$\text{simulatedProfit} = \text{theoreticalProfit} \times \text{profitMultiplier} \times \text{shouldOpen} \quad (14)$$

It is crucial to note that this metric does not guarantee a model's profitability, since it does not consider slippage (see Sec [2.1.5](#)).

4.3.4 AvgProfit

One limitation of *ProfitSimulation* is that its magnitude is influenced by the size of the dataset it is evaluated on. When there are more potential trading opportunities and the metric is expressed as a cumulative total, a higher number of simulated trades will most likely result in a higher total profit.

Therefore the metric *AvgProfit* was introduced, building upon the calculation of *ProfitSimulation*. Unlike *ProfitSimulation* it determines the average profit of the total profit. For building this mean, the amount of opened positions will be considered, rather than the total amount of observations (Eq [15](#)). Instead of expressing the profit as an absolute value, its relative value is derived, which removes the influence of the base amount.

$$\text{averageProfit} = \frac{\text{simulatedProfit}}{\text{openedPositions} \times \text{baseAmount}} \quad (15)$$

These adjustments make the *AvgProfit* more suitable to be evaluated across different datasets.

5 Experiments

This section presents all experiments conducted to determine the optimal configurations for the dataset, model, and training parameters. Based on their results, the reasoning behind the chosen values and decisions is explained. Furthermore, the final performance of the model is evaluated on our own dataset.

5.1 Hyperparameter Tuning Results

To find the most suitable configuration for the training processes used in the experiments, several *Sweeps* were run on different datasets. All of these sweeps used the same search space, whose boundaries and distributions are shown in Tabular [3](#).

Hyperparameter	Range/Options
learning_rate	1×10^{-5} to 5×10^{-2} (log-uniform distribution)
per_device_train_batch_size	{4, 8, 16, 32, 64}
num_train_epochs	{3, 6, 12, 24, 48}
neftune_noise_alpha	{None, 1, 2, 5, 10, 15, 20, 25}
train_augmentation	{None, SynonymReplacement, RandomInsertion, RandomSwap, RandomDeletion, Noise, BackTranslation}
train_augmentation_prob	{0.1, 0.2, 0.3, 0.5, 0.6, 0.7, 0.8, 0.9}
weight_decay	{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.8, 0.9}
warmup_ratio	{0.1, 0.15, 0.2, 0.25, 0.3}

Table 3: Hyperparameter Search Space

It was decided to optimize the hyperparameter (HP) tuning on a different metric, than the one employed for fine-tuning. Utilizing *MSE* was beneficial for training, as it ensured more precise prediction. However, it is challenging to tell if an improvement in the error means, that the model is able to distinguish better between the cases, or if the predictions are simply closer to the target value, without significantly impacting the outcome. Additionally, the requirement for the metric to be differentiable for gradient-based optimization was a decisive factor. Since this rule is no longer applicable for the hyperparameter tuning, it is possible here to select a metric that is more meaningful for trading decisions. Therefore, it was decided to perform the optimization on the **ProfitSimulation** metric.

To ensure that the sweep does not select the parameters based on overfitting to the training data, the metric is evaluated on a different set. It is important here that the evaluation is run on the validation set since optimization on the test can lead to overfitting the hyperparameters.

Figure [12](#) presents the results of the hyperparameter tuning can be seen, as visualized with *WandB*. Although the tuning process included parameters for the training

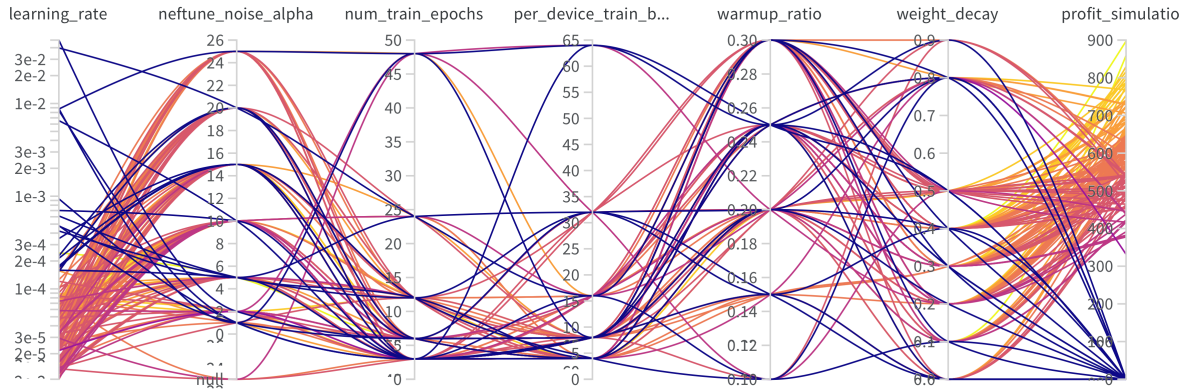


Figure 12: Overview of the Hyperparameters and their results

pipeline augmentation, they were omitted in this figure and will be further investigated in Section 5.6. The hyperparameter tuning was performed on multiple versions of the dataset, varying in sampling and augmentation strategies. This section focuses on the results obtained from the dataset delivering the best results, which were achieved through sampling and generative augmentation. Its results were analyzed by considering the run with the best performance and examining the mean accuracies, grouped by the unique values of each hyperparameter.

A majority of the learning rates fall within the range of $2e - 5$ to $3e - 4$. While a few exploratory runs with higher learning rates were attempted, they did not yield promising results. Nonetheless, the run achieving the best results has a learning rate lying slightly above the majority. The number of training epochs settled to values below 12 epochs, demonstrating BERT’s ability to generalize effectively in a relatively short time. The reason for avoiding longer training periods is likely indicating overfitting. The best results could be found in runs with 6 epochs.

For the batch sizes, a clear trend emerged when analyzing the mean accuracies for each value, with smaller batch sizes consistently yielding better training results. However, with a batch size of 32, the best result from the sweep run approaches the upper limit of the search space for batch sizes. To ensure this batch size was not chosen due to an insufficient number of runs exploring other options for the given hyperparameters, an additional hyperparameter grid search focussing on batch size was started, keeping all other values fixed. This sweep confirmed 32 as the best batch size configuration while achieving significantly better accuracies (64.71%) than all lower tested batch sizes (48.86%).

The evaluation of different warmup ratios exhibited that building up the learning rate during the initial epochs leads to improved results. Increasing the learning rate over a period of 20% of the total epochs achieves a performance peak, after which the results begin to gradually decline. Analyzing the results of different weight decays showed that any value lower than 0.8 yields better results than not using

weight decay at all. The best performance was achieved with a weight decay of 0.4, starting from then the penalty outweighed the effect of regularization leading to poor results.

For the alpha value of added NEFTune noise, no definitive conclusion could be drawn. While values such as 10 and 20 deliver good results, it could not be proven that they outperform the scenarios with no added NEFTune noise.

5.2 Setup

For all other experiments besides the hyperparameter tuning, the models were evaluated on a separate test set, which was saved during the creation process of the train set and does not contain any augmentations.

For better reproducibility of the experiments, all training sessions relevant and shown in these evaluations were trained with manually set random seeds for the model. To keep track of these seeds, they were logged along with the other metadata. For each experiment involving training three runs were executed, each with unique seeds and a common group identifier. This approach provides a more reliable estimate of the model’s performance, eliminating any uncertainties or overfitting due to randomness. The group identifier can then be used to summarize the metrics of the individual runs and display them in a box plot using *WandB* visualization.

For each dataset, there will also be a comparison with FinBERT in the same instances. Before starting any training on a new dataset, the fine-tune script automatically checks for an existing evaluation of *FinBERT* for this dataset and initiates one if necessary.

5.3 Scaling

While the range of the target value does not strictly need to be between 0 and 1, it can be beneficial for various reasons, as discussed in Section 4.2. To evaluate its importance, some experiments were executed with the same model configuration on the same dataset, with the only difference being the scaling of the target value and varying model seeds. The results, as shown in Figure 13, demonstrate that training with scaling consistently achieved notably better performance compared to models without scaling.

The scaling of the target values also provides more stability during the training, which can be seen when comparing the variances in the average profit metric between the two conditions. Without scaling, the average profit was 0.50563% with a standard deviation of 0.098678, while with scaling, the average profit increased to 1.268% with a notably smaller standard deviation of 0.036182.

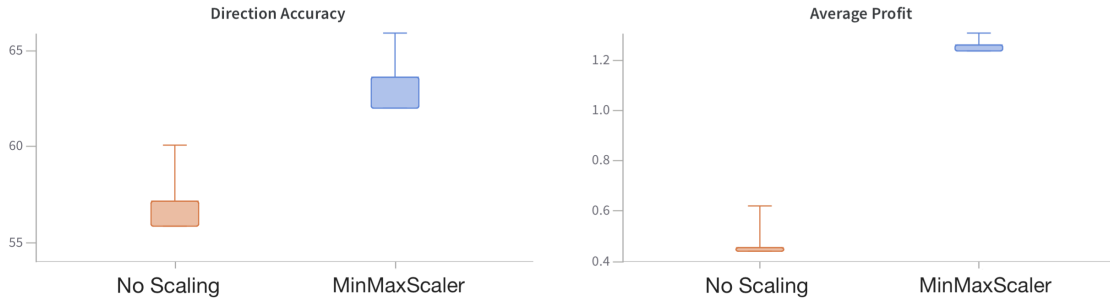


Figure 13: Comparison between Training with and without Scaled Target Values

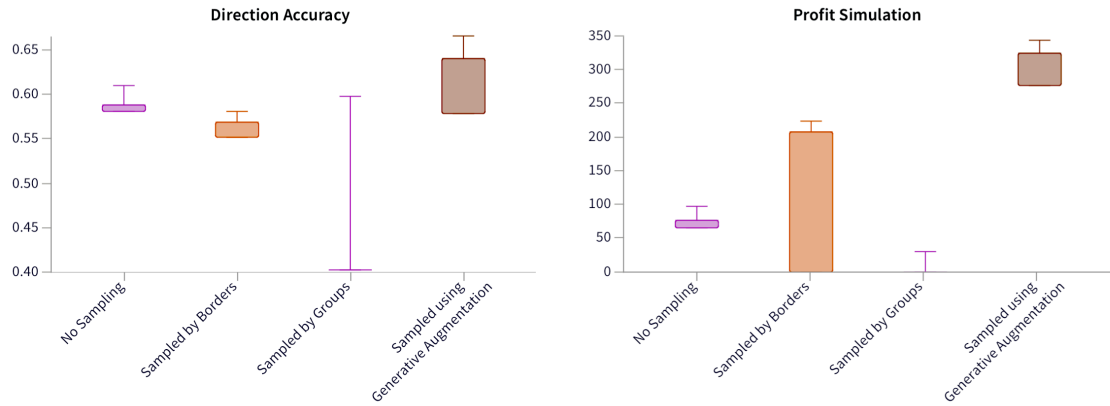


Figure 14: Evaluation on different Sampling Approaches

5.4 Sampling Methods

To evaluate the best-performing sampling method, a hyperparameter search was conducted for each dataset, to prevent any datasets from achieving better performances due to more favorable parameters for its pattern. For the best result of each of these sweeps, there were three models trained with varying seeds. The training datasets were created through various sampling techniques, including no sampling, stratified sampling by groups, stratified sampling by borders, and stratified sampling assisted by augmentation. The process of generating these datasets is detailed in Section 3.4.2. Their performance, evaluated on a separate test set that remained consistent across all sampling versions, is shown in Figure 14.

When being trained with the dataset sampled by 10 groups, worse results were achieved compared to the baseline, without any stratified sampling. The underperformance can be attributed to the insufficient amount of training observations, as the size per stratum conforms to the length of the smallest minority class.

The sampling approach using borders as a stratification constraint, divided the target variable into four groups. The outer groups are defined by the threshold also used in *Profit Simulation* to open a position, while the inner groups are separated

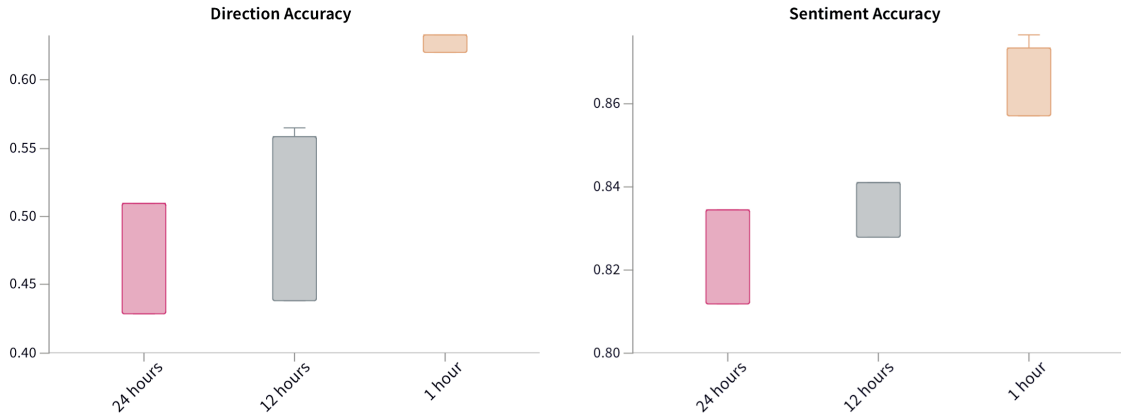


Figure 15: Comparison between Different Periods for the Target Values

by the neutral value of zero. While this strategy led to improved results in the *Profit Simulation*, the model’s predictions remain too unstable for real-world application.

The sampling assisted with LLM-generated augmentations yielded the best overall results. At first glance at the *Direction Accuracy*, its performance appears comparable to the *No Sampling* approach. However, when evaluated using the *Profit Simulation*, it outperformed the other strategies, demonstrating more consistent and reliable predictions, particularly for the minority classes. This improvement is particularly evident in the range where predictions are most critical for profitability, highlighting the potential of generative augmentation to enhance both stability and profitability.

5.5 Time Period for Price Development

To determine the most effective time period for calculating the returns, different periods were evaluated. For this three different versions of the dataset were created, differing in the period length used for the computation of the target variable. Using our standard approach to create the test set would lead to varying versions of it, as the process depends on the target variable. Therefore, an alternative approach was implemented, splitting the data before considering any characteristics of the target variables, to ensure comparability across the datasets. The subsequent filtering and sampling process remained the same.

This procedure was evaluated for the intervals *1 hour*, *12 hours*, and *24 hours*, with the results presented in Figure 15. The evaluation did not consider any intervals beyond *24 hours*, since following the EMH (see 2.1.7) the effects of the information should already be priced in by this time, and longer intervals could introduce noise from external factors.

Examining the accuracies achieved across different configurations reveals a decreasing trend as the time window for the calculation of the target variable increases.

This trend applies for the *Direction Accuracy*, as well as the *Sentiment Accuracy*. The superior performance of the *1 hour* configuration compared to longer windows can likely be attributed to the EMH. Longer time windows increase the likelihood of capturing external factors unrelated to the news event, introducing additional noise into the data. This is also reflected in the larger standard deviations observed for the *12 hours* and *24 hours* windows (0.0574 and 0.0713, compared to 0.0075 for the *1 hour* window).

5.6 Data Augmentation Methods

To evaluate the impact of the different augmentation methods in the training pipeline, over 90 runs were conducted varying, in their configuration of seeds and augmentation type and probability. The dataset and all other hyperparameters were kept consistent with the previously best-evaluated results. The means and variances of the direction accuracy and profit simulation for the resulting runs of each augmentation type are presented in Table 4. *Synonym Replacement* is the only method achieving, on average, slightly better results than using no augmentation, which could also be attributed to randomness, given its subtle deviation. All other traditional text augmentation methods either yielded similar or even poorer results.

It could be argued that there is only minimal potential for improvement through further augmentation methods since the dataset already incorporates generative augmentation. However, this pattern can also be observed in sweep runs on datasets without any previous augmentation. Therefore, it is more likely that this effect can be explained by the hypothesis of Longpre et al. (2020), that augmentation is only beneficial for LLMs when it creates new linguistic patterns – something not achieved by most traditional text augmentations methods.

Augmentation Type	Direction Accuracy		Profit Simulation	
	Mean	Std. dev	Mean	Std. dev
None	64.495%	0.165291381	602.6899\$	105.1580125
SynonymReplacement	64.577%	0.313279844	754.2226\$	88.3151921
Noise	63.225%	0.325887124	608.8095\$	133.8827053
BackTranslation	63.084%	0.300257334	590.7555\$	136.7412437
RandomInsertion	62.678%	0.659140044	569.2456\$	265.5292116
RandomSwap	61.645%	0.741508752	553.311\$	348.7875201
RandomDeletion	60.935%	0.453417066	606.2777\$	185.1755443

Table 4: Mean Metrics for each Augmentation Type

5.7 Quality over Quantity

In this experiment, the impact of favoring quality over quantity in the data selection is examined. For this, training on two different datasets was conducted. The quality

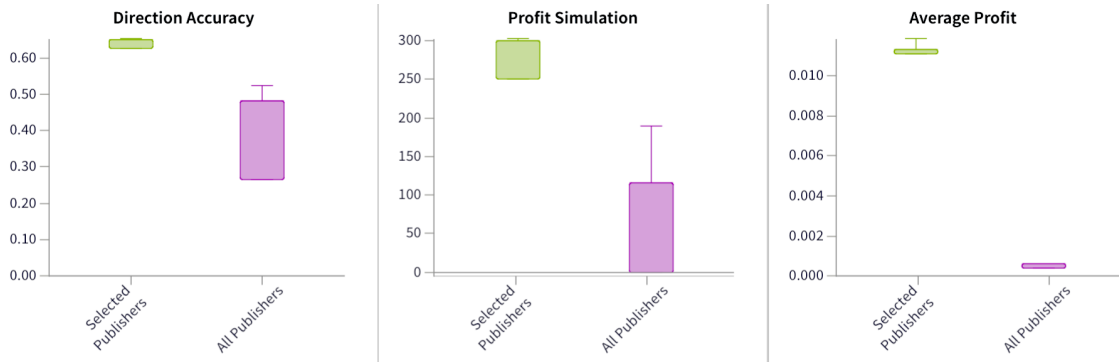


Figure 16: Performance Comparison between Selected Publishers and All Publishers

dataset follows all criteria described in Section [3.3.1](#). The quantity data applies preprocessing, such as the removal of outliers, but does not make any distinctions based on publishers. Although no filtering on this criterion would yield enough posts for a comparable dataset, augmentation was still employed, as it has proven to have positive effects on the performance.

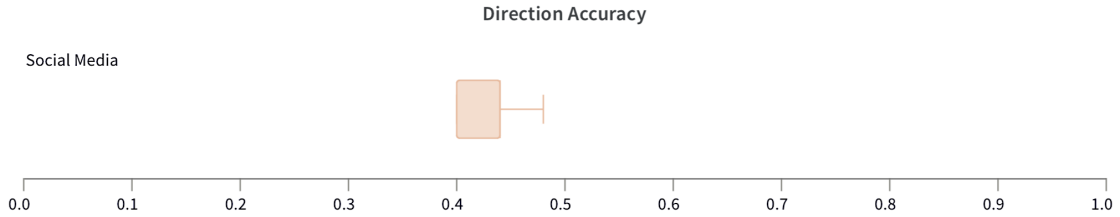
The results of the test runs can be seen in Figure [16](#). With an accuracy of $64.23\% \pm 1.4\%$ the quality delivers notable higher and more stable results compared to the quantity dataset ($50.2\% \pm 2.8\%$). These findings are further reflected in the average profit, where the quality dataset achieves an average profit of $1.14\% \pm 0.04\%$, outperforming the quantity dataset's $0.05\% \pm 0.02\%$. While the *Profit Simulation* metric is not ideal for direct comparison across different test sets, it is still noteworthy that the model trained on the quality dataset achieved a higher profit despite having fewer observations in the test set.

5.8 Excess Return

Target Variable	Direction Accuracy	Sentiment Accuracy
1h RoR	$65.4\% \pm 2.5\%$	$87.1\% \pm 2.8\%$
1h Excess Return	$68.1\% \pm 5.12\%$	$91.9\% \pm 1.58\%$

Table 5: Comparison between RoR and Excess Return as Target Variable.

Comparison between training with the *RoR* or the *Excess Return* as target variable has shown that the *Excess Return* yields slightly better results on the *Direction Accuracy*. However, its higher standard deviation also indicates a less stable performance. A possible reason for this is that through the adjustment by the *CAPM* rather neutral observations are shifted from slightly positive to slightly negative, and vice versa. In the *Sentiment Accuracy*, which distinguishes also neutral cases besides bullish and bearish ones, a more stable performance is achieved.

Figure 17: Attempt on Training on small Dataset of \mathbb{X} Posts

5.9 Social Media

During the first experiments with posts retrieved from *Reddit* high similarities to the news articles could be observed. Often, posts refer to news articles and therefore contain some time discrepancy compared to the original article. To focus more on the primary sources, further evaluations were omitted and devoted to the news dataset.

The training data based on the most popular list of 30 selected financial influencers yielded an accuracy of $44\% \pm 4\%$ on its test set (see Fig 17). The reason for the poor performance is most likely due to the low amount of observations. After completing the preprocessing steps – including filtering out tweets unrelated to the selected stocks, removing outliers, splitting off a test set, and applying augmentation-assisted sampling – a total of 420 instances remained for use in the training process, including the validation data.

5.10 Performance

Model	Dir. Acc.	Sentiment Acc.	Profit	Avg. Profit
Ours	$65.4\% \pm 2.5$	$87.1\% \pm 2.8$	$294.67\$ \pm 25.72$	$1.19\% \pm 0.15$
FinBERT	$46.77\% \pm 9$	$45.16\% \pm 13.5$	$-34.28\$ \pm 41.77$	$-0.42\% \pm 0.52$

Table 6: Final Metrics

The final performance of the trained model on our test set is presented in Table 6. Its results are comparable to those achieved by other ML-assisted trading strategies (see Sec 1.1.5). The second row of the table shows the performance of the FinBERT model (Araci, 2019), evaluated on the same samples. It is important to emphasize that this is not intended to be a comparison between FinBERT and our trained model, as the models are designed for different tasks. Nevertheless, it is interesting to see how the predictions of a promising text sentiment classification model can be applied to trading strategies, demonstrating that the sentiment of a text is not always aligned with the stock movement.

6 Discussion

6.1 Interpretation of Results

Evaluating the performance of fine-tuned LLMs in making assumptions on the subsequent price change based on a news article has demonstrated a notable difference between the sentiment perceived by readers and the actual stock market development. While models like FinBERT (Araci, 2019) achieve impressive results in the classification of the sentiment on financial news, it is still quite challenging to transfer this knowledge to stock market trading. This can be attributed to the complex behavior of the market, which is also influenced by additional factors having impacts on the price. There are news articles, that create a positive impression to the reader but may still be irrelevant for the stock market pricing. Some events might be relevant for the market, yet fail to affect the price since their impact was already priced in due to prior events anticipating them.

Although the accuracies achieved in this study may appear less impressive when compared to those in other fields, they nonetheless remain valuable for algorithmic trading strategies. Any method that can reliably demonstrate an expected profit greater than \$0 is commonly considered profitable in the field of finance. This begins with strategies that outperform random guessing, achieving success rates above 50%. Those with consistently positive expected values over extended backtesting periods are often leveraged, through options trading, to amplify potential returns. Often such strategies do not exceed accuracies of 70% (see Section 1.1.5), which is consistent with the *Efficient Market Hypothesis* (see Section 2.1.7).

6.2 Future Work

6.2.1 Backtesting Framework

Even when a model appears to be profitable based on the simulation metrics used in this study, it should be evaluated using a more complex testing framework before being applied in real-world scenarios. In addition to testing the strategy over a longer period, the framework should consider additional factors that can erode the profitability. One such factor is slippage, which refers to the difference between the anticipated price and the actual price at which the trade is executed. To mitigate this deviation, the testing framework must account for the correct price (ask or bid), depending on the side of the trade (buy or close) and its market depth. Additionally, the framework should factor in that the expected price may not always be realized, especially in volatile markets. Finally, other costs incurring when trading, including commissions and transaction costs, must also be considered.

6.2.2 Further Social Media Evaluations

The analysis of the effects of social media posts on stock market development was kept brief due to resource constraints. Nonetheless, there are a number of valid points that offer grounds for further investigation. A further evaluation of the data from *Reddit* was omitted due to a majority of the selected posts referring to news articles. Therefore, the decision was made to focus on the primary sources of the articles instead. However, comments to such posts on *Reddit* could still be a promising source for analyzing the sentiment of the community, which could be explored in future work. The evaluation of Section 5.9 revealed that the gathered posts are not sufficient for effective training. An examination of the posts, however, revealed that they contain a distinct type of information not appearing in news portals. Platforms like *X* are popular among users for sharing trading signals, which could serve as a useful resource for market trend analysis. To train a model effectively on this data, a larger number of instances is required, which is not achievable with the approach employed in this work. Due to the considerable costs associated with the API for historical search functionality, it was not viable for this study to further explore its potential.

6.2.3 Informational Pre-Training

In their study, Araci (2019) concluded that domain pre-training does not provide a significant advantage. A topic for future exploration could be investigating alternative ways of pre-training. One concept could involve introducing company-specific semantic information during pre-training, rather than focusing solely on enhancing text comprehension within the targeted domain. This approach might provide the model with a deeper understanding of company-related contexts, enabling it to draw conclusions when news titles, for example, mention the sector of a company. Such datasets could be dynamically generated using data from sources like *yfinance* and constructed either through templates or by utilizing LLMs for phrasing.

6.2.4 Stock-Aware Predictions

While this study focussed solely on the evaluation of news involving a single asset, the dataset also has the potential for training models on news covering multiple companies, producing contradictory sentiments. Future work could explore training techniques to enable company-aware predictions on such data. Two approaches for incorporating conditional information of the targeted stock alongside the textual data into the model are illustrated in Figure 18. One approach involves including the targeted stock ticker in the text before it is processed by the model (Sinha et al., 2022; Son et al., 2023). This can be achieved by replacing the company name in the text with a target token, indicating which company is the subject of the prediction. As a side effect, this substitution eliminates the information regarding which company the news is about, which can help mitigate biases of the

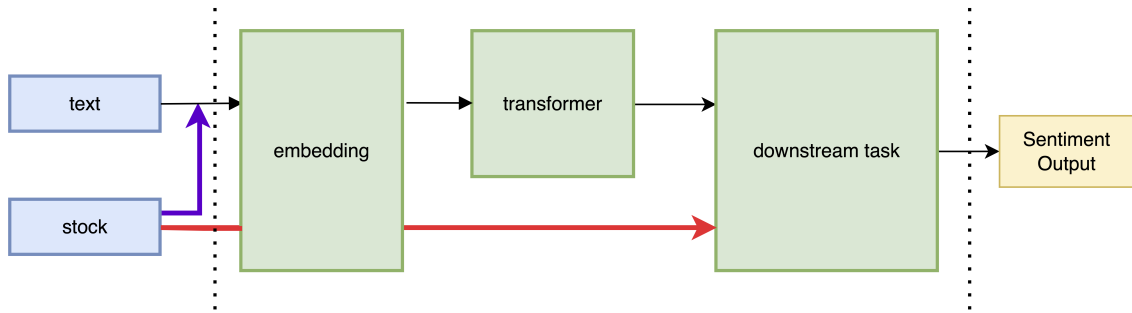


Figure 18: Approaches for Incorporating Stock Information as Context to the Model. (Blue): Add the stock ticker as token to the embeddings. (Red): Provide the stock ticker as additional context to the downstream task.

model towards certain companies. However, for this approach to be applicable, the text must explicitly mention the company name, making it unsuitable for instances that reference a company indirectly. An example of this is the headline "**Sharp slows larger *iPad* screen production as demand shifts**", which refers to Apple through its product but not by its name. Another option is to pass the stock identifier at a later stage as additional input to the downstream task. For this approach, the targeted company does not need to be explicitly mentioned in the text. At the same time, the transformer itself will not be able to capture a deeper interaction between the stock and the textual data before passing it to the downstream task.

7 Conclusions

Evaluations have shown that models achieving state-of-the-art results on text sentiment classification tasks, such as *FinBERT* (Araci, 2019), can not be applied directly on the market. This is due to the sentiment of the text is often not aligned with the actual subsequent stock price development, which can be explained by other influencing factors of the market. Therefore, training the model with target data based on price data is required for a model to make assumptions about market development. For this, a new dataset was created, which involves the *Rate of Return* and *Excess Return* as target variables. In this way, it is able to connect news events with price developments, while trying its best to isolate external factors. For every observation a list of involved stocks is included, paving the way for future work on stock-aware prediction models.

Using augmentation-assisted sampling it was possible to address the imbalanced data problem inherent to this type of data and achieve results that are compatible with other ML-assisted trading strategies. For this different textual augmentation methods were evaluated, demonstrating that the added complexity of LLM-assisted augmentation methods is required to mitigate overfitting for LLM.

A Appendix

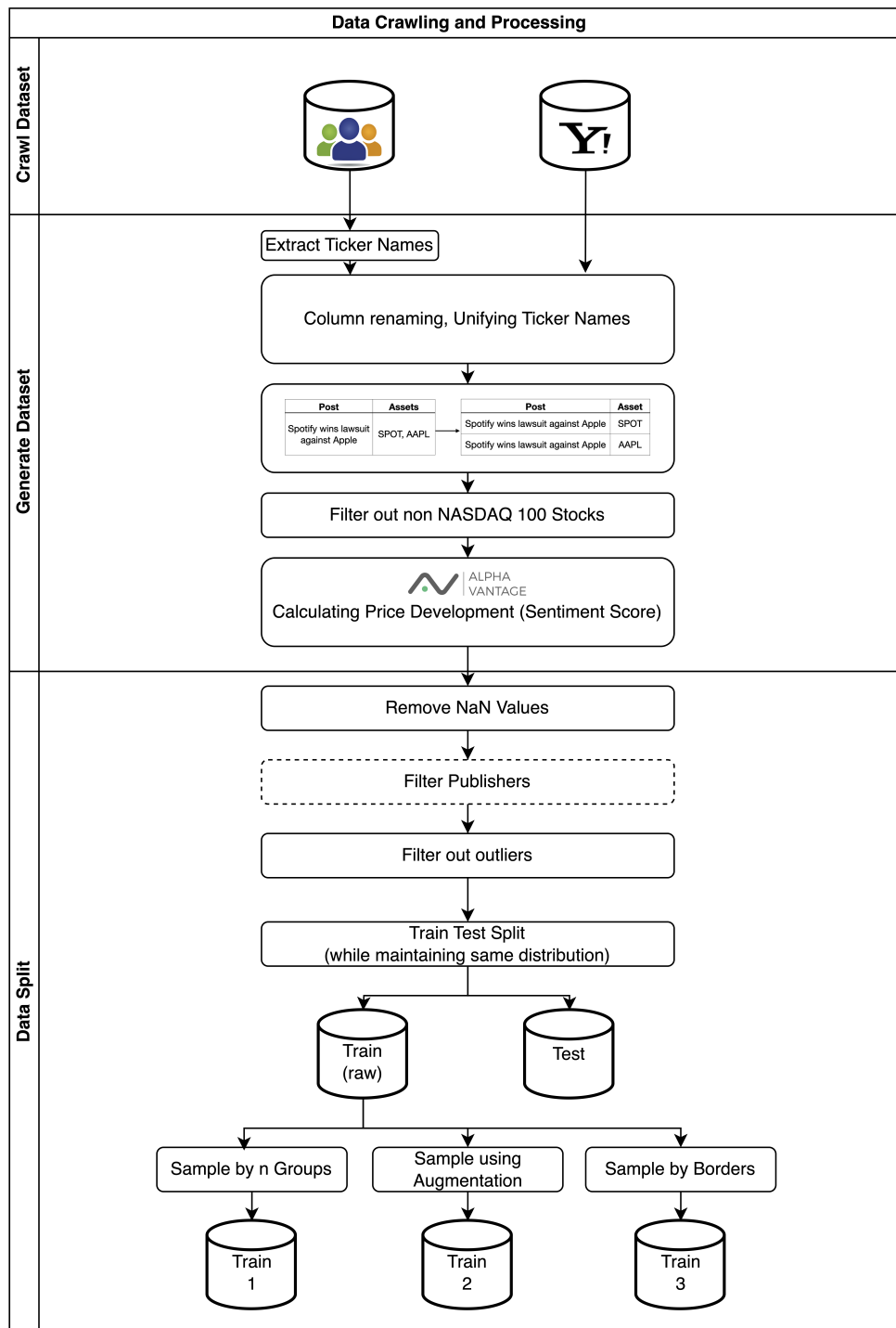


Figure 19: Visualization of the whole data processing applied to the scrapped data originating from Social Media and Yahoo News. `generate_dataset.py` splits and select the stocks of the posts and calculate their target values. `data_split.py` splits the dataset into train and test after some filtering.

Bibliography

- Milam Aiken and Mina Park. The efficacy of round-trip translation for mt evaluation. *Translation Journal*, 14(1):1–10, 2010.
- Inês Almeida and André Sabino. Classification of financial markets influencers on twitter. In *CLASSIFICATION OF FINANCIAL MARKETS INFLUENCERS ON TWITTER*, 2022.
- Dogu Araci. Finbert: Financial sentiment analysis with pre-trained language models, 2019. URL <https://arxiv.org/abs/1908.10063>.
- Markus Bayer, Marc-André Kaufhold, and Christian Reuter. A survey on data augmentation for text classification. *ACM Computing Surveys*, 55(7):1–39, December 2022. ISSN 1557-7341. doi: 10.1145/3544558. URL <http://dx.doi.org/10.1145/3544558>.
- Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.
- Lukas Budach, Moritz Feuerpfeil, Nina Ihde, Andrea Nathansen, Nele Noack, Hendrik Patzlaff, Felix Naumann, and Hazar Harmouch. The effects of data quality on machine learning performance, 2022. URL <https://arxiv.org/abs/2207.14529>.
- John A Bullinaria and Joseph P Levy. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior research methods*, 39:510–526, 2007.
- Sin-wai Chan. *A dictionary of translation technology*. Chinese University Press, 2004.
- Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.
- Foo Yun Chee. Apple hit with \$2 billion eu antitrust fine in spotify case. *Reuters*, 2024. URL <https://www.reuters.com/technology/apple-hit-with-over-18-bln-euro-eu-antitrust-fine-spotify-case-2024-03-04/>. Accessed: 2024-10-09.
- Qinkai Chen. Stock movement prediction with financial news using contextualized embedding from bert, 2021. URL <https://arxiv.org/abs/2107.08721>.
- Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

- Bálint Csanády, Lajos Muzsai, Péter Vedres, Zoltán Nádasdy, and András Lukács. Llmber: Large-scale low-cost data annotation in nlp, 2024. URL <https://arxiv.org/abs/2403.15938>.
- Aswath Damodaran. What is the riskfree rate? a search for the basic building block. *A Search for the Basic Building Block (December 14, 2008)*, 2008.
- Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL <https://arxiv.org/abs/1810.04805>.
- Ian Dewancker, Michael McCourt, and Scott Clark. Bayesian optimization primer. URL https://app.sigopt.com/static/pdf/SigOpt_Bayesian_Optimization_Primer.pdf, 2015.
- Bosheng Ding, Chengwei Qin, Ruochen Zhao, Tianze Luo, Xinze Li, Guizhen Chen, Junjie Xia, Wenhan andD Hu, Anh Tuan Luu, and Shafiq Joty. Data augmentation using LLMs: Data perspectives, learning paradigms and challenges. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics ACL 2024*, pages 1679–1705, Bangkok, Thailand and virtual meeting, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.97. URL <https://aclanthology.org/2024.findings-acl.97>.
- Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale, 2018. URL <https://arxiv.org/abs/1808.09381>.
- Richard Fairchild. *Emotions in the Financial Markets*, chapter 19, pages 347–364. John Wiley & Sons, Ltd, 2014. ISBN 9781118813454. doi: <https://doi.org/10.1002/9781118813454.ch19>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118813454.ch19>.
- Eugene F Fama. Efficient capital markets. *Journal of finance*, 25(2):383–417, 1970.
- Yaoyao Fiona Zhao, Jiarui Xie, and Lijun Sun. On the data quality and imbalance in machine learning-based design and manufacturing—a systematic review. *Engineering*, 2024. ISSN 2095-8099. doi: <https://doi.org/10.1016/j.eng.2024.04.024>. URL <https://www.sciencedirect.com/science/article/pii/S2095809924003734>.
- JR Firth. A synopsis of linguistic theory 1930-1955. *Studies in Linguistic Analysis, Special Volume/Blackwell*, 1957.

- Felix Hamborg, Karsten Donnay, Paola Merlo, et al. Newsmtsc: a dataset for (multi-) target-dependent sentiment classification in political news articles. Association for Computational Linguistics (ACL), 2021.
- Haibo He, Yang Bai, Eduardo A Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, pages 1322–1328. Ieee, 2008.
- Franz A. Heinsen. An algorithm for routing vectors in sequences, 2022. URL <https://arxiv.org/abs/2211.11754>.
- S Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997.
- Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, page 168–177, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138881. doi: 10.1145/1014052.1014073. URL <https://doi.org/10.1145/1014052.1014073>.
- Mohd Sabri Ismail, Mohd Salmi Md Noorani, Munira Ismail, Fatimah Abdul Razak, and Mohd Almie Alias. Predicting next day direction of stock price movement using machine learning methods with persistent homology: Evidence from kuala lumpur stock exchange. *Applied Soft Computing*, 93:106422, 2020. ISSN 1568-4946. doi: <https://doi.org/10.1016/j.asoc.2020.106422>. URL <https://www.sciencedirect.com/science/article/pii/S1568494620303628>.
- Neel Jain, Ping yeh Chiang, Yuxin Wen, John Kirchenbauer, Hong-Min Chu, Gowthami Somepalli, Brian R. Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Aniruddha Saha, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Neptune: Noisy embeddings improve instruction finetuning, 2023. URL <https://arxiv.org/abs/2310.05914>.
- Michael C Jensen. The performance of mutual funds in the period 1945-1964. *The Journal of finance*, 23(2):389–416, 1968.
- Yi Ji, Yuxuan Luo, Aixia Lu, Duanyang Xia, Lixia Yang, and Alan Wee-Chung Liew. Galformer: a transformer with generative decoding and a hybrid loss function for multi-step stock market index prediction. *Scientific Reports*, 14(1):23762, 2024.
- Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. Marian: Fast neural machine translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia, July 2018. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/P18-4020>.

- Daniel Jurafsky. Speech and language processing, 2000.
- Anders Krogh and John Hertz. A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4, 1991.
- John Lintner. The valuation of risk assets and the selection of risky investments in stock portfolios and capital budgets. In *Stochastic optimization models in finance*, pages 131–155. Elsevier, 1975.
- Bing Liu. Sentiment analysis and opinion mining. *Morgan & Claypool Publishers*, 168, 2012.
- Shayne Longpre, Yu Wang, and Chris DuBois. How effective is task-agnostic data augmentation for pretrained transformers? In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4401–4411, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.394. URL <https://aclanthology.org/2020.findings-emnlp.394>.
- Macedo Maia, Siegfried Handschuh, André Freitas, Brian Davis, Ross McDermott, Manel Zarrouk, and Alexandra Balahur. Www’18 open challenge: Financial opinion mining and question answering. In *Companion Proceedings of the The Web Conference 2018, WWW ’18*, page 1941–1942, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee. ISBN 9781450356404. doi: 10.1145/3184558.3192301. URL <https://doi.org/10.1145/3184558.3192301>.
- Burton G Malkiel. The efficient market hypothesis and its critics. *Journal of economic perspectives*, 17(1):59–82, 2003.
- P. Malo, A. Sinha, P. Korhonen, J. Wallenius, and P. Takala. Good debt or bad debt: Detecting semantic orientations in economic texts. *Journal of the Association for Information Science and Technology*, 65, 2014.
- Tomas Mikolov. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 3781, 2013.
- George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J. Miller. Introduction to wordnet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–244, 12 1990. ISSN 0950-3846. doi: 10.1093/ijl/3.4.235. URL <https://doi.org/10.1093/ijl/3.4.235>.
- Anders Giovanni Møller, Arianna Pera, Jacob Dalsgaard, and Luca Aiello. The parrot dilemma: Human-labeled vs. LLM-augmented data in classification tasks. In Yvette Graham and Matthew Purver, editors, *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 179–192, St. Julian’s, Malta, March 2024. Association for Computational Linguistics. URL <https://aclanthology.org/2024.eacl-short.17>.

- Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.
- Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.
- Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. *arXiv preprint cs/0205070*, 2002.
- Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL <http://www.aclweb.org/anthology/D14-1162>.
- Alec Radford. Improving language understanding by generative pre-training. 2018.
- Pranati Rakshit and Avik Sarkar. A supervised deep learning-based sentiment analysis by the implementation of word2vec and glove embedding techniques. *Multi-media Tools and Applications*, pages 1–34, 2024.
- Ramit Sawhney, Shivam Agarwal, Arnav Wadhwa, and Rajiv Ratn Shah. Deep attentive learning for stock movement prediction from social media text and company correlations. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8415–8426, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.676. URL <https://www.aclweb.org/anthology/2020.emnlp-main.676>.
- Peter Sellin. Monetary policy and the stock market: Theory and empirical evidence. *Journal of Economic Surveys*, 15(4):491–541, 2001. doi: <https://doi.org/10.1111/1467-6419.00147>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-6419.00147>.
- William F Sharpe. Capital asset prices: A theory of market equilibrium under conditions of risk. *The journal of finance*, 19(3):425–442, 1964.
- Ankur Sinha, Satishwar Kedas, Rishu Kumar, and Pekka Malo. Sentfin 1.0: Entity-aware sentiment analysis for financial news. *Journal of the Association for Information Science and Technology*, 73(9):1314–1335, 2022. doi: <https://doi.org/10.1002/asi.24634>. URL <https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/asi.24634>.
- Guijin Son, Hanwool Lee, Nahyeon Kang, and Moonjeong Hahm. Removing non-stationary knowledge from pre-trained language models for entity-level sentiment classification in finance. *arXiv preprint arXiv:2301.03136*, 2023.
- Duyu Tang, Bing Qin, and Ting Liu. Deep learning for sentiment analysis: successful approaches and future challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 5(6):292–303, 2015a.

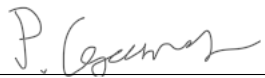
- Duyu Tang, Furu Wei, Bing Qin, Nan Yang, Ting Liu, and Ming Zhou. Sentiment embeddings with applications to sentiment analysis. *IEEE transactions on knowledge and data Engineering*, 28(2):496–509, 2015b.
- Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.
- Richard J Teweles and Edward S Bradley. *The stock market*, volume 64. John Wiley & Sons, 1998.
- Jörg Tiedemann and Santhosh Thottingal. OPUS-MT — Building open translation services for the World. In *Proceedings of the 22nd Annual Conferenec of the European Association for Machine Translation (EAMT)*, Lisbon, Portugal, 2020.
- Jörg Tiedemann, Mikko Aulamo, Daria Bakshandaeva, Michele Boggia, Stig-Arne Grönroos, Tommi Nieminen, Alessandro Raganato Yves Scherrer, Raul Vazquez, and Sami Virpioja. Democratizing neural machine translation with OPUS-MT. *Language Resources and Evaluation*, pages 713–755, 2023. ISSN 1574-0218. doi: 10.1007/s10579-023-09704-w.
- Alexandra Gabriela Țițan. The efficient market hypothesis: Review of specialized literature and empirical research. *Procedia Economics and Finance*, 32:442–449, 2015. ISSN 2212-5671. doi: [https://doi.org/10.1016/S2212-5671\(15\)01416-1](https://doi.org/10.1016/S2212-5671(15)01416-1). URL <https://www.sciencedirect.com/science/article/pii/S2212567115014161>. Emerging Markets Queries in Finance and Business 2014, EMQFB 2014, 24-25 October 2014, Bucharest, Romania.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. URL <https://arxiv.org/abs/2307.09288>.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

- Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019. URL <https://arxiv.org/abs/1804.07461>.
- Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. Attention-based lstm for aspect-level sentiment classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 606–615, 2016.
- Philippe Weil. The equity premium puzzle and the risk-free rate puzzle. *Journal of monetary economics*, 24(3):401–421, 1989.
- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/2020.emnlp-demos.6>.
- Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V. Le. Unsupervised data augmentation for consistency training, 2020. URL <https://arxiv.org/abs/1904.12848>.
- Yumo Xu and Shay B Cohen. Stock movement prediction from tweets and historical prices. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1970–1979, 2018.
- Shiwei Zhang, Xiuzhen Zhang, Jeffrey Chan, and Paolo Rosso. Irony detection via sentiment-based transfer learning. *Information Processing & Management*, 56(5): 1633–1644, 2019.
- Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

Declaration of Authorship

Ich erkläre hiermit gemäß §9 Abs. 12 APO, dass ich die vorstehende Abschlussarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Des Weiteren erkläre ich, dass die digitale Fassung der gedruckten Ausfertigung der Abschlussarbeit ausnahmslos in Inhalt und Wortlaut entspricht und zur Kenntnis genommen wurde, dass diese digitale Fassung einer durch Software unterstützten, anonymisierten Prüfung auf Plagiate unterzogen werden kann.

Bamberg, 01.12.2024
Place, Date


Signature