Imperial College London

Imperial College London

Department of Mathematics

Bitcoin Trading Strategies based on Twitter Sentiment Analysis

Author: Konstantinos Tsoulias (CID: 02007404)

A thesis submitted for the degree of

MSc in Mathematics and Finance, 2020-2021

Declaration

The work contained in this thesis is my own work, unless otherwise stated.

Acknowledgements

I would like to thank Avery Soh, Jordan Anaya, and Stuart Jamieson for their constant support, guidance, ideas, and trust throughout this project, which gave me confidence and pushed me to do better and go further.

Special thanks to Jose Forero, Keshav Nehra, Valentina Frezza, and Caspar Marney, as they provided me with all the help and equipment needed to complete this project. I would like to extend my deepest gratitude to the entire Velador family, for the heartfelt welcome and for making sure that my internship would be a great experience.

I want to also express my appreciation to my tutor Dr. Antoine Jacquier for his crucial input and assistance that allowed me to structure and present the contexts of this project in a comprehensible manner. I would like to thank all the teaching staff of MSc Mathematics and Finance for the quality lectures delivered that provided me with great knowledge.

Lastly, I would like to sincerely thank my family and friends for their immense support all these years, as nothing would have been possible without it.

Abstract

Asset price movements are the results of buy and sell orders executed over time. Knowing how every market participant will behave in the future is equivalent to foreseeing the trajectory of future market prices. Although this information is impossible to obtain, one can search for an indication of participants' intentions. Since thought precedes action, insights regarding participants' emotions and perceptions can indeed provide such indication. For many years now, social media have become an integral part of our daily lives, with almost 90% of Internet users expressing their thoughts and emotions on these platforms every day. These public channels that host our online interactions have grown to become huge public databases of our own opinions and ideas, that are conveniently timestamped and categorized. The recent advancements in Natural Language Processing have led to the development of state of art models able to understand the contexts and meanings of written language with great precision. In this project, I employed some of the most advanced language models for classifying tweets about Bitcoin as either positive or negative depending on their context. I then used the calculated sentiment values to construct trading signals which I then incorporated into a trading strategy. Specifically, I created 4 signals in total, distinguished by the timeframe on which they are traded (daily or hourly) and the model they were created by (BERT model or GloVe model). After trading those signals through simple buy-only strategies for the period between September 2017 and January 2019, all strategies ended up profitable. The two strategies trading on BERT generated signals were able to produce returns of 100% during that period. Interestingly enough, the two strategies based on signals calculated by GloVe, performed even better, reaching total returns of 200% during testing.

Contents

List of Figures

List of Tables

Introduction

Predicting price movements is proven to be no easy task. The efficient-market hypothesis suggests, that stock prices reflect all currently available information, and any price changes that are based on newly revealed information thus are inherently unpredictable. However, it has been seen that there are inefficiencies like inside information and market manipulation that one can exploit for profit, which makes the financial market efficiently-inefficient. In a more simplistic view, and since a price chart is the product of buy and sell orders being executed over time, one can state that price movements are a function of the behavior of market participants. Furthermore, since thought precedes action, the price can be assumed to always reflect the participants' psychology and understanding of the information rather than the information itself. It is, therefore, reasonable to query whether knowledge on the psychology of market participants and one time could give insights about price movements in the future.

Acquiring such information would have been almost impossible in past years. Nevertheless, the rise of social media has provided us a place, where people can interact, state their opinions, and express their feelings publicly. Moreover, their increased popularity has made it so that social media are now an integrated part of everybody's life. Today, over 90% of the Internet is active in social media [1], which translates to over 50% of Earth's entire population. As a result, huge datasets of public timestamped opinions have been formed in these networks, enabling research and statistical analyses to be performed. Other advantages come from the fact that many of the most popular platforms like Twitter or Reddit categorize their content with hashtags and subreddits, making it easier to pinpoint textual data of a specific topic or content. Combined with the simplifications and automation in accessing data supplied by the webpages' APIs, these factors make obtaining and handling social media data in real-time even easier.

Possessing such data however is not of any importance unless there is a way to process them systematically and extract their informational value. The recent advancements in Natural Language Processing and Machine Learning have allowed computers to understand human language, enabling textual data to be processed, evaluated, and classified by their context and meaning. A common categorization is to identify whether a message has a positive or negative meaning, known as sentiment classification. This is usually done with the use of neural networks that have been trained on huge amounts of data to receive text and encode it in real-valued vectors that capture its context. As for the classification, this is usually a supervised approach that requires the training of a model on pre-labeled data.

As it is understood, the quality of the evaluation depends directly on the quality of the data. Even though social media data come in big quantities, they are usually low in quality. Most of the messages contain slang language, sarcasm, and misspellings all of which make the job of the classifier significantly harder. In addition, the amount of spam and advertisement in these datasets, which provides no insight on user's psychology, is overwhelming and adds noise, thus worsening the classification results. Therefore, cleaning the data is an essential prerequisite for any analysis to be performed.

With all the necessary tools available and data prepared, one can begin to investigate the relationship between social media sentiment and market price movements. The main question to be answered is whether the current emotional state of social media users can predict their future behavior and in part that of market participants. Or whether it just reflects the reaction of the participants to recent news and price changes.

While proving one or the other is generally difficult, in the past social media activity has been seen to heavily influence assets' price movements. The most known example of this is possibly Elon Musk's tweets, which have been followed by significant movements in cryptocurrency prices in the past. Another such case is that of the /wallstreetbets subreddit which has been held responsible for the enormous rise in GameStop stock prices. Several works have also been examining the relationship between social media emotional content with financial markets over the past decade. In [2] the author developed a model with four different types of investors; naive investors following social media, fanatics that spread fake news, and rational short-term and long-term investors. The model showed the eventual domination of fanatics and rational belief structures while many historic market events were able to be explained. Other studies focus on utilizing social media sentiment for producing returns. In [3] researchers were able to associate the sentiment in tweets from users of few followers about a specific stock with its next-day returns. Authors in [4] derived the emotional states of Twitter users and used the state calm to predict the price movement of the Dow Jones Industrial Average (DJIA).

These findings are promising and encourage us to further search for these connections between social media content and future price movements and exploit them for profit. For that, knowing the sentiment of Twitter is not enough, for it needs to be incorporated into a suitable trading strategy. The strategy is responsible for making decisions after observing a signal and it is directly related to the hypothesis tested. For example, we can assume that positive sentiment can be translated into an increased willingness of participants to buy assets in the future. In that case, we expect a positive correlation between sentiment and future asset returns. On the contrary, a case can be made for extreme optimism or pessimism having the opposite effect on market prices. As John Templeton said "The time of maximum pessimism is the best time to buy, and the time of maximum optimism is the best time to sell" as sellers and buyers are getting exhausted in each case.

In this project, I attempt to answer all these questions by developing trading strategies based on social media sentiment. The hypothesis to be tested is whether social media data can be used as an indication for future price movements and specifically, if positive sentiment is an indication of a future increase in prices. Acknowledging the aforementioned difficulties and limitations of such an approach, Bitcoin will be the asset to be examined, as it is one of the most popular topics on the Internet, and is more accessible to social media users than stocks are. Moreover, the state of the art in NLP models has been employed for analyzing and classifying textual data, to reduce operational risk as much as possible. The data, are collected from Twitter for the period between September 2017 and January 2019. The data are preprocessed, cleaned of spam and advertisement before their classification. As for the trading, the strategies implemented will be as simple strategy mainly dependent on the calculated signal to guarantee that any benefit will come from the signal itself rather than the complexity of the strategy.

In this report, I will present the methods employed and the incentives behind them as well as the results of this work and their evaluation. The structure stands as follows: In Chapter 1 I introduce some of the key concepts upon which this project is based. Chapter 2 summarizes a literature review, where research papers of similar contexts are presented, together with explanations of how this work utilizes their findings and ideas and in what ways it tries to extend them. Chapter 3 is where an overview of the methodology followed is given as well as an analysis of the project's main components. In Chapter 4, an analysis of the data is presented. Data are a major part of this work, and getting a better idea of their structure and features is necessary, to best utilize them later. Chapter 5 is dedicated to Spam Filtering and cleaning of the data sets as noise can be devastating for obtaining meaningful results, and more in our case due to the peculiarities of social media data. Chapter 6 contains the core methods, models, and results related to the Natural Language Processing section of this project. Chapter 7 focuses on the calculation of the trading signal and the construction of the trading strategy. The evaluation of the latter is also performed through various methods and metrics after being backtested on historical data. In the final chapter, Chapter 8, I summarize the findings of this project, evaluate them, and discuss how the methods employed can be enriched in the future.

Chapter 1

Background

1.1 Word Vectors

Transforming categorical data to vectors of numbers is a well-established technique in the field of Natural Language Processing. One-hot-encoding [5] and bag-of-words [6], two widely used models, create vectors that represent the presence or absence of a word within an observation with 1 and 0 respectively. However useful in particular projects, the encodings created by the aforementioned techniques fail to capture the meaning and relationships between the words, thus making them unsuitable for any type of semantic analysis on textual data.

Word vectors address these shortcomings by representing words as vectors of real-valued numbers, where each point reflects a dimension of the word's meaning so that similar words have similar representations, capturing many linguistic regularities. The similarity or difference between two words can be measure by the cosine distance between them. That being the case, the cosine distances between vector('king') and vector('queen') should be similar to that between vector('man') and vector('woman'). In that way we can use arithmetic on these vectors to derive their meanings, as for example vector operations like vector('king') - vector('man') + vector('woman') is close to vector('queen') 1.1.

1.1.1 Word2Vec

To take advantage of these regularities tools have been developed, like word2vec [7][8] that make it possible to train neural network models on huge data sets to learn word associations. The functionality of word2vec can be summarized as a feed-forward neural network taking a text corpus as input and producing the word vectors as output. The two main learning algorithms used to train word2vec are continuous bag-of-words and continuous skip-gram [9]. The former takes the input as each word in the corpus, sends them to a hidden layer (embedding layer) and from there it predicts the context words. While the latter has the reverse functionality predicting the original word given the word's neighbors. In both cases, the embedding vector for a particular word can be obtained by taking the hidden layer value.

1.1.2 GloVe

Another similar model for obtaining word embeddings is Glove [10], which differs from word2vec from the way it was trained. Specifically, it constructs a word/word co-occurrence matrix, that describes how frequently words co-occur with one another in a given corpus, the intuition being that word-word co-occurrence probabilities have the potential for encoding some form of meaning. By factorizing this matrix it produces a word/features matrix of lower dimensions where each row holds a vector representation for a particular word. The best factorization is achieved when the lower-dimensional representations can explain most of the variance in the high-dimensional data i.e. the "reconstruction loss" is minimal.

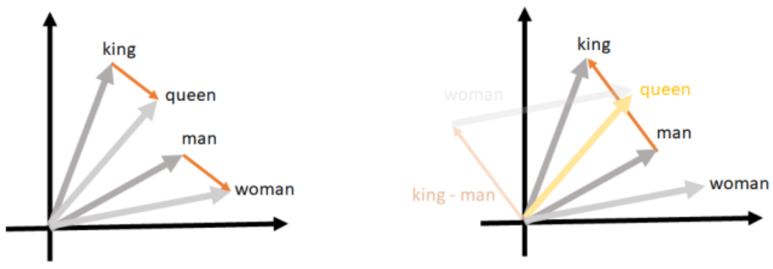


Figure 1.1: Word vector illustration

Both of the aforementioned models are used in many NLP applications that require information about the meaning of words such as sentiment analysis, document clustering, question answering, etc. For this project, I used the pre-trained word vectors provided by GloVe that was trained on Twitter data which makes them more suitable for this project's purpose.

1.1.3 ELMo

While both Glove and Word2Vec are tools that immensely accelerated the progress in the NLP field, they do have limitations. The most important one is that the embeddings fail to capture the context in which the word has been used. The word "bat" for example can mean either a weapon or an animal. To address this models have been developed that involve the contextual information [11][12] to their vector representations, one of which is ELMo [13].

ELMo uses bi-directional LSTMs [14] and creates word embedding after first looking at the entire sentence, in order to obtain contextual information. ELMo can do that by having been trained to predict the next word in word sequences, an effective and convenient method for understanding language since it can utilize huge amounts of textual data without requiring labels. Furthermore, by training bi-directional LSTMs the model is also aware of the words that precede the word that's being processed. The final contextualized word embedding is a weighted summation of the LSTMs' hidden states after having been concatenated in an appropriate way.

1.2 Deep Learning for NLP

1.2.1 Sequence to Sequence models

Sequence to sequence neural networks has become the center of attention during the recent innovations in the field of Natural Language Processing [15]. As the name implies a Seq2seq model takes as input a sequence of items and outputs another sequence of items. An application where that would be optimal is neural machine text translation. The hidden layers can be distinguished into two separate parts, the encoder, and the decoder.

In most NLP applications both the encoder and the decoder are recurrent neural networks, where the encoder processes the input, compiles it into a vector which is then passed to the decoder for producing the sequence output. If the input sequence is for example a sentence then the encoder processes it word by word. However for the processing to begin the input firsts need to be represented by a vector that captures its meaning, a procedure in which word embedding algorithms, mentioned before, find great utility. The RNN in the encoder starts off with an initial hidden state and the first-word vector as an input. It then updates the hidden states and gets the next item. The process is repeated until every word vector of the input has been encoded. After the encoding, the last hidden state is passed to the decoder. The latter uses this input along with the hidden states of its own calculates and outputs the resulting sequence word by word.

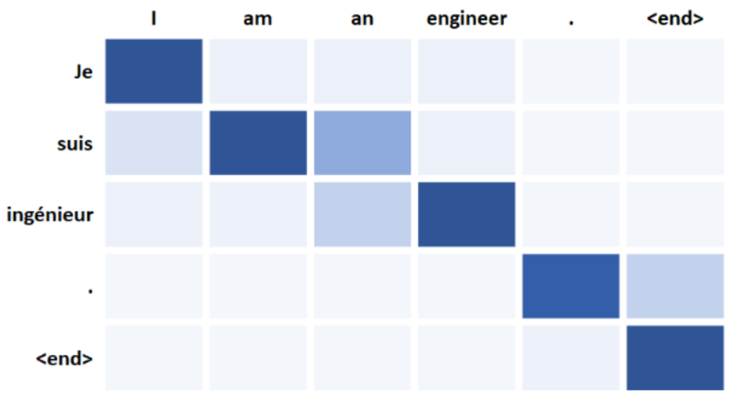


Figure 1.2: Example of attention matrix in machine translation.

1.2.2 Attention models

Of course, the seq2seq models have their limitations. One of them comes from the fact that only the last hidden state is passed from the encoder to the decoder, which makes it hard for the model to cope with long sentences. For that matter "Attention" was introduced [16, 17] aiming to improve the quality of machine translation by controlling the focus on the relevant words in the input text.

In terms of functionality, attention models introduced two main changes to seq2seq models. The first one is that the encoder now passes all hidden states to the decoder rather than the last one. The second one is the inclusion attention decoder which precedes the decoding process. Since each of the encoder's hidden states is associated with a certain word in the input sequence the attention decoder can learn to amplify or reduce each hidden state bringing attention to the relevant parts of the text for each part of the output. In machine translation, for example, the model learns how to strongly align words with their translations and weakly with the rest of them. Note that this allocation input to output words is not always one to one thus training is needed.

1.2.3 Transformers

The impact that the concept of Attention had in improving machine translation is undoubted. However, it also allowed the development of new models that use this idea to improve the accuracy and the speed of training. One of them is the Transformer proposed by Google [18] that adopts the same high-level architecture as seq2seq models, receiving and outputting a sequence while having an encoding and a decoding component with connections between them.

Both the encoding and the decoding components are stacks of encoders and decoders respectively of the same number. Each encoder can be divided into two main parts a self-attention layer and a Feed-Forward Neural Network connect serially. The process begins by transforming each word into vectors using an embedding algorithm, which is then passed into each encoder. The self-attention layer receives the input and while processing each word in it, it looks at the other positions to find some relevance leading to a much better understanding and thus a more efficient encoding overall [19]. As a result in the sentence "The cat runs away from the dog because it was afraid of it" the machine has can distinguish the meaning of the two "it" as they refer to different things, which further translates into a different encoding. Before getting to the next encoder the new vectors are passed to the FNN. In that phase they don't share any dependencies, a fact that allows from some parallelization in execution, speeding up further the training phase.

The decoder on the other hand differs from the encoder structure-wise as it interposes a third

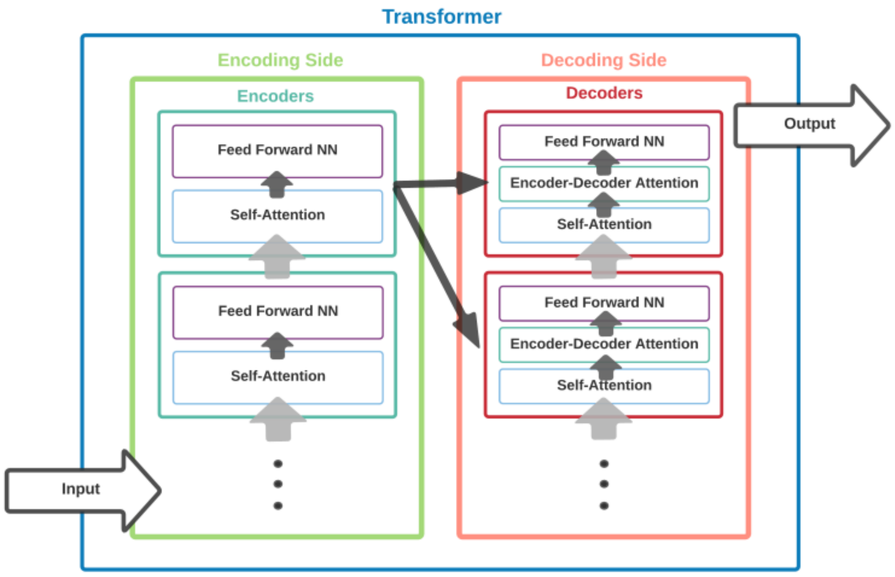


Figure 1.3: Transformer architecture illustrated

layer between self-attention and FNN, the "encoder-decoder attention" layer. This layer's objective is to utilize the set of attention vectors that the last encoder outputs to direct the decoder's attention on the relevant parts in the input sequence. As in the encoding side, each decoder passes its output to the next one until reaching the final Linear and Softmax Layer that is responsible for translating the number vectors back into words.

1.2.4 BERT model

Since the introduction of the Transformer, other variants of this architecture have been implemented, with the most known of them being the BERT model [20]. BERT, a model that builds on top of the Transformer and other clever ideas in Machnine Learning and NLP [21][22] provides a contextual representation of sentences and has grown to dominate the models of NLP in various tasks. BERT stands for Bidirectional Encoder Representations from Transformer, which perfectly describes it since it is essentially a stack of bidirectional layers of trained transformer encoders.

Before passing the input to the first encoder of the stack a special token needs to be inputted that lets the model know the type of task BERT is being used for. Just like on the encoding side of the Transformer the input is being passed from one encoder to the next while having self-attention and FNN layer applied to it each step of the way [23]. Things start to differ at the output of the Bert model. There according to the NLP task given by the special token supplied in the first position of the input. Therefore by accessing the first position of the output, we can retrieve a task-related vector representation of the input, that can be then passed into a trained neural network suitable for the task. In the paper researchers got great results in the classification task passing the output vector through a single layer classifier.

Interestingly enough it is possible to extract word embeddings from the hidden layers of the BERT model. The outputs per token of each encoding layer are essentially an embedding for that particular token. Authors have identified as best performing choices the sum the last 4 layers.

As for the tremendous benefits of BERT, they come from its bidirectional nature, its convenient transformer-like architecture, and finally from its training process. Similar to Elmo's contextualized embeddings BERT is trained on next-sentence predictions, where it receives pairs of sentences and identifies whether they are subsequent sentences in the same document. Again this is very convenient as we have an abundance of texts on which we can train the model. In order to use BERT however, it is proposed by its developer to fine-tune it by training on data relevant to the task at hand. In that phase, only specific hyper-parameters are getting adjusted allowing BERT to achieve state-of-the-art results on a variety of tasks.

Chapter 2

Related Work

Using Natural Language Processing and Deep-Learning Techniques to make stock market predictions is a well-established idea. In this chapter, I present several studies that have examined the forecasting power of unstructured data in the stock market, while also underlying their similarities and differences from this project.

In the past several efforts have been made to extract information from news websites and use them to forecast future stock price movements [24]. Here researchers tried to capture structured relationships from unstructured textual data outsourced by news websites and headlines. The idea is to identify big events stated in these articles and foresee their effect on future stock prices. The approaches used to achieve that differ slightly in each paper. Specifically, the methodology of [24] involves identifying organization names in news headlines and recognizing their actions. Then they relate these actions to post-event drifts in the stock price, and finally, use the identified relationships to construct an optimal portfolio.

Other papers have followed a deep learning approach, by developing and training language models. Specifically, in [25] researches, events are compiled as dense vectors and passed into a convolutional neural network, which then produces a trading signal. The informational content of the produced signal is then evaluated by incorporating it into a simple event-based trading strategy, that opens a long position if the model predicts a rise in stock prices the next day. To test the significance of their results randomized strategies and the S&P 500 were used as a benchmark.

This paper provides some significant guidelines for using Natural Language Processing and Deep Learning models to make forecasts. This includes the prepossessing of data, the use of word embeddings, the creation and training of the neural network as well as the evaluation of the produced signal. These general guidelines are also followed in this project and built upon to produce and test the final trading signal. As for the main differences they are found in the data used in each case and in the models employed. Here we examine only social media content which is more cumbersome in nature as it requires cleanings from spam messages, is rarely grammatically correct, and might contain slang, sarcasm, and misspellings all very difficult for a model to identify. Additionally for this project more sophisticated NLP models need to be employed that have been lately developed and seen to produce greater results than CNNs and LSTMs, and also specialize in handling social media data.

Nevertheless, a large literature regarding sentiment analysis on social media already exists [26] [27] [28]. These studies are focused on employing NLP techniques and supervised machine learning algorithms to correctly classify text as positive or negative. These efforts are concentrated on overcoming the barriers set by the very nature of the data i.e. slang, misspellings, limited amount of words, and leveraging other characteristics i.e. emoticons, hashtags to obtain an accurate classification. This objective is a prerequisite for this project as well, in order to construct a legitimate trading signal. However, this is extended here as one needs to aggregate the sentiment of individual postings on greater periods of time and use the calculated product to construct a profitable trading strategy.

The use of social media sentiment as stock market indicator is also a well studied subject. In [29] researchers studied the sentiment from a stock microblogging service over the period of three months and found it to have strong predictive value for future market prices. Other works such that of [30] that classified the sentiment of tweets with a Support Vector machine as positives or negatives, showed that changes in positive sentiment probability can be used as indicators of the changes in stock closing prices. Based on a similar hypothesis, researchers in [31] incorporated Twitter sentiment scores on a trading strategy, a methodology that closely relates to the contents of this report. Specifically, authors collected financial data from S&P 500 to trade on, and Twitter textual data for sentiment analysis. The tweets for classification were examined to involve information about a specific company or ticker, while the classification itself was done through word analysis a technique that involves mapping each word to a dictionary of terms in order to determine its sentiment. The sentiment is aggregated for each day by calculating negative to positive rates. Regarding the trading strategy, it involved buying stocks in the top 10% of sentiment score and selling those in the bottom 10%. Even though the NLP Techniques used were not extremely sophisticated researchers managed to accept the hypothesis and come up with a profitable strategy. For this project, some of these methods will be modernized by utilizing progressive deep learning models for text classification, and it will be investigated whether this additional complexity of the classification methods will result in better results.

Chapter 3

Overview

The scope of this project is to examine if social media sentiment regarding Bitcoin can be used as a signal and incorporated into a profitable trading strategy. Specifically, tweets from publicly available sources will be classified through pre-trained classifiers and be consolidated in the decision-making of a trading strategy. Lastly, I examine the results of testing the strategy during the period 09-2017-01-2019 and conclude the fitness of Twitter sentiment as an indicator for future Bitcoin price movements. The final implementation can be distinguished into four major parts that will be presented separately in this report:

Data Analysis: Here I inspect the pre-classified Data that will be used for training the text classifier and the Twitter parsed data that will be evaluated to obtain the final signal. The goal is to verify that the datasets are fit for the task that they will be used for.

Spam Filtering: After analyzing the datasets, the next and crucial objective is to clean them from spam and advertisements that add noise to the final sentiment evaluation. For that matter a few techniques are being applied and evaluated by drawing samples of the Tweets they label as "spam" or "not spam" and manually checking their correctness.

NLP classifiers and sentiment analysis: This is a major part of this project that concerns the classification of tweets as positive or negative. The first step here is to bring the inputs to a suitable form for the model to process. After that, we can deploy, train and evaluate several NLP classification models. Then we apply the same prepossessing to the unlabeled data before passing them through the trained classifier to end up with the final sentiment score that captures the positivity/negativity of Twitter users regarding Bitcoin. A final step is to evaluate the classification models and methods and decide which are more suitable to use for this particular application.

Trading strategy: This section involves all segments directly related to the operation of our trading strategy; trading signal construction and evaluation, trading strategy development, testing, and results in the evaluation. The most significant problem faced in this section is how to utilize the calculated sentiment in our trading strategy. However, this is not the only parameter that needs to be determined as the trading strategy development also involves deciding on the entering, exiting criteria, risk-management, and sizing of positions as well as the timeframe of trading. Finally, the strategy will be backtested and the informational value of the signal will be assessed.

Before looking at each part separately, the whole process can be visualized in Figure 3.1 below.

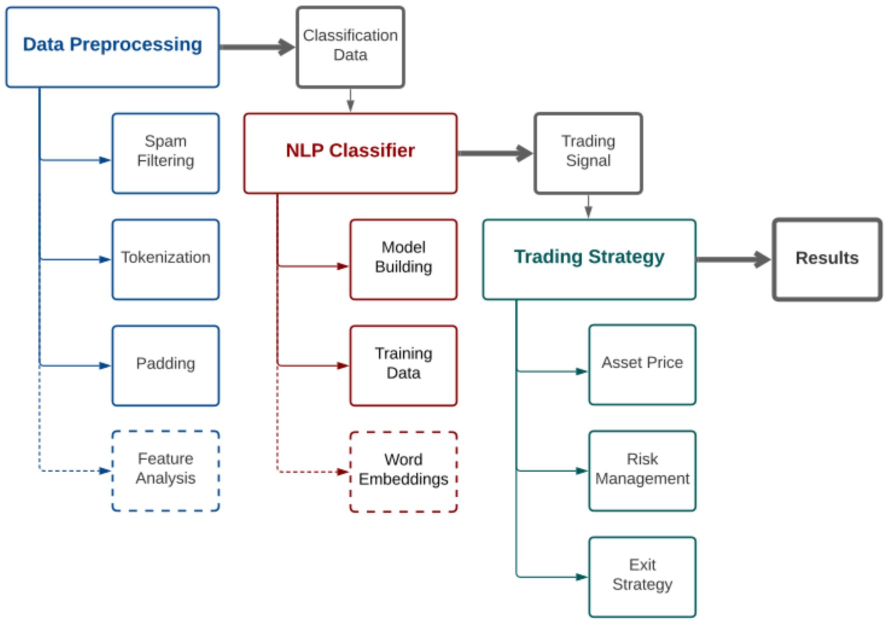


Figure 3.1: Project's main parts visualized.

Chapter 4

Data

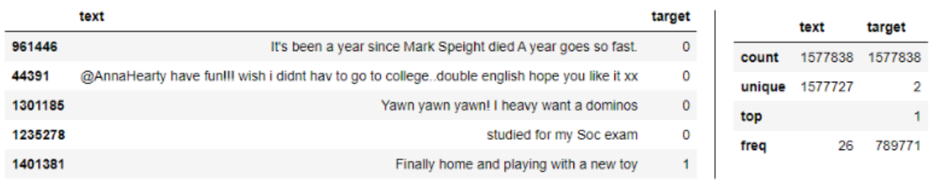
A core part of any analysis is the Data used. Thus it is crucial to visualize the input data sets, to gain a better understanding of their contents and evaluate their fitness for the task at hand. For this project, two different data sets were utilized,

1. a set of 1.5 million Tweets[1], labeled as 1 for positive or 0 for negative sentiment, that are used for the training of the classifier NLP model.
2. a set of 17.7 million Tweets related to Bitcoin[2], for the period 09-2017 - 01-2019, timestamped and unlabelled, for the trained classifier to evaluate and construct the trading signal.

Both of the aforementioned data sets are publicly available, and so are all the tools and models that were used for this project, assuring that any benefit will derive from the usefulness of the employed methods rather than our accessing of hard to come by information.

4.1 Training Data

Starting with the labeled Tweets, we get an idea of its contents by drawing a random sample and by printing the description of the set, as presented in 4.1.



	text	target
961446	It's been a year since Mark Speight died A year goes so fast.	0
44391	@AnnaHearty have fun!!! wish i didnt hav to go to college..double english hope you like it xx	0
1301185	Yawn yawn yawn! I heavy want a dominos	0
1235278	studied for my Soc exam	0
1401381	Finally home and playing with a new toy	1

	text	target
count	1577838	1577838
unique	1577727	2
top		1
freq	26	789771

Figure 4.1: Pre-classified tweets data set inspection.

The set has 2 columns, text and target, for the tweet and its labeled sentiment, respectively. A target of 0 corresponds to a negative sentiment whereas when labeled 1 the tweet is considered positive. From the descriptive statistics on the right, perhaps the most interesting one is the difference between the unique and total counts of text data, which hints the existence of duplicate tweets in our labeled dataset, perhaps consisting of spam messages or retweets. Fortunately, the number of them is quite low (111 out of 1,577,838 total Tweets), corresponding only to the 0.007% of the whole set, which is acceptable for our purpose. Then, examining the distribution of tweets in positives and negatives (Figure 4.2), it is clear that there is no class imbalance. This is encouraging, as it won't bias the model towards a particular direction, which can happen with heavily imbalanced training sets.

[1]Source: `http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/`
[2]Source: `https://www.kaggle.com/jaimebadiola/177-million-bitcoin-tweets`

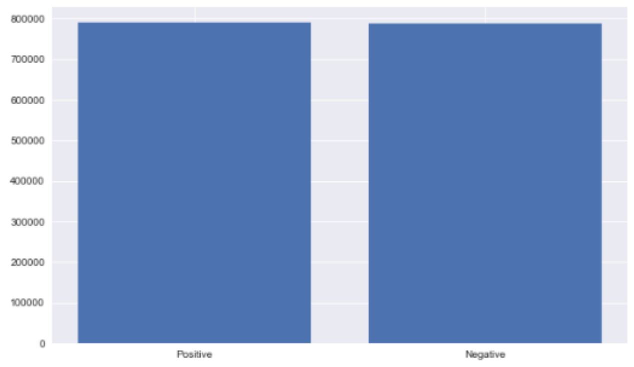


Figure 4.2: Label distribution for positive and negative pre-classified tweets.

However, there are still other characteristics that may differ between positive and negative tweets, thus making it easier for them to be distinguished. An example is the fact that negative texts tend to be more lengthy on average than positive ones, as people tend to over justify themselves when expressing a negative position. To check if that is the case here as well, I calculated the length of each tweet, classified them in positives and negatives, and plotted the histograms of the two groups as seen in Figure 4.3. The results, show that the the distribution is slightly more heavy-tailed for the negative group, but other than that they don't reveal any significant difference between the two classes.

Another important note here is the fact that some outliers seemingly exceed the limit of 250 set by Twitter. This is because mentions and hyperlinks are included here in the text column which may extend a tweets length beyond the defined limits.

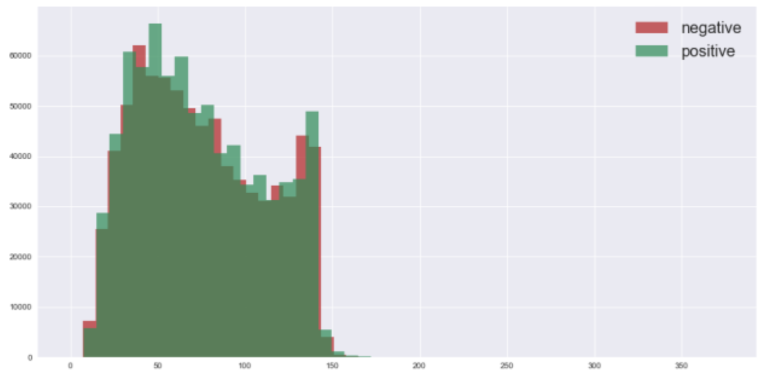


Figure 4.3: Tweet lengths distribution for positive and negative pre-classified tweets.

The same comparative analysis of distributions between the positive and negative tweets we can apply to several other features. Below I have included the distribution of word count and unique word count, both very important features in any NLP analysis. In Figure 4.4 we see the distribution of both sets having a smooth curve with some spikes in certain word numbers, while negative tweets appear to contain more words on average. This is also evident in Figure 4.5 for unique words counts, although here instead of spikes we have sudden drops for certain values.

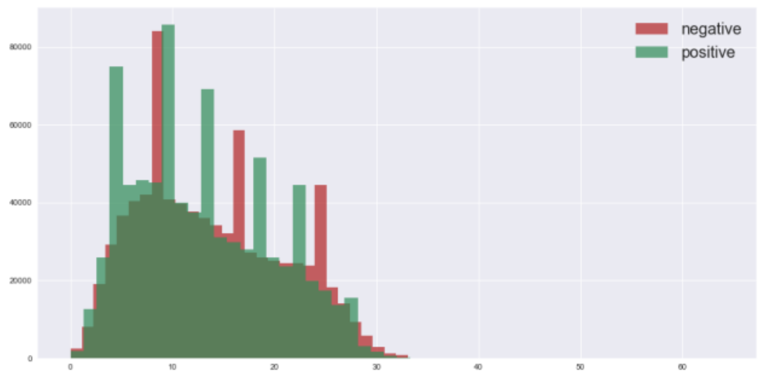


Figure 4.4: Number of words distribution for positive and negative pre-classified tweets.

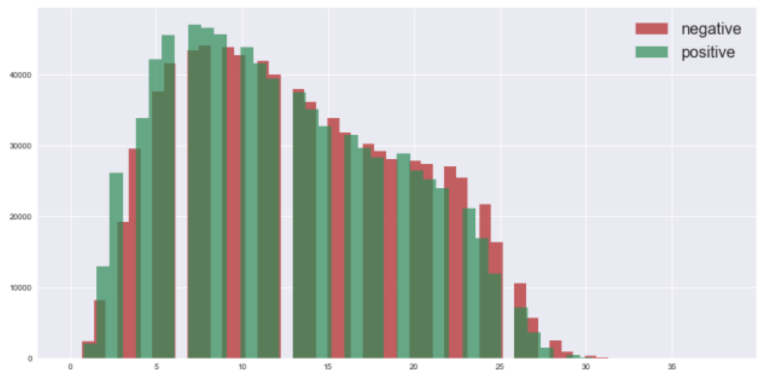


Figure 4.5: Number of unique words distribution for positive and negative pre-classified tweets.

Lastly, average word length and punctuation count distributions are plotted in logarithmic scale in Figure 4.6 for viewing purposes since the vast majority of the tweets being on the lower end makes it impossible to view the right tail. Looking at the histograms there doesn't seem to be any significant difference between the groups in both cases.

From the feature analysis we can conclude that even though the negative tweets tend to be lengthier and wordier than the positive, the differences are so marginal that we can safely say that the two groups are almost identical.

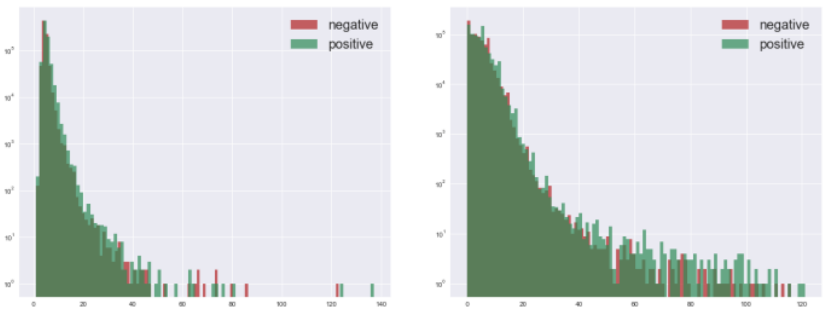
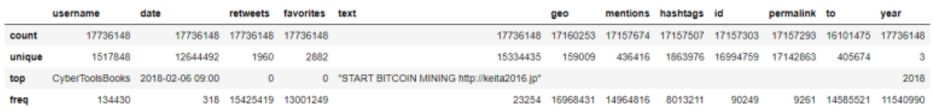


Figure 4.6: Average word length distribution (left). Number of punctuation distribution (right) for positive and negative pre-classified tweets.

4.2 Data for classification

Moving on, an overview of the unlabelled Bitcoin tweets data set is necessary, as it is on this data set where the trading signal and thus the trading strategy will be based. Starting with an overview of the set like we did before, we get the following (Figure 4.7).



	username	date	retweets	favorites	text	geo	mentions	hashtags	id	permalink	to	year
count	17736148	17736148	17736148	17736148	17736148	17160253	17157674	17157507	17157303	17157293	16101475	17736148
unique	1517848	12644492	1960	2882	15334435	159009	436416	1863976	16994759	17142863	405674	3
top	CyberToolsBooks	2018-02-06 09:00	0	0	"START BITCOIN MINING http://keita2016.jp"							2018
freq	134430	318	15425419	13001249	23254	16968431	14964816	8013211	90249	9261	14585521	11540990

Figure 4.7: Tweets for classification dataset overview.

From this description, the following can be noticed:

1. The data set contains much more information (columns) than needed for the NLP analysis. In the following figure, we can see a sample of the data that will be actually used for the NLP classification.

2. From the unique user number, it can be calculated that each user in the data set has sent approximately 11.67 messages. In reality, as we will see this ratio might be way smaller for real users since the current dataset also contains spam or tweet bots, which tend to output a lot more messages than an average user.

3. Another indication for spam is the number of duplicate texts contained in the set. This number corresponds to around 2.4 million tweets that needs to be addressed moving forward.

4. The current dataset contains tweets about Bitcoin, posted during the period of one and half years and specifically from 2017-08-01 to 2019-01-22.

For the inspection of the data, I performed the same analysis as on the training data set and looked for similar distributions in text length, number of words and number of unique words. Looking at Figure 4.9 it seems that this is the case, as distributions seem to follow similar patterns as before. This is encouraging as it is important for the model to be trained on data close to those it will eventually evaluate, in order to achieve optimal performance.

As stated before another strong indication of spam is users that have posted an unrealistic number of tweets during that period. To pinpoint such users the number of tweets per user distribution needs to be plotted. The right tail of the distribution in Figure 4.9 gives us the number of users suspected of being spambots. We can look at a sample of tweets sent by these users and calculate the average number of tweets they output in a day on average, to get an indication of a daily threshold. In order to be able to see the outliers, the histogram is plotted on a logarithmic scale.

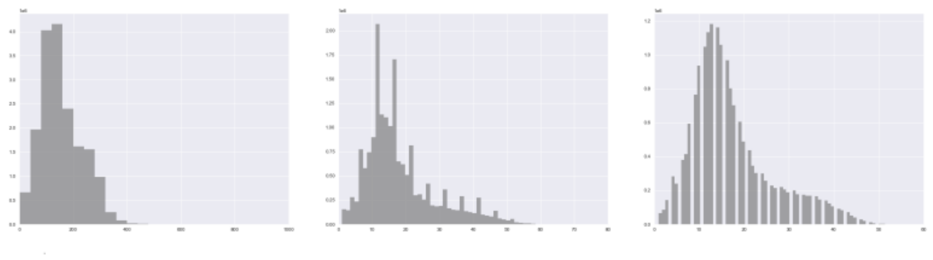


Figure 4.8: Average tweet length distribution (left). Number of words distribution (middle). Number of unique words distribution (right) for the unlabelled tweets.

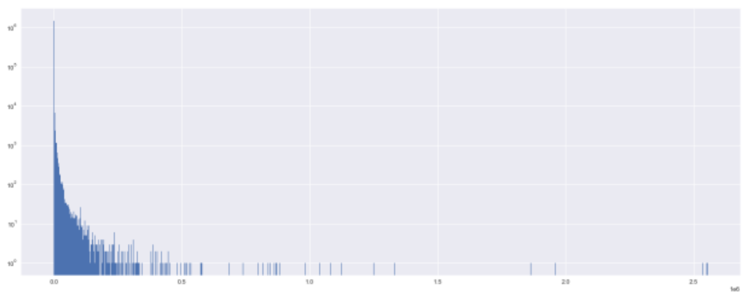


Figure 4.9: Tweets per user distribution for the unlabeled tweets

Another analysis that we can make in this data set is to get the number of tweets per month, weekday and hour. This will give us insights about the average twitter activity during certain periods and maybe also reveal information about the geographic location of the majority of the users included in this dataset. Firstly looking at the months chart in Figure 4.10 we can see an obvious decrease in activity during the summer months which is expected for people living in the north hemisphere not being so active while on vacation.

Moving on to weekdays, from the analysis presented in Figure 4.11 it is obvious that less people tweet during the weekend which can be taken into account when examining the significance of a signal during these days. Lastly, a view on hourly activity (Figure 4.12, reveals an increase during 7:00 - 17:00 Central Daylight Time (US) or 14:00 - 00:00 Central European Summer Time. Hence we can expect the majority of tweets to come from Europe and US, as these results are inline with those of past works related to this topic [32].

Finally, a look at the evolution of the number of tweets over different timeframes is necessary, since multi-timeframe profitability is an important criterion of the fitness of the trading strategy implemented. First, the number of tweets per year is plotted in Figure 4.13 but it doesn't reveal much, since the only full year for which we have data is the year 2018.

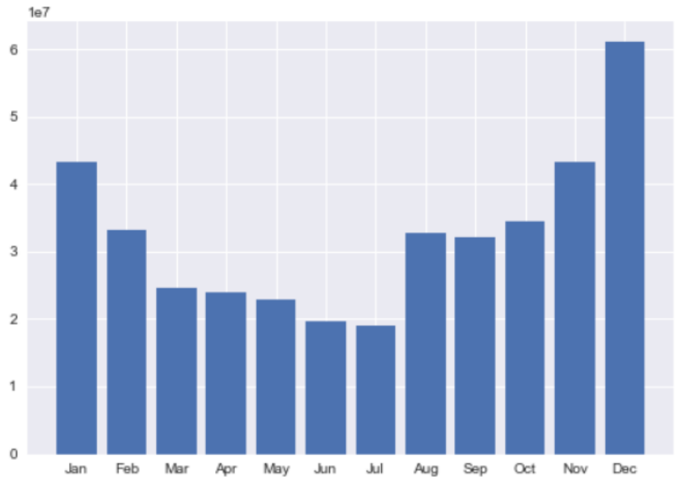


Figure 4.10: Number of unlabelled tweets per month

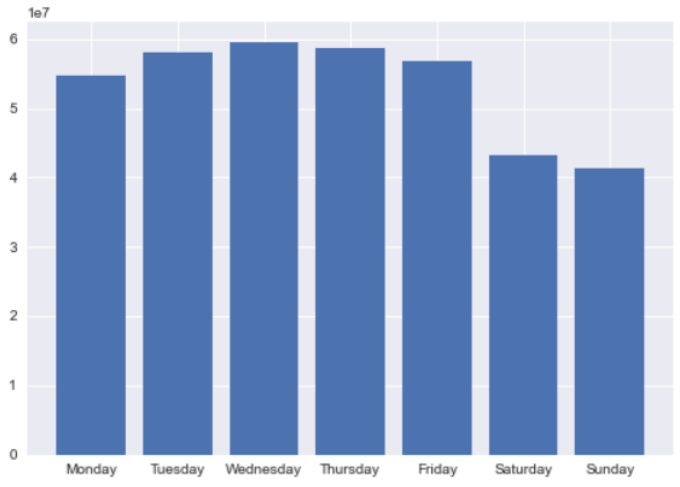


Figure 4.11: Number of unlabelled tweets per weekday

Next, an overview of number of tweets in monthly, daily, hourly, and minutely timeframe is given in Figure 4.14. In these charts, we look for inconsistencies like huge spikes or dips and verify that are few in number. Such inconsistencies might be caused by many factors varying from significant price movement resulting in abnormal twitter activity, to even defective scrapping by the researchers that gathered this data.

Indeed such inconsistencies are detected more often in smaller timeframes but they average out for bigger periods resulting in a cleaner looking behaviour overall. Another interesting fact is that

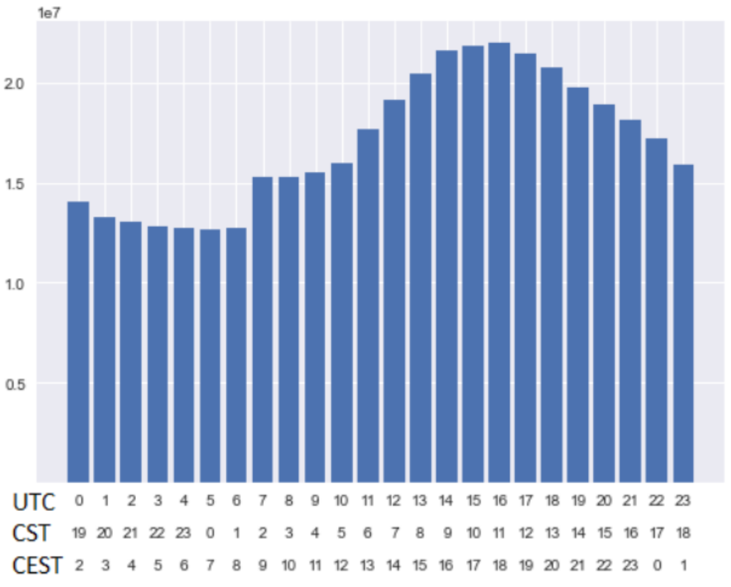


Figure 4.12: Number of unlabelled tweets per hour

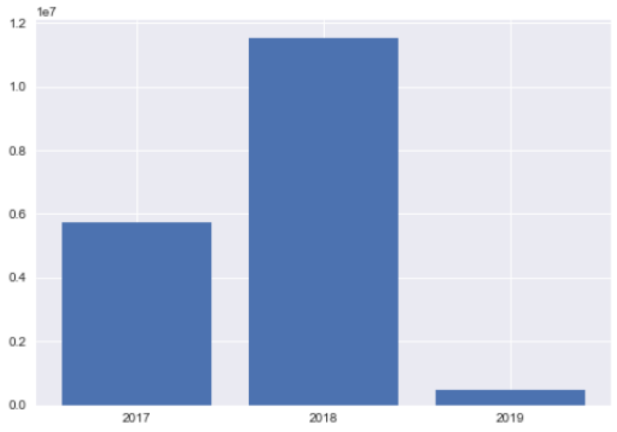


Figure 4.13: Number of unlabelled tweets per year

the number per tweet resembles the actual price of bitcoin during that time, again this is especially obvious in the monthly chart where the noise is minimal.

In order to see that in more detail, I once again plot the number of tweets for each of these timeframes against the Bitcoin Open price for the same timeframe both min-max normalized for visualization purposes.

Figure 4.15 reveals some strong correlation between the monthly bitcoin price and the monthly number of tweets. In fact the number of tweets behaviour front runs that of the bitcoin price, meaning that if we were to use that statistic to predict to drift of Bitcoin price over the next

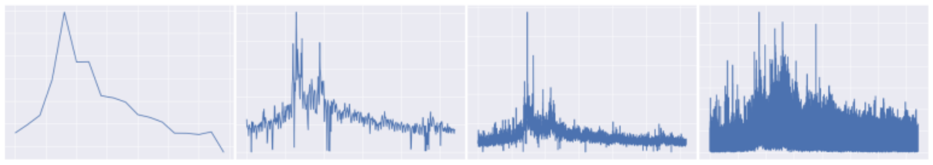


Figure 4.14: Number of unlabeled tweets monthly, daily, hourly, and minutely timeframe (left to right).

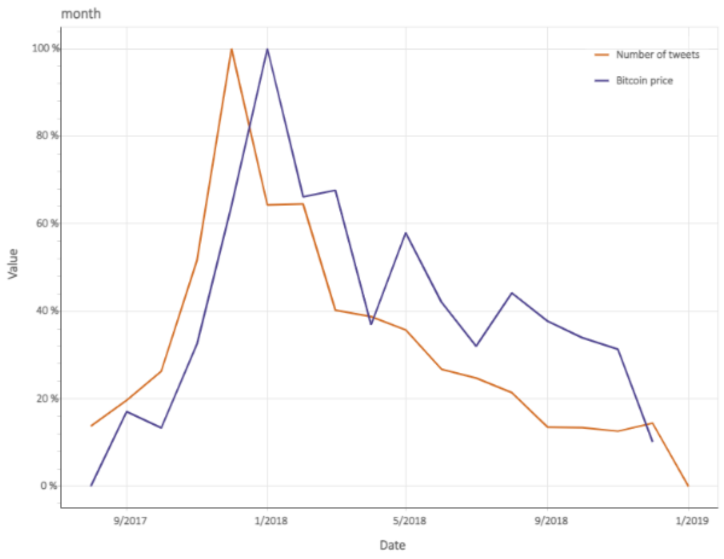


Figure 4.15: Number of unlabelled tweets and Bitcoin price in monthly timeframe.

month, we would have made profits during that period. Moreover, it can be viewed as evidence that social media bear some forecasting power in cryptocurrencies. An intuitive explanation can be also given as in general when the popularity of an asset peaks, the number of potential buyers caps, following a decrease in price. On the contrary when an asset is overlooked the sellers are being squished an increase in popularity might cause imbalance in the order books towards the buy side. However this effect is becomes less prominent in lower time frames as seen in Figures 4.16, 4.17 and 4.18 below.

Again this is expected as noise related to the parsing method used, the week day and the data used kicks in, one cannot be sure about the significance of a dip or spike in the number of tweets. However to goal of this report is to go further than that, by understanding and evaluating the contents of these tweets rather than merely looking at their number. This evaluation might give us greater insight regarding the current views of social media users and their future decision making. Of course, even in that case we should expect greater amount of inconsistencies appearing in low time frames, but before dealing with that we first should handle another more noticeable source of noise in our dataset.

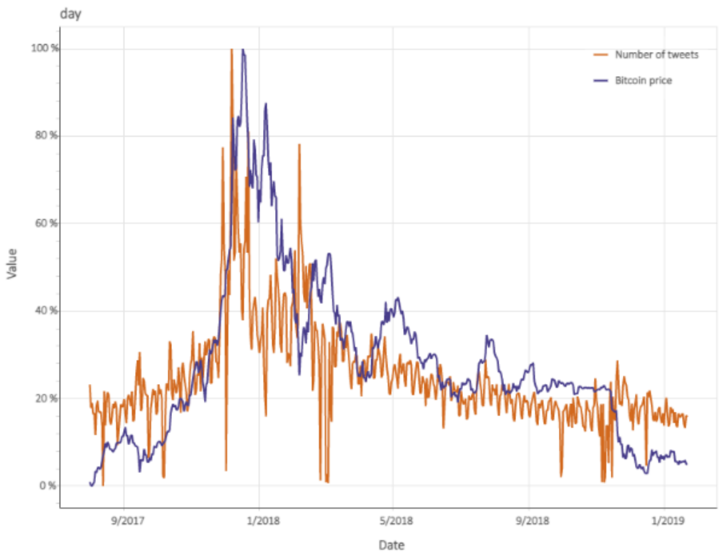


Figure 4.16: Number of unlabelled tweets and Bitcoin price in daily timeframe.

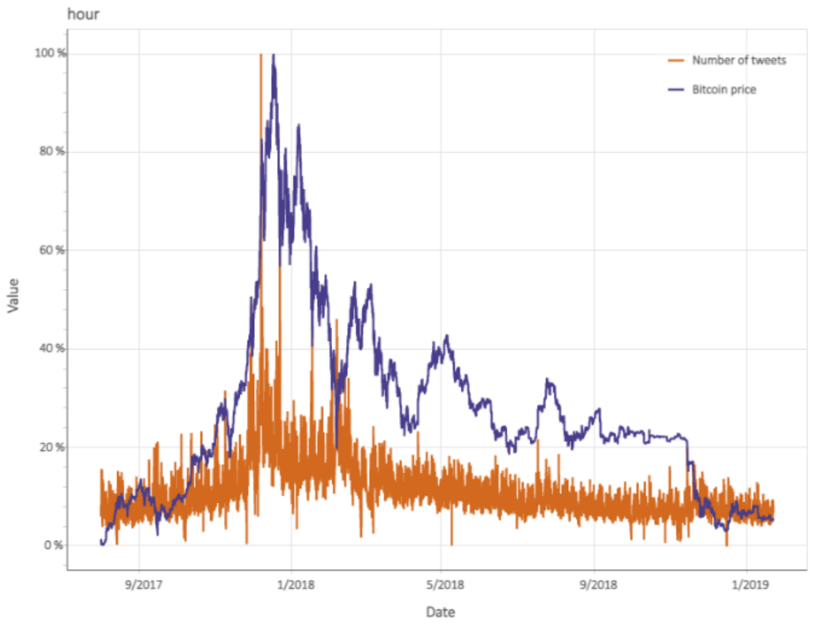


Figure 4.17: Number of unlabelled tweets and Bitcoin price in hourly timeframe.

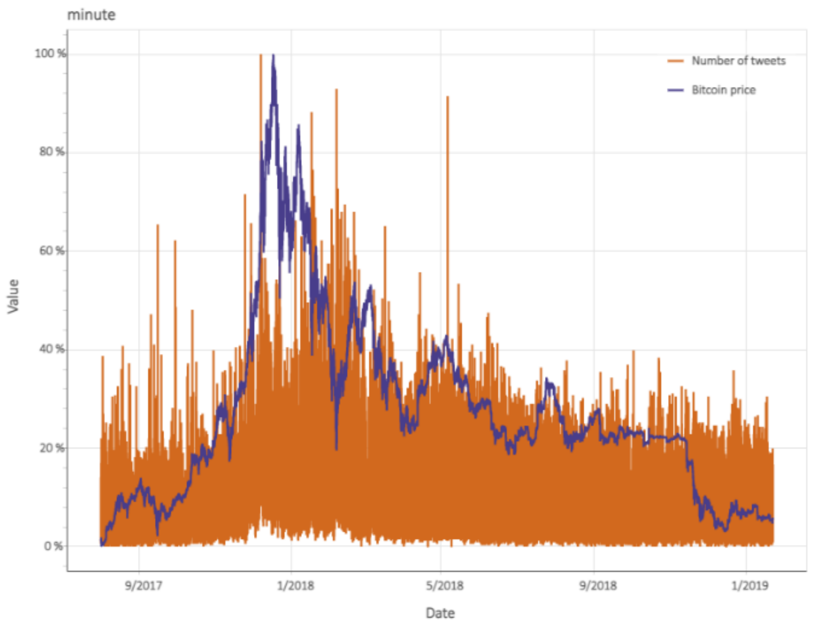


Figure 4.18: Number of unlabelled tweets and Bitcoin price in minute timeframe.

Chapter 5

Data Cleaning and Preprocessing

From analyzing the data in the previous chapter, the existence of spam in the data has been noticed. This is problematic for two reasons, first, it adds noise to the calculation of the signal since spam messages do not contain any valuable information and second spambots tend to output a huge amount of tweets which amplifies their effect in the final signal. Moreover, understanding the purpose and content of spam tweets which are usually to sell or advertise, one can expect these messages to have an optimistic tone on average which if included would bias our classification towards positive sentiment. Hence, cleaning of the data set is necessary.

In order to classify something as spam, we need to know how it looks like in our case We can do that by getting a sample from the most frequent posters assuming they are most likely spambots.

Before moving any further it should be made clear that not all spam is the same, and thus not all should be handled in the same way. Dealing with spam on e-mail is a much different task since in that case, false positives are devastating since wrong classifying a mail as spam may cause one to lose an important or urgent e-mail [33]. In our case, however, it only causes us to lose a data point that can be overcome. Optimally the filtering criteria should be such that correctly identify a high percentage of spam messages without too many false classifications. For that matter, I developed several mechanisms to spot suspicious tweets or users.

1. (C1) Excluding suspicious users from the dataset. Here I have excluded once and for all, users that have sent the exact same message multiple times. This is the definition of spam so it is safe to completely exclude these users from our dataset.

2. (C2) Another idea is to also exclude users have posted several tweets in one day that exceeds a predefined threshold (e.g., 100 tweet per day). Abnormally increased activity is another main characteristic of spam usernames.

3. (C3) Removing messages that contain suspicious n-grams [34]. For example, a suspicious 2-grams would be "free bitcoin" or "learn more", which are phrases commonly used for advertisement and have been seen frequently in this data set as well. This method relies too much on the n-grams we check for. The good choice is to use the n-grams that appear most frequently in suspicious tweets so that we get a high number of true positives and a low number of false negatives. To find the most common n-grams in spam tweets we can either:
   (a) use the results of existing research in spam filtering or
   (b) find these n-grams that appear with high frequency in spam messages and with low frequency in non-spam messages, by performing analysis in a small subset of spam messages from our data set.

We can test the effectiveness of each method by drawing random samples of the tweets that they flag. Checking the first two criteria is simple as they only require finding the number of tweets per user per period and finding of duplicates, both feasible with some data manipulation. The third method is slightly different as it takes into account the context of each tweet, hence it heavily depends and what we do classify as "spam" in this setting. Here a tweet is considered to have informational value if it reflects the opinion of an individual in some way while the rest of them are unwanted. After inspecting some samples I understood that the unwanted category is populated by bots that advertise products or services and news bots that constantly post headlines from other sources.

Going back to the n-gram detection method, I tried both methods, that is, excluding tweets that contained suspicious words according to existing research on spam filtering and calculating the suspicious words for our set by performing word frequency analysis in small subsets of the current data set. Unlike the first method which is quite straightforward, finding the n-grams popular in spam requires classifying some data as spam by hand. To expedite things I classified a small number of tweets as "spam"/ "not spam"/ "N/A" and kept a score for their senders. If a username has over 2 tweets classified as spam then it is categorized as spam user and the contrary for the "not spam" group. In the case of both then the username gets excluded from both. Now I can find all tweets outputted by these usernames of each group which provides me with a greater amount of classified tweets than those classified by hand. This is true only by the assumption that if a user outputs at least 2 tweets that can be definitively characterized as either spam or not spam then all future messages by the same user also inherit this classification. After acquiring the two opposite sets, I can calculate the frequencies of n-grams contained in each one and pinpoint from that, the n-grams that appear much more frequently in the "spam" set than in the "non-spam" set. Those are the n-grams that have a greater probability of appearing in spam than in a non-spam group, thus can potentially work as an accurate signal of suspicious tweets.

In Figure 5.1 the frequencies of the most common words are plotted for each group. The plot is on logarithmic scale for visualization purposes since the words "bitcoin" and "http" dominate the set. From the bar chart, we see that the words "reddit", "http", "news" and "cryptocurrency" appear more frequently in the spam than in the non-spam set, which gives us an idea of what "spam" in our data looks like.

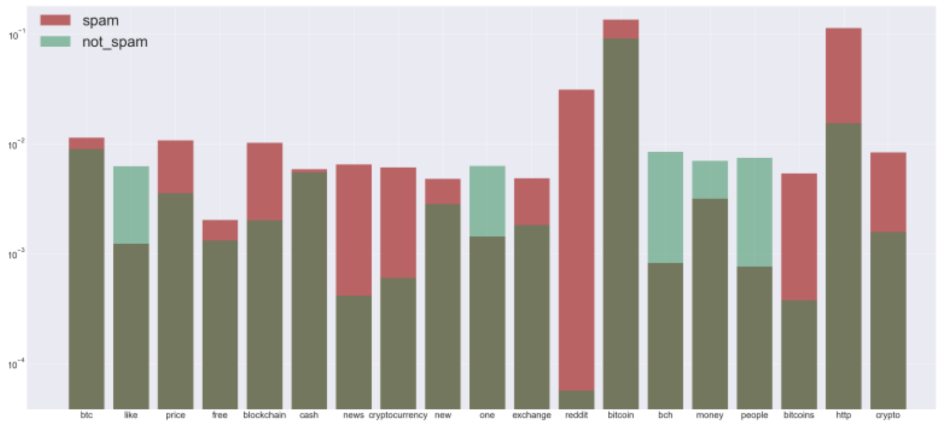


Figure 5.1: Frequencies of most common words (1-gramms) in each group

By utilizing the difference in frequencies of such words between the groups we build the set of spam words that when found in a text associate the highest probability of being spam to that text. This is calculated as follows:

Events:

s: text is spam

w: word is found in text

Probabilities:

$P(s \mid w)$: text is spam given that a specific word is found in it

$$P(s \mid w) = \frac{P(w \mid s)}{P(w)} \times P(s) = \frac{P(w \mid s)}{P(w \mid s) + P(w \mid s')} \times P(s)$$

Where $P(w \mid s)$ and $P(w \mid s')$ are the probabilities that the word in the text, given a text is spam or not spam respectively. These probabilities directly relate to the frequencies calculated before for the two sets. $P(s)$ is the probability of the tweet being spam which is the same for all tweets in our data set. By weighting each word by its $P(s \mid w)$ probability we construct the word cloud presented in Figure 5.2 below.

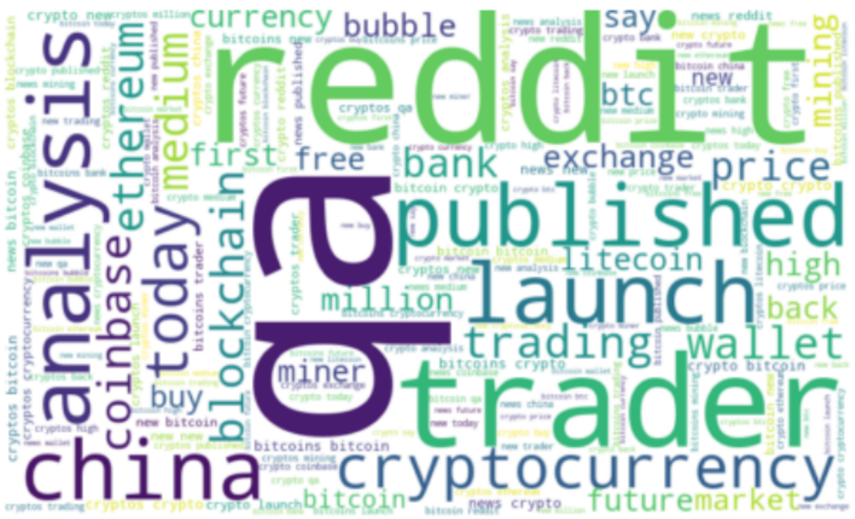
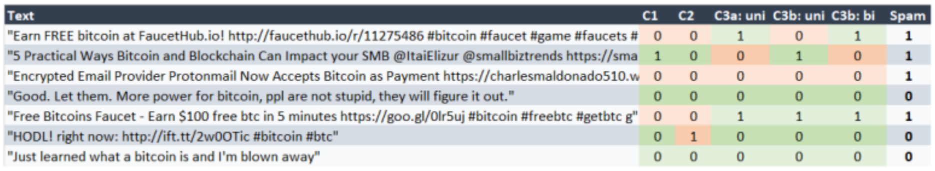


Figure 5.2: Spam words word cloud

For the word cloud, I used only the words that appear more often than the 99% of the words in the spam set so that to eliminate the effect of very unique words that happened to appear in one group and not in the other during the testing. In the word cloud we can see words like "reddit" which stood out during the frequency analysis before, words that one expects to find in the spam set like "launch", "analysis", "buy", "trader", "free" and others that is unclear why they are in the set like "qa" and "china". In addition, one can construct the data set of the word that favor a non-spam classification and cross-examine the two.

The aforementioned method however bears, of course, some drawbacks. The classification was performed on a limited amount of data which makes results difficult to generalize, while the implications of possible misclassification may be catastrophic. Hence, testing the effectiveness of these methods is necessary. To do that, I sampled 200 lines of data from the textual data set classified them by hand, and then by each of the three criteria. A smaller sample of this classification is presented in Figure 5.3.

Here "1" stands for spam and the opposite for "0". Moreover from the C3 method I used, commonly found words in spam and data set specific uni-grams and bi-grams associated with high spam probability. In the above sample, only in one case, every spam method failed to catch the spam message and the same goes for a false positive classification. It is important to minimize False Positives as much as possible since eliminating valid data points is unwanted behavior. Now, to analyze the effectiveness of each method in more depth I calculated their accuracy and true/false negatives/positives which are presented in Table 5.1.



Text	C1	C2	C3a: uni	C3b: uni	C3b: bi	Spam
"Earn FREE bitcoin at FaucetHub.io! http://faucethub.io/r/11275486 #bitcoin #faucet #game #faucets #	0	0	1	0	1	**1**
"5 Practical Ways Bitcoin and Blockchain Can Impact your SMB @ItaiElizur @smallbiztrends https://sma	1	0	0	1	0	**1**
"Encrypted Email Provider Protonmail Now Accepts Bitcoin as Payment https://charlesmaldonado510.w	0	0	0	0	0	**1**
"Good. Let them. More power for bitcoin, ppl are not stupid, they will figure it out."	0	0	0	0	0	**0**
"Free Bitcoins Faucet - Earn $100 free btc in 5 minutes https://goo.gl/0lr5uj #bitcoin #freebtc #getbtc g"	0	0	1	1	1	**1**
"HODL! right now: http://ift.tt/2w0OTic #bitcoin #btc"	0	1	0	0	0	**0**
"Just learned what a bitcoin is and I'm blown away"	0	0	0	0	0	**0**

Figure 5.3: Spam filtering techniques classification sample.

Metrics	$C1$	$C2$	$C3a : unigrams$	$C3b : unigrams$	$C3b : bigrams$
Accuracy	38%	40%	38%	46%	37%
True Positives	12	22	8	30	6
Accuracy	64	58	68	62	68
Accuracy	4	10	0	6	0
Accuracy	120	110	124	102	126

Table 5.1: Spam filtering methods evaluation results.

As seen in these metrics, some methods are more conservatives than others, as they classify only a few messages as spam with great precision. On the contrary to achieve greater accuracy overall one needs to allow for some false positives. The results shown are quite average and from the low number of positives seems that not all characteristics that classify a tweet as spam have been captured. Other more sophisticated methods [35] [36] involving language models and neural networks can be applied to achieve greater results overall that also use the context of a message to make classifications. However, this exceeds the scope of this research so instead, for our purposes a combination of the above methods will be utilized. Finally given that spam is most likely a news bot or some type of advertisement they almost always will include some type of URL. Using that characteristic I immediately classify as "non-spam" all tweets that don't contain hyperlinks, the contrary is not true.

Chapter 6

Natural Language Processing

Now that the analysis of the data has been completed, their structure, contexts, and features have become more apparent, which enables the construction of more suitable language models for classification. The idea is to construct a black box which when fed with a tweet will output a number signifying the sentiment expressed by the tweet's context, either positive or negative. By repeating the process for all tweets in the data set and by aggregating these values for longer time periods, the trading signal will be constructed. As a result, the efficacy of this "black box" also known as the classifier, will strongly affect the quality of the final signal.

For constructing a sentiment classifier, there exist several alternatives and so one can follow many different approaches. In this section four different models are presented, all of which have produced great results in the recent past in text classification but slightly differ in the ideas they introduce to the field of Natural Language Processing. Every model requires supervised learning [37, p. 9], which involves training on labeled data that have been first prepossessed and brought to the correct form for each model to handle. Progressing from the simplest to the more complex models, I present the implemented classifiers, evaluate and compare their results.

6.1 Language Models for Sentiment Classification

6.1.1 Baseline model with LSTM

The first classifier used consists of only 3 layers; an embedding layer, a bidirectional LSTM with a dropout layer, and a dense output layer. Prior to training this model, a basic prepossessing of the data needs to take place. This is initiated with the removal of mentions, URLs, punctuation, and stop words from the text as they don't usually reveal any particular sentiment for the text. The only exception to that is perhaps the punctuation, but due to the symbol-heavy nature of tweets (hashtags, mentions, emojis), such modification is necessary. Finally, words are lowercased so that there exists a unique representation for each word.

The next steps are to split the tweets into lists of separate words. Then a word index is created which similar to a dictionary, assigns a unique integer value to each word in the corpus. The split sentences are then transformed into lists of integers by replacing each word with its key in the index. This whole operation is known as tokenization. Now that each tweet is represented by lists of numbers, the final step is to enforce a fixed length on them, to be handled by the model. This procedure is called padding, where after setting a fixed threshold, every tweet-sequence is enforced to the same length. Specifically, tweets with a lower amount of words than the predetermined threshold have 0's appended to them while lengthier tweets have words taken out. Hence the final fixed length of the sequences is chosen so that the vast majority of tweets can be represented without removing words from them. From the analysis presented in Figure 4.4, one can see that a length of 50 should be enough for almost every tweet in our data set. In Figure 6.1 an example of tokenization and padding procedure with fix length 10 is illustrated in order to consolidate our understanding of the preprocessing stage, as it is quite similar for most of the implemented models we will develop.

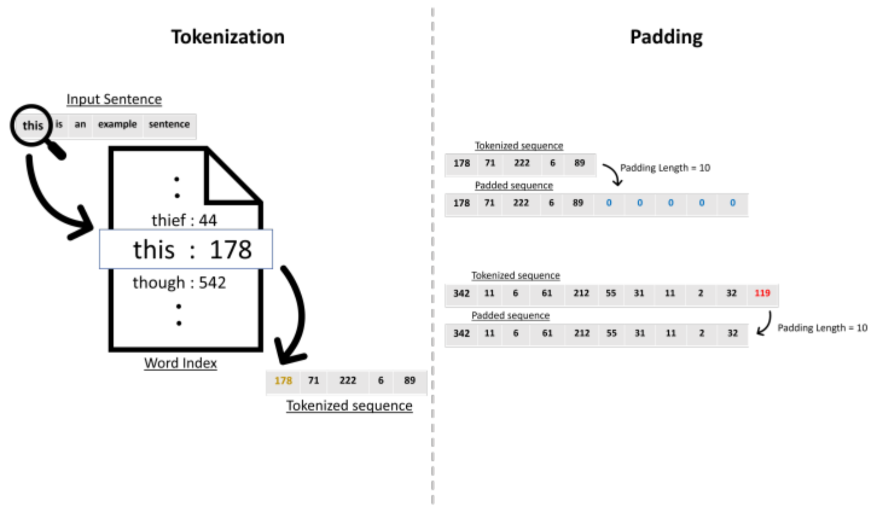


Figure 6.1: Tokenization and padding procedures illustrated example.

Once the input sequences are all of the same lengths, the next step is to pass through an embedding layer. Contrary to a dense layer that performs the dot product operation and adds a bias vector, an embedding layer performs the select operation. Its functionality is to convert each word token into a fixed-length vector of defined size. There are several ways to convert the word features to input features. A common technique is to use one-hot encoding to transform these indices into vectors of 0's and 1's and use dense layers instead. In that case, however, the vectors would be of the size of the dictionary that would increase the dimensionality of the features' space severely. To avoid the "curse of dimensionality", embedding layers are used, which map each word to a vector of a smaller, fixed-size vector of real numbers, allowing for an infinite number of possible encoding with finite vectors.

The vectors produced by the embedding layer are then fed to a bidirectional LSTM with a dropout layer. An LSTM consists of a group of recurrently connected blocks, known as memory blocks [38]. These feedback connections allow them to deal better with long time lags between the inputs and their corresponding outputs, introducing some form of memory to the model. It is that trait that makes LSTMs well-suited for language modeling since there can be lags of unknown duration between important events in a time series. Introducing bidirectionality to the network enables the processing of data in both directions, utilizing both the previous and future context of the text. And while may appear unconventional at first, it often that sentences gain meaning after the reveal of their future context. Bi-directional LSTM's exploit these language characteristics to produce even better results and have an increased understanding of context relative to their unidirectional counterparts. The Dropout layer after the LSTM is used to prevent overfitting.

Lastly, the output layer is a dense layer with a sigmoid activation function. Since we have a binary classification task, I need an activation function on the output layer to produce a bounded output of range (0,1). Another option would be tanh but it has a range of (-1,1) which doesn't work for the pre-labeled we have data that are labeled 0 for negative and 1 for positive. The output of the model is a real number ranging from 0 to 1, with 0 being absolutely negative and 1 being absolutely positive. This value is the confidence score, which answers how certain is the model for the classification, which is basically the confidence score rounded. Here a reasonable question arises, as to whether one should use the confidence score or the classification completely disregarding the certainty of the model about it, something that will be examined in the next section.

6.1.2 The GloVe model

The previous model seems to be well suited for the sentiment classification task of categorical data, yet it allows for improvements. The biggest flaw of this implementation derives from the fact that embeddings carry none of the contextual meaning of the encoded-word. Fortunately, other NLP models have been developed and trained on huge amounts of data to understand relationships between words. Then by utilizing these relationships, vector representation that reflects some of the contexts of the word can be created. Such trained word vectors are definitely a superior method of word encoding, hence by incorporating them into our existing model one can expect superior results. To test that GloVe pre-trained word vectors have been recruited.

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. GloVe vectors offer a great way of encoding words to number vectors, so that they can be handled and classified by the model while inheriting both the meanings and the relationships between the words they encode. This is achieved as GloVe utilizes the frequency of word co-occurrences to construct word vectors that directly relate to the probability that these words co-occur in the corpus. Through this embedding [10], it is possible to incorporate word context into the sentence and therefore to the classification. Lastly, GloVe developers offer pre-trained word vectors, trained on tweets which satisfies the needs of this project to the fullest.

Following the same procedures as before data are preprocessed, tokenized, and padded. Then the downloaded GloVe pre-trained vectors are imported and from them, the embedding dictionary is constructed mapping each word to its vector representation. Lastly, the word index is updated with the embedding of each word in the corpus instead of a single number representation. The final set is passed into the embedding layer which will provide the encoding of each word it encounters. The input dimension in this layer should be set to match the dimension of the word vectors provided by GloVe developers which in this case is 100.

6.1.3 The ELMo model

It has now been made clear that the quality of the classification is strongly correlated with the quality of the word embeddings we use. GloVe was a big leap forward going from random embeddings to pre-trained vectors that inherit the meaning of words. However, it is possible to go one step further and have encodings that associate not only with the meaning of a word but its contextual information in the particular sentence. This is incentivized by the fact that the same word can appear the same in two instances and have much different meaning like the words "stick and "left". To allow for such representations I used ELMo a model trained on large datasets to understand language by next word prediction.

Preprocessing with ELMo is similar to the previous models. An exception is the removal of stopwords and punctuation which is avoided since ELMo can utilize them as extra information for the context of the sentence. Tokenization and padding are still necessary for ELMo to handle the input. The model's architecture consists of an input layer, ELMo embedding layer, bidirectional LSTM, and a Dense layer with Dropout for the output.

6.1.4 The BERT model

Lastly, I experimented with the BERT model [20]. BERT is a bidirectional sequence to sequence model that uses transformers (encoders), a neural network that seems to deal better with long and short-term memory than LSTMs do. It was introduced by Google and has been quite popular due to its many different applications. BERT has been trained to find the missing word in a text and answer questions will allows it to extract the contextual meaning of each sentence. While BERT was initially implemented to improve machine translation (being a sequence to sequence model) its ability to understand language made it stand out and eventually become the best NLP model on several other tasks, one of which is text classification.

To use BERT as our classifier we simply pass as input a $< CLS >$ token at the beginning of each sequence. Then at the output layer, we take the first element of the sequence and pass it to a custom classifier built on top of BERT. Another choice we can make is if we want BERT model parameters to be trainable or not. In the first case, we fine-tune the BERT model to optimally perform our task, while in the second approach we only train our classifier on the data. The fine-tuning of BERT is the suggested method to use BERT according to its developers as it offers greater adaptability with some time and memory overhead. Having tested both, the trainable BERT seems to perform much better than its non-trainable counterpart indeed, yet it introduces over 170M parameters which make it slow to train and cumbersome to use especially on conventional machines since the use of a GPU is absolutely necessary.

Unlike the previous models BERT doesn't need separate word vectors as input, in fact, it calculates its own representations internally, which are accessible from its hidden layers even though it is not of any importance for this application. Moreover, the removal of stopwords and punctuation is not necessary nor suggested, since the model is pre-trained on whole sentences. Additionally, stopwords even though overused in language still provide context which BERT can utilize for the classification. Still, tokenization and padding of the inputs are performed, this time using the model's tokenizer for the former. That is the case because BERT's tokenization is not the standard procedure of mapping each word to a unique integer. In addition to that special characters are included in each sequence such as the $< CLS >$ token mentioned before and the $< SEP >$ representing whitespace. Finally, BERT requires two additional inputs to function; the input mask and the input type ids. The input mask, used mainly for word prediction in this case indicates which input ids correspond to padding, having value 1 where tokens are and 0 for padding. Similar to that the input type ids signify the position of actual words (excluding tokens) in the input.

After the preprocessing has taken place, the data are passed into the final model. The model can be distinguished into 3 main parts: the input, the BERT language model, and the classifier. The input is essentially 3 input layers for the three sets of input constructed. The input layers are connected to BERT which undertakes the majority of the computations and returns two types of output; the pooled output and the sequence output. The former holds representations for the entire input sequences while the second fore each token in the input sequence separately. For the task of classification, only the pooled output is needed and passed into the classifier. The classifier is a feedforward neural network of dense layers not of any specific architecture. BERT researchers have suggested that a unique Dense layer with Dropout achieves great results so this is what I went with as well. Now that the model is set the training phase can begin, which will fine-tune BERT to the Twitter data and train the classifier.

6.2 Training Results

After all models have been trained on the same data, they can be evaluated on their performance. The evaluation will determine if the added layers of complexity can be translated to an increase in effectiveness as well. There as several measures that quantify effectiveness in classification, the simplest of which are accuracy and loss. Accuracy is the main measure of performance and represents the percentage of correct classifications within an epoch. Loss on the other hand is a number that also takes into account the confidence with which the classification was made. Measurements for both accuracy and loss are taken for the training and validation set separately. In general, the results on the testing set are considered to be more representative of the model's actual skills, since they are produced from newly seen data.

As straightforward of a concept as accuracy is, it can be analyzed even further to better understand the performance of your classification model. In practice, for a binary classifier such as this, the possible errors are of two types: classifying a negative tweet as positive known as a false positive or classifying a positive tweet as negative known as a false negative. Respectively, correct predictions can be seen as true positives and true negatives. This separation between classes allows one to understand the types of error being made and investigate possible class imbalances in the evaluations. These four metrics are displayed in tables, known as confusion matrices. The confusion matrices for our models are displayed in Figure **??** below.

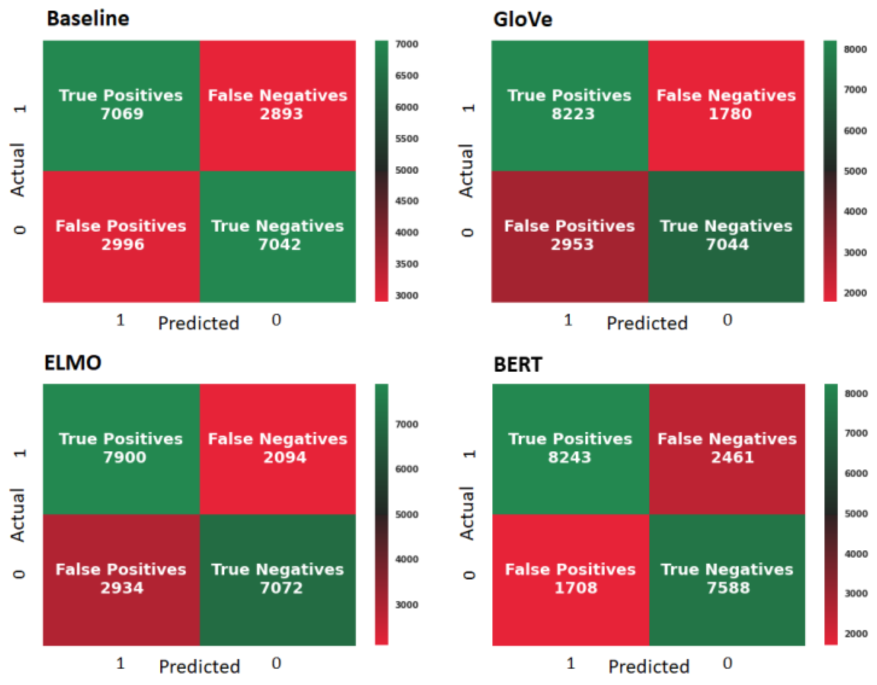


Figure 6.2: Confusion matrix for all implemented models.

Looking at the above tables, we can see a significant improvement going from our baseline model to the three more complex ones. Specifically, the baseline model seems to identify positives and negatives with the same accuracy, while this is not the case for the rest of them. GloVe, clearly better than the baseline model in both ends, seems to be predicting negative sentiment more accurately than positive ones. The opposite is true for the BERT model, that is the results indicate better performance on identifying positive sentiment. This characteristic will be very useful when constructing our trading strategy later, as what's indicated by each sentiment class may be different in non-symmetric strategies. Elmo results are near the same range but seem worse than those of the GloVe model even though it is a more complex model.

From these values four additional measurements can be extracted; sensitivity, specificity, precision, and false positive rate. Sensitivity or recall is the true positive rate and is calculated as the number of true positives over all positive labeled data evaluated. Sensitivity describes the ability of the classifier to identify positive sentiment. For identifying negative sentiment we use specificity, which is the percentage of negative labeled tweets classified correctly. False positive rate of fall-out is the number of false positives over the total number of data predicted as negatives. This measure is also known as the false alarm ratio since it describes the percentage of negative classifications that were correct. The precision or positive predictive value is the percentage of true positives over all positive classifications that the model made. Precision is also referred to as the positive predictive value. Lastly the f1-score is an average of precision and recall that quantifies the accuracy of the binary classifier. The aforementioned measures are calculated for each model and displayed in the classification report of Table 6.1.

In these tables, we can see that BERT scores higher than the rest of the models, while excelling in precision. On the contrary the baseline model trails in performance in each category. These measurements can be also illustrated graphically by certain curves. One such plot is the receiver operating characteristic (ROC) curve [39], which summarizes the trade-off between the true positive

	Baseline			
	precision	recall	f1-score	support
class 0	0.7	0.71	0.71	9962
class 1	0.71	0.7	0.71	10038
accuracy			0.71	20000
macro average	0.71	0.71	0.71	20000
weighted average	0.71	0.71	0.71	20000

	GloVe			
	precision	recall	f1-score	support
class 0	0.74	0.82	0.78	10003
class 1	0.8	0.7	0.75	9997
accuracy			0.76	20000
macro average	0.77	0.76	0.76	20000
weighted average	0.77	0.76	0.76	20000

	Elmo			
	precision	recall	f1-score	support
class 0	0.73	0.79	0.76	9994
class 1	0.77	0.71	0.74	10006
accuracy			0.75	20000
macro average	0.75	0.75	0.75	20000
weighted average	0.75	0.75	0.75	20000

	BERT			
	precision	recall	f1-score	support
class 0	0.83	0.77	0.8	10704
class 1	0.76	0.82	0.78	9296
accuracy			0.79	20000
macro average	0.79	0.79	0.79	20000
weighted average	0.79	0.79	0.79	20000

Table 6.1: Classification report for all four implemented models

rate and false positive rate for a predictive model using different probability thresholds. Essentially ROC is a probability curve that tells us how effective is the model in distinguishing between classes. Graphically, the closer the curve is to the upper left corner the better the model's performance is. This is quantified further through the area under the ROC curve (AUC) measure which represents the degree of separability ranging from 0.5 to 1. Thus in our case, a higher AUC corresponds to a better ability in discriminating between positive and negative sentiment.

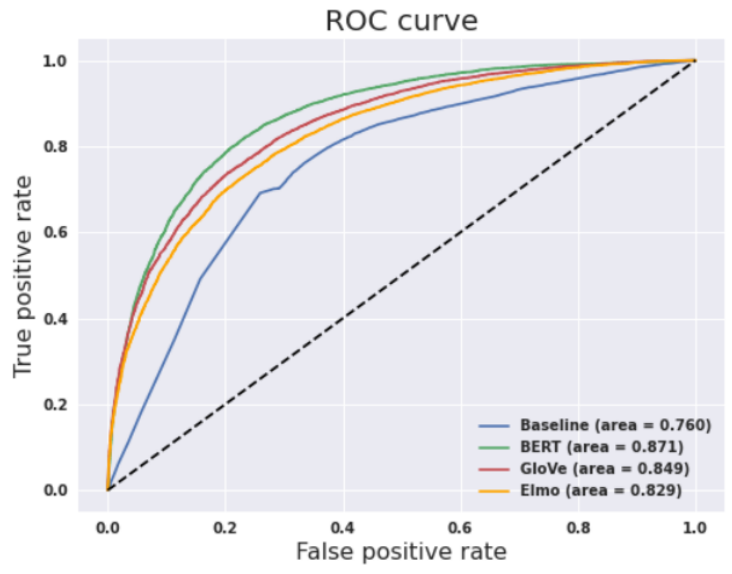


Figure 6.3: ROC curve for all four implemented models.

The ROC curve and AUC presented in Figure 6.3 also indicate the overall superiority of BERT compared to the rest of the models, but not by a large margin. GloVe even though far more simplistic and faster than BERT performs almost as well. This makes GloVe model a strong candidate, as space and time complexity are equally important as accuracy, especially in the case of online classification. As for Elmo, I found it to be the most cumbersome to use, as it requires outdated python libraries to operate and a complicated setup. Moreover is quite slow to train, while all this added complexity doesn't really translate to better results as it closely competes with the much simpler Glove. Lastly, the baseline model, however, is definitively worse than the rest of the models as expected, since it doesn't utilize any meaningful information related to the text's meaning.

Another common way of visualizing a model's performance is the Precision-Recall curve [40]. As the name suggests, the precision score, representing the ability of the model to identify positive sentiment is plotted against the recall, indicating the sensitivity of the model in identifying positives. Similar to the ROC curve, as the Precision-Recall curve approaches the top-right corner the better the model is considered to be. The recision-Recall curve for all four model is displayed in Figure 6.4

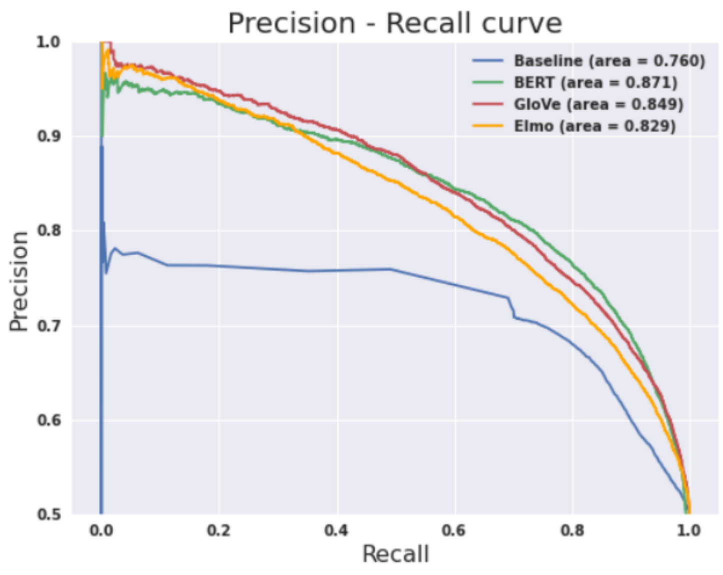


Figure 6.4: Precision-Recall curve for all four implemented models.

The four measurements reveal a certain aspect of the classifier's performance, not all equally important. The significance of each evaluation high relates to the task at hand. For example, a trading strategy that goes long on positive sentiment should value a good precision score much higher than a high recall score, since a false positive may cause money losses while false negative only results in missed opportunities. Recall would be however more important in cases where there is a higher cost associated with a false negative than with a false positive.

After analyzing the performance of the NLP deep learning classifiers, I chose to use the BERT model as the most accurate one, and the Glove model as the one with the highest accuracy-to-speed of evaluation ratio. Another advantage of the BERT model is that it is a multilingual model that has been trained to understand language so it can be easily applied to a larger variety of textual data without compromising its performance.

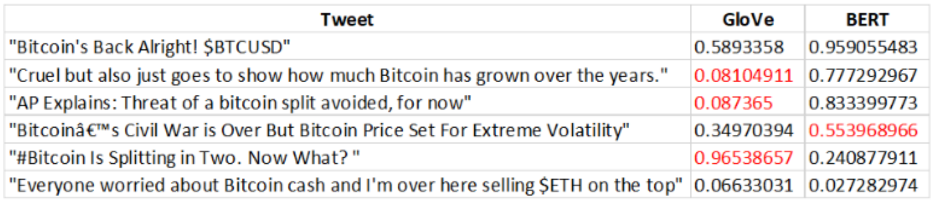
Chapter 7

Trading Strategy and Backtesting

7.1 Trading Signal Construction

Now that the NLP models are built and trained the construction of the trading signal is next. The first step is the classification of the unlabelled Bitcoin tweets. For that, the models I am going to use are the GloVe model and the BERT model as they showed great results during their training phase.

While the evaluation performed in the previous section gives us great intuition about the abilities of each model, results can differ slightly when evaluating completely new data. An example of that is that having been trained on general tweets, it may be hard for a model to associate positive sentiment with the word "long" and negative sentiment with the word "short" with high confidence. Thus, one can expect pre-trained models that have greater language understanding like BERT and Elmo, to perform better in this case since they are more adaptable in general. However their cumbersome nature and high time complexity overhead make it harder to use in an online setting. On the other hand, the Glove model is shown to combine speed and accuracy during training which makes it a legitimate candidate for the test. In order to examine the models' effectiveness on the unlabelled data set, I present a sample of unlabelled Bitcoin tweets classified by the two models in Figure 7.1 below.



Tweet	GloVe	BERT
"Bitcoin's Back Alright! $BTCUSD"	0.5893358	0.959055483
"Cruel but also just goes to show how much Bitcoin has grown over the years."	0.08104911	0.777292967
"AP Explains: Threat of a bitcoin split avoided, for now"	0.087365	0.833399773
"Bitcoinâ€™s Civil War is Over But Bitcoin Price Set For Extreme Volatility"	0.34970394	0.553968966
"#Bitcoin Is Splitting in Two. Now What? "	0.96538657	0.240877911
"Everyone worried about Bitcoin cash and I'm over here selling $ETH on the top"	0.06633031	0.027282974

Figure 7.1: Classification sample of unlabelled data

In this example, the misclassifications are indicated with red color. We can observe that GloVe's sentiment evaluation is heavily influenced by strong words in the text like "Cruel" and "Threat". On the contrary, BERT seems to understand better the meaning of the tweet. This, of course, doesn't always work out as demonstrated in the case of the 4th tweet, where BERT correctly classifies the phrase "Civil War is Over" as positive, but doesn't value correctly the second part "But Price Set For Extreme Volatility" resulting in a false classification. On the contrary, GloVe looking at the phrases "Civil War" and "Extreme Volatility" happens to correctly classify the Tweet in this case. Here one needs to be reminded that, unlike BERT, GloVe doesn't take the whole Tweets as displayed here as an input, but rather a processed version of them without stop-words and punctuation since these bear small informational value on their own. However, in the context of a sentence, this information is crucial for understanding the meaning of the tweet, and that is why BERT utilizes them to derive its final classification.

Before the evaluation can begin the data set needs to been cleaned of spam, using the criteria addressed in the section about spam filtering. This is a necessary step as spam adds noise to the final signal since its populate by tweets that don't reflect a market's participant sentiment in any way. During some test classification without filtering, I observed an elevated tendency towards optimistic sentiment, possibly due to the positive nature of spam and advertisement tweets. Furthermore for the data to be passed into a model, one needs to follow the same preprocessing steps as for the training dataset of the particular model. That also includes saving and importing the same tokenizer and word index since the encodings created by the models are based on these particular token ids. For words that did not appear in the training set, but exist in the evaluation set, we use a standard $< oov >$ (out of vocabulary) token that maps these words to a predefined representation.

After calculating the confidence score for each tweet, we need to process them and aggregate them for longer periods, to construct the final trading signal. Several choices can be made here, with the most obvious one being the average of the confidence scores for the period (e.g per hour). Another possibility is to map each sentiment to either 1 or -1 (positive or negative respectively) before averaging them out. The created signal will show what is the percentage of outstanding positively identified tweets in the current period. This excludes the confidence level of the prediction from the final signal, accounting only for the direction. We can further expand this calculation by including an additional class for neutral sentiment. In that way, we map each sentiment to -1, 0, or 1 depending on the confidence of the prediction (confidently negative, neutral, confidently positive respectively) in an attempt to increase the definiteness of the final signal. The signals calculated with each aforementioned technique are min-max normalized and presented in 7.2 and 7.3 for each model on monthly and daily timeframe resepctively.

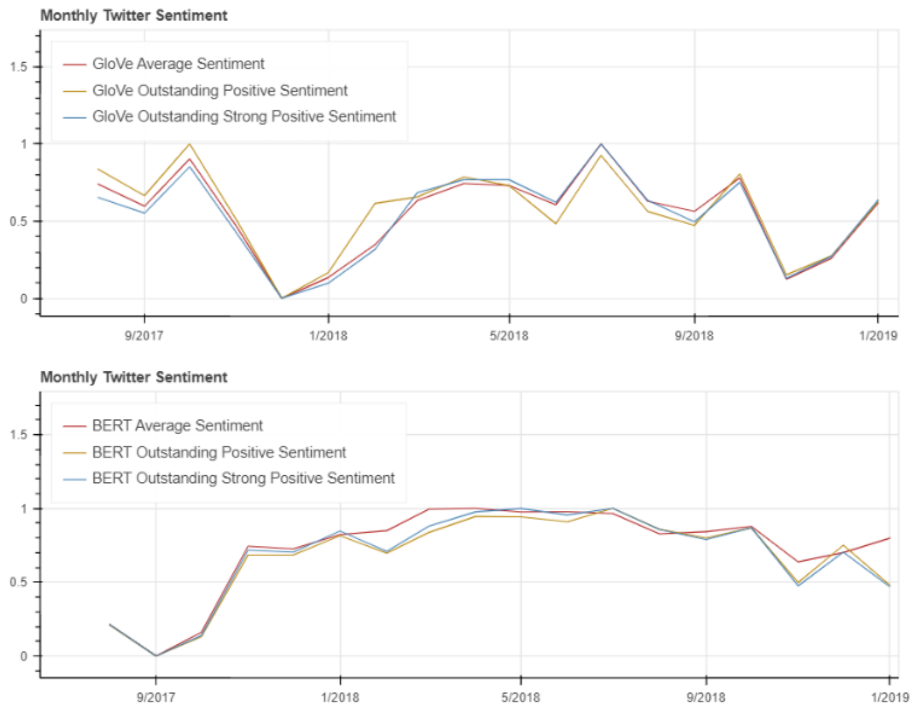


Figure 7.2: Comparison of trading signals on monthly timeframe for three aggregation techniques, for models GloVe and BERT (min-max normalized)

These charts indicate strong similarities between the results of each method. However, the

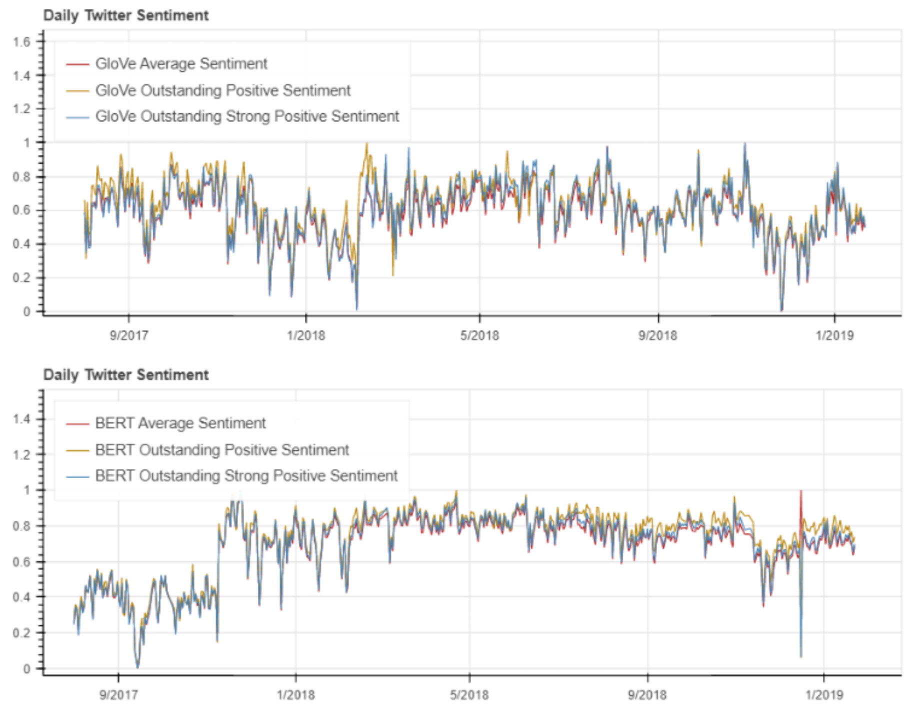


Figure 7.3: Comparison of trading signals on daily timeframe for three aggregation techniques, for models GloVe and BERT (min-max normalized)

evaluation performed by each model seems to differ substantially during certain periods especially for the year 2017. However, In lower timeframes, we can see the effect of noise in the calculation through the fluctuations of the trading signal. However, this is amplified by the min-max normalization, since without it we can see the signal agreeing on the range of the sentiment score.

Another way of combining the scores is to the first average per user for each period and then per period. The idea behind this method is to eliminate the impact of users that output a large number of messages per period. While spam filtering helps by cutting outliers, the opinion of a more "vocal" user will still be valued more than more reserved ones. Visualizing the two signals deriving from each method side by side helps to identify their characteristics. In 7.4 the two signals are min-max normalized and plotted for each model separately for the daily timeframe. Here one can see that averaging over user first can result in cutting off some extreme spikes that may appear in the final sentiment score.

7.2 Developing the Trading Strategy

After the signals have been constructed, it is time to test them by incorporating them into a trading strategy. In order to implement a trading strategy, several characteristics need to be specified. Most importantly is how the strategy will utilize the trading signal for decision making. This closely relates to the hypothesis tested in each case. For example, one hypothesis could be that in periods of extreme optimism the buyer is being depleted leading to a future decrease in price. In that case, one should go short when observing high sentiment values. On the contrary, if one wants to test the simple hypothesis that prices will increase when people are optimistic, they must

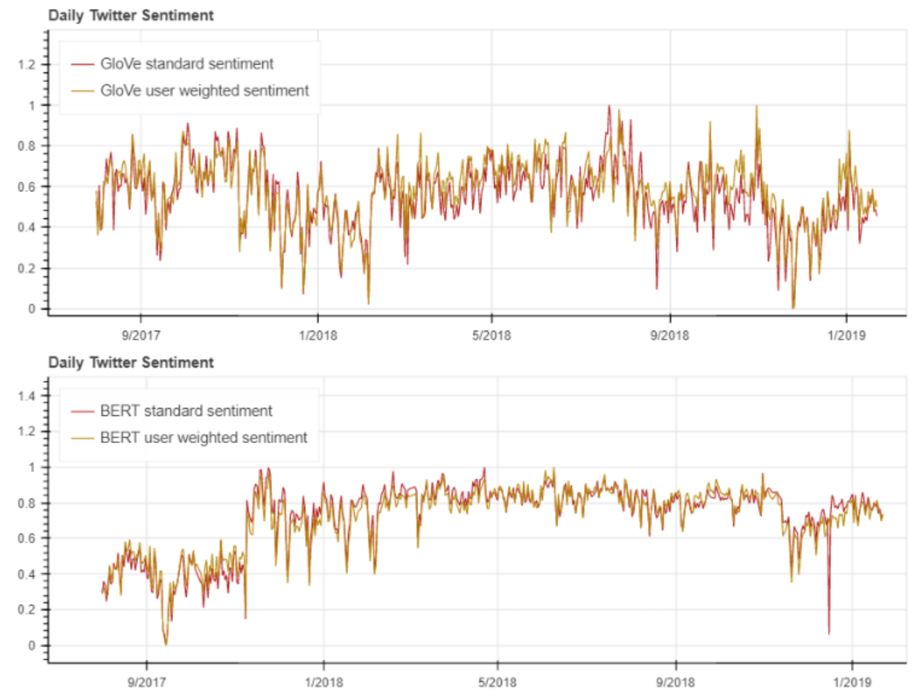


Figure 7.4: Comparison of user-averaged and non user-averaged daily sentiment for models GloVe and BERT (min-max normalized)

go long on high sentiment values. In each case the thresholds that signify the opening of a position needs to be determined. In this project, I'm examining whether signal values above a rolling mean indicate a future increase in Bitcoin's value, hence when such values are observed the strategy will buy Bitcoin.

To demonstrate this idea of using sentiment moving averages for decision-making, in Figure 7.5 the outstanding positive sentiment calculated by the BERT model is plotted along with its 50-Day Rolling Average and the Bitcoin price, all min-max normalized for better visualization. I have marked the areas where the daily sentiment is above the 50 rolling average with green where the strategy (with threshold value 1) would buy and those below with red where it would exit, we can get a better understanding of our strategy's potential. On a first look, we can see more red areas after the Bitcoin crash than before which is a wanted behavior.

The timeframe of the trading plays also a significant part in the strategy and is connected to the initial hypothesis as well since it relates to the duration of the impact of the signal. Specifically, one idea might be that the average minute sentiment can provide insights for the price movements of the next minute as its impact decays over time. A counterargument to that could suggest that is that calculations on low timeframes tend to be extremely noisy as other parameters influence the quality of the signal. Timeframe selection also associates with the number of trades we allow for during a period. More trades allow for a more reliable evaluation of the trading strategy in the same amount of time but are involve increased trading costs, while strategies with a small number of trades can be harder to evaluate as any profits can be attributed to luck. Here I develop strategies that trade on daily and hourly Bitcoin prices.

A third parameter has to do with the sizing of the positions and whether this will be a fixed

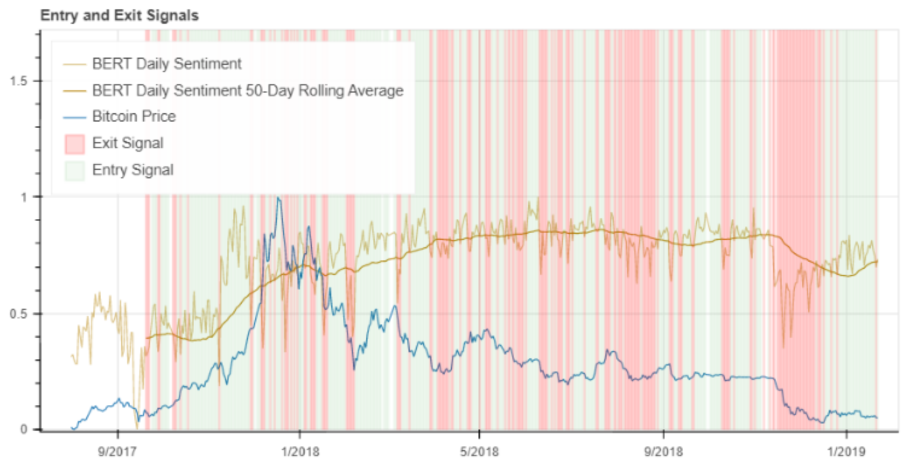


Figure 7.5: Entry and Exit Signals visualized on the Bitcoin price

amount, a fixed percentage of our portfolio value, or whether it should be solely determined by the intensity of our signal. The sizing is directly connected to the maximum number of open positions allowed at a time by the trading strategy. So whether it is better to invest the whole capital at once or average it into multiple trades. But again in that case this needs to be done carefully as it will have devastating consequences for our profit if done excessively, due to trading costs. In our case, for each trading signal, I searched for the optimal sizing and entering threshold as will be demonstrated later in this report.

Now that the criteria for entering a trade have been set, so it must be for exiting one. Exiting a strategy is equally important a can be done in several ways; after a fixed amount of time, on specific prices, through the trading signal, or a combination of the three. Fixed exit points are usually set at the end of the day for stocks since the market closes and trading is not possible. Exiting on specific prices is related to risk management and is usually done through stop losses and take profits. Lastly using the trading signal for exiting is also logical and closer to the objective of this project. Thus, the strategy exits and long position when the entry criteria are not met, while in the case of hourly timeframe temporal exits are also applied at the end of each day. The final component of our trading strategy is the analyzer, responsible for tracking and storing our trading history, which is crucial for the backtesting and evaluation of our strategy.

One can understand the possible arrangements of these specifications are endless and each can result in a much different performance. In this case, however, the core objective is not to produce a profitable strategy but rather to evaluate the informational value of our calculated signal. Therefore, I focused on constructing simple strategies that base their decision-making on the trading signal as much as possible so that any profits can be attributed to social media sentiment instead of the complexity of the trading strategy.

To summarize, the strategy operates as follows. Starting with a unit portfolio the strategy checks at each time step if the current average sentiment value is above a threshold. The threshold represents a percentage of the sentiment's scoring rolling mean value and is optimized for each signal through testing. After examination, I determined that the optimal window size of the rolling mean is 20 days for daily strategies and 200 hours for hourly ones. If this criterion is satisfied the strategy will buy at the open price of the next time step. The sizing of the positions is a fixed percentage of our portfolio which corresponds to a maximum number of concurrent open trades. This parameter is also adjusted for each signal and timeframe individually. On the contrary, if the entering criterion is not met the strategy closes all open positions on the open price of the current time step. Additionally for sub-day strategies and additional exiting condition is applied which forces the strategy to close all open positions at the close price of each day. The workings of the daily strategy are presented in the pseudocode of Figure 7.6.

```
Algorithm 1 Daily Sentiment Strategy
  Initialize: $threshold$                                   ▷ buy threshold
  Initialize: $maximum\ allowed\ positions$     ▷ max concurrent open positions
  Initialize: $current\ positions \leftarrow 0$                ▷ current open positions

  while not $final\ date$ do
      if $current\ positions = 0$ then
          $size \leftarrow portfolio.value/maximum\ allowed\ positions$
      end if
      if $sentiment > threshold \times sentiment.rolling.mean(20)$ then
          if $current\ positions < maximum\ allowed\ positions$ then
              $buy(price = market,\ size)$
              $maximum\ allowed\ positions{+}{+}$
          end if
      else
          if $current\ positions \geq 1$ then
              $close(all\ positions,\ price = market)$
          end if
      end if
  end while

  $close(all\ positions,\ price = market)$
```

Figure 7.6: Daily sentiment strategy pseudocode

Buying and closing positions on the open price of the next time step based on the average sentiment values of the current time assumes a low time complexity in sentiment evaluation to be able to apply in the online fashion. This is a safe assumption in the case of the GloVe model since it is quite lightweight. For BERT, on the other hand, it is best to exclude the last 10 seconds of each time step from hour evaluation (or include them in the evaluation of the next), but since the average sentiment derives from thousands of tweets per hour I assumed that the difference is marginal.

7.3 Evaluation of the Trading Strategy

I developed 4 strategies in total, which will be evaluated in this section:

- GloVe - Hourly Sentiment Strategy: Uses the outstanding positive sentiment percentage of each hour calculated by the GloVe model, and compares it to the average outstanding positive sentiment percentage of the last 200 hours.
- GloVe - Daily Sentiment Strategy: Uses the outstanding positive sentiment percentage of each day calculated by the GloVe model, and compares it to the average outstanding positive sentiment percentage of the last 20 days.
- BERT - Hourly Sentiment Strategy: Uses the outstanding positive sentiment percentage of each hour calculated by the BERT model, and compares it to the average outstanding positive sentiment percentage of the last 200 hours.

- BERT - Daily Sentiment Strategy: Uses the outstanding positive sentiment percentage of each day calculated by the BERT model, and compares it to the average outstanding positive sentiment percentage of the last 20 days.

In order to evaluate the performance of the implemented strategies, we will perform several tests. All the results presented below are calculated under the assumption of 0% commission. First, we track the equity curves of the implemented strategies, which is simply a graphical representation of the change in the value of their portfolios. This is presented in Figure 7.7.

Starting with an initial value of 1 the strategy in most cases doubled the portfolio over the period of 2 years. However, a substantially different performance can be observed before and after the Bitcoin crash. This can be visualized better by plotting the equity curve for each strategy for the two calendar years 2017 and 2018 separately. In Figures 7.8, 7.9 we can see that each strategy benefits from the uptrend of 2017 but struggles during the bear market of 2018. However, GloVe strategies were still able to generate profit during that time which shows great resilience.

Next up are the returns series, which are presented in Figure 7.10, calculated both per hour and per day for sub-day strategies. Here we look for periods of extreme losses or great fluctuation which are considered an unwanted characteristic for a trading strategy. Indeed we can see increased returns in both directions between the end of 2017 and the first months of 2018. However to better visualize the profitability of our strategies we plot the histogram of the returns of each trade. This is displayed in Figure 7.11 for each timeframe separately. There we can see that in most cases the right tail of the distributions is heavier than the left one, which reveals a high probability of profits than losses. This is more evident for GloVe based strategies, which seem to outperform those based on BERT calculated signals.

To put this in numbers we can calculate the Value-at-Risk and Expected Shortfall of the daily returns for 95% and 99% confidence intervals. Value-at-Risk (VaR) is the most widely used loss-distribution-based risk measure in quantitative risk management. By fixing a large probability α, VaR indicates the level for which the probability that the loss L exceeds this level is less or equal than $1-\alpha$. In other words VaR is simply the α-quantile of loss L. Mathematically VaR is defined as follows:

$$VaR_a(L) = \inf\{l \in \mathbb{R} : \mathbb{P}[L > l] \leq 1 - \alpha\} = q_a(L)$$

However useful, VaR has limitations as it does not show the severity of losses and tail risk, in cases where with a very small probability extreme losses might occur. To address that the expected shortfall (ES) was introduced also known as average value-at-risk. Since ES at confidence level α is essentially the expectation of losses L that exceed the VaR at confidence level α, and is mathematically described as follows:

$$ES_a(L) = \frac{1}{1-\alpha} \int_a^1 VaR_u(L)\, du = \frac{1}{1-\alpha} \int_a^1 q_u(L)\, du$$

Another popular metric for evaluating portfolios returns is the Sharpe Ratio, a risk-reward ratio, which compares the expected excess return generated by the strategy to the corresponding standard deviations:

$$Sharpe\ ratio = \frac{E[R - R^f]}{Std[R - R^f]}$$

R: annual return of the asset
R^f: annual risk-free rate

This performance measure depends on the time horizon over which it is measured. For example, the same strategy does not have the same Sharpe ratio if measured over a day as in this case or a year. However, if we assume returns to be independent and identically distributed we can calculate the annualized Sharpe Ratio from daily (assuming 252 trading days) returns as seen below:

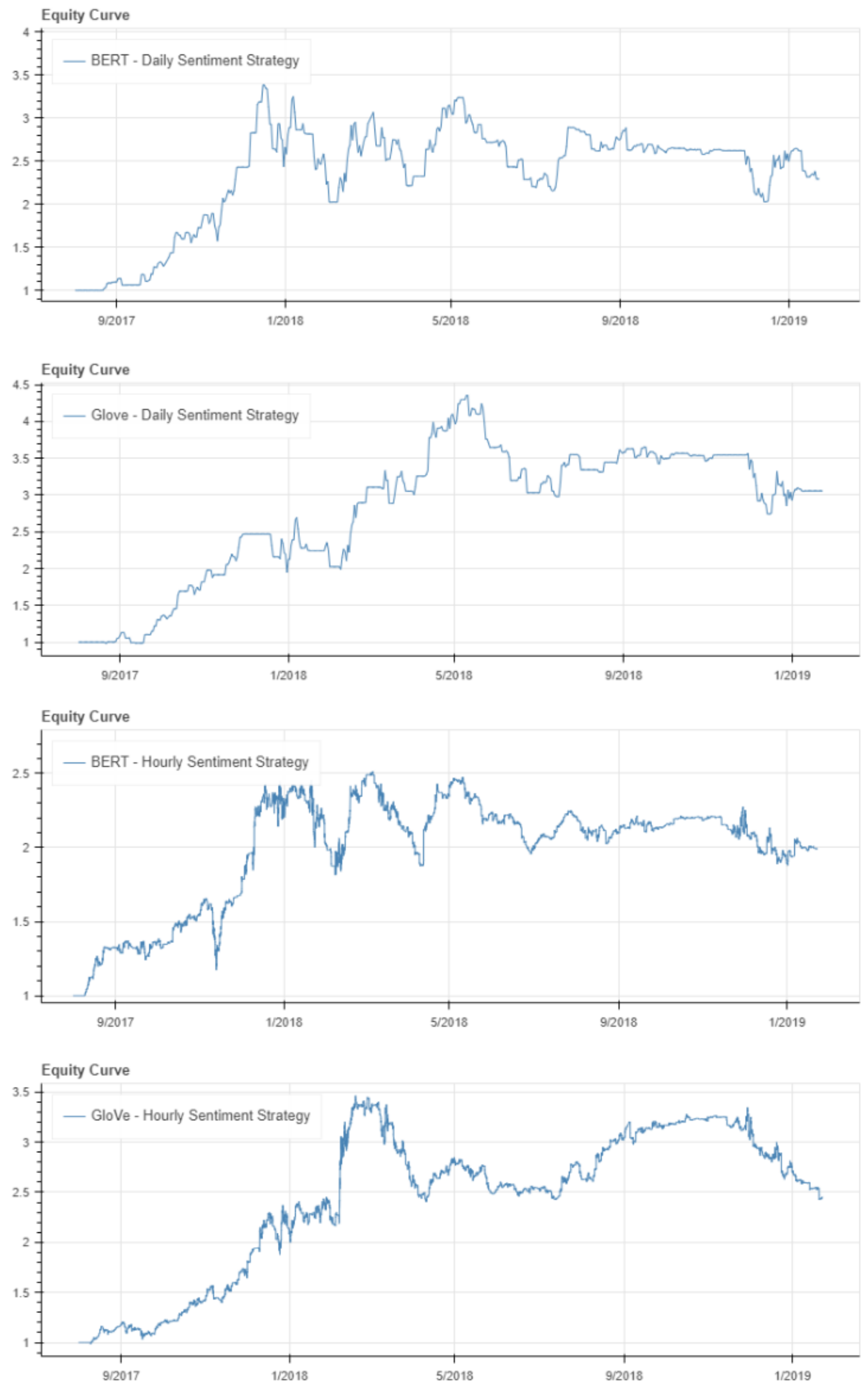


Figure 7.7: Equity curves of the four implemented strategies.

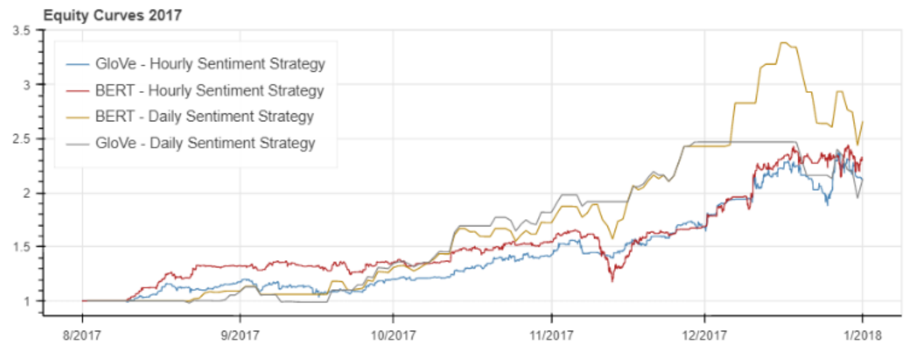


Figure 7.8: Equity curves for all strategies for year 2017

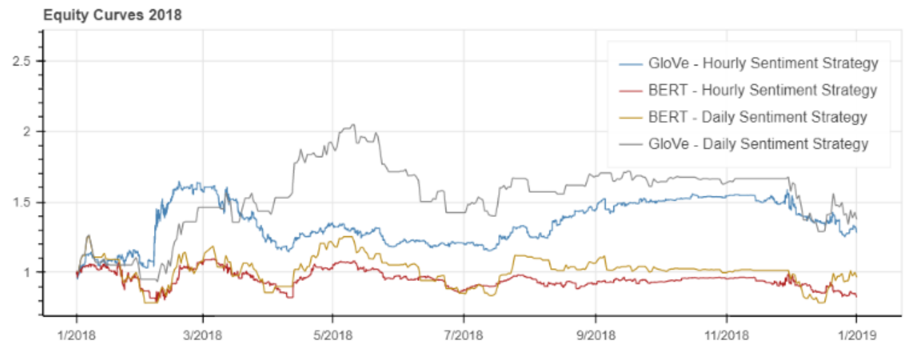


Figure 7.9: Equity curves for all strategies for year 2018

$$SR_{yearly} = \frac{E[\sum_{t=1}^{252} R_{daily} - R^f_{daily}]}{\sqrt{Var[\sum_{t=1}^{252} R_{daily} - R^f_{daily}]}} = \frac{252E[R_{daily} - R^f_{daily}]}{\sqrt{252Var[R_{daily} - R^f_{daily}]}} = SR_{daily}\sqrt{252}$$

R: daily return of the asset
R^f: daily risk-free rate
SR_{daily}: daily Sharpe ratio

While this is true for stocks, in the case of Bitcoin, however, which is traded 365 days per year we can state:

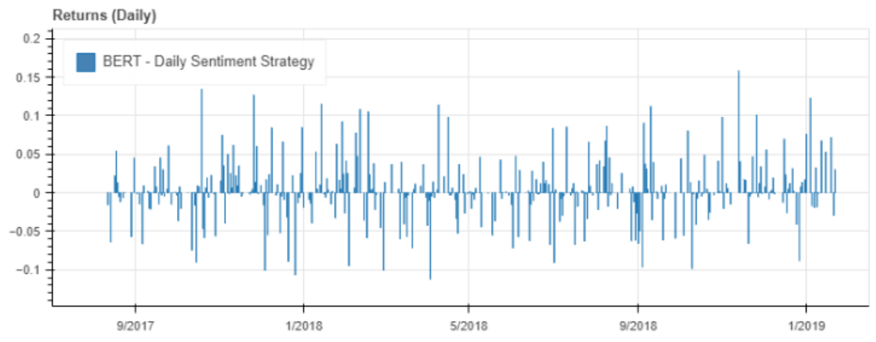
$$SR_{yearly} = SR_{daily}\sqrt{365}$$

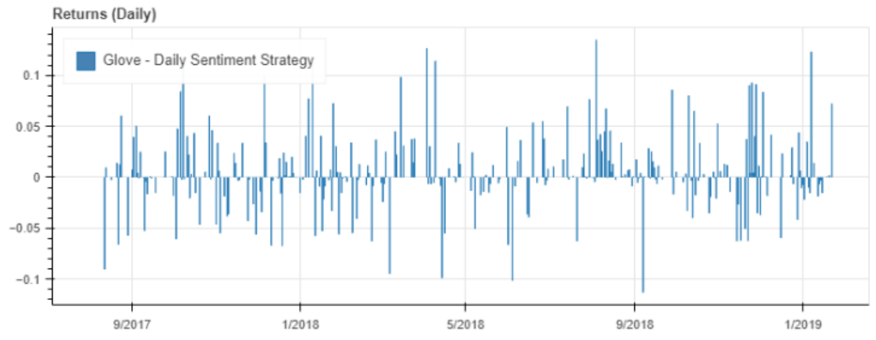
While the Sharpe ratio penalizes for all types of volatility in return series, the Sortino Ratio takes only into account the downside deviation for making calculations, as this is what's devastating for any trading strategy. Sortino ratio is defined as follows:

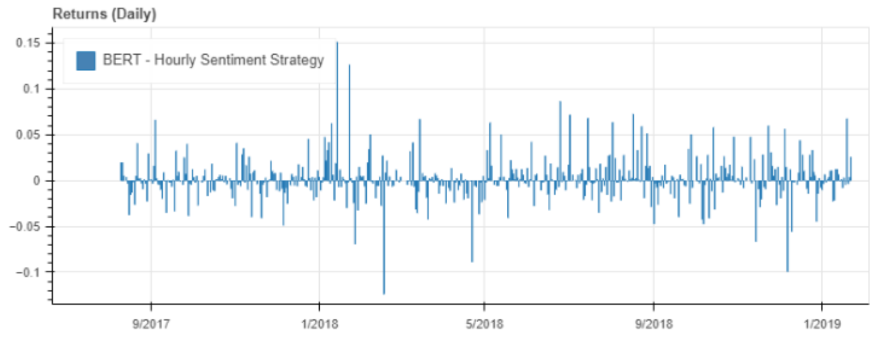
$$Sortino\ ratio = \frac{E[R - R^f]}{Std_d[R - R^f]} = \frac{E[R - R^f]}{Std[R - R^f|R < 0]}$$

R: annual return of the strategy
R^f: annual risk-free rate

Another important characteristic of strategies is their performance relative to their past maximum, known as the high-water mark:







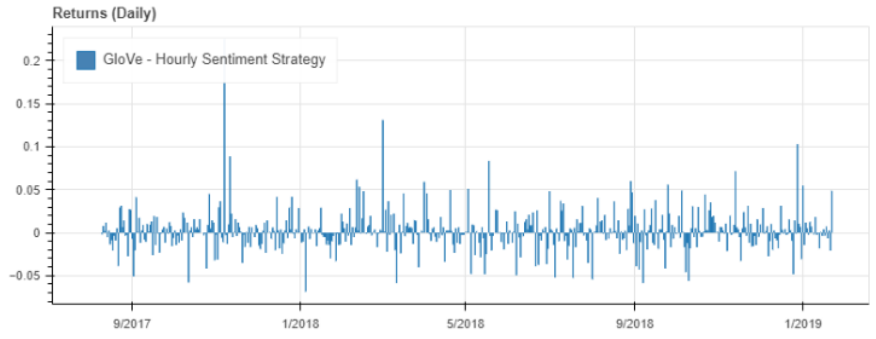


Figure 7.10: Daily returns of the four implemented strategies.

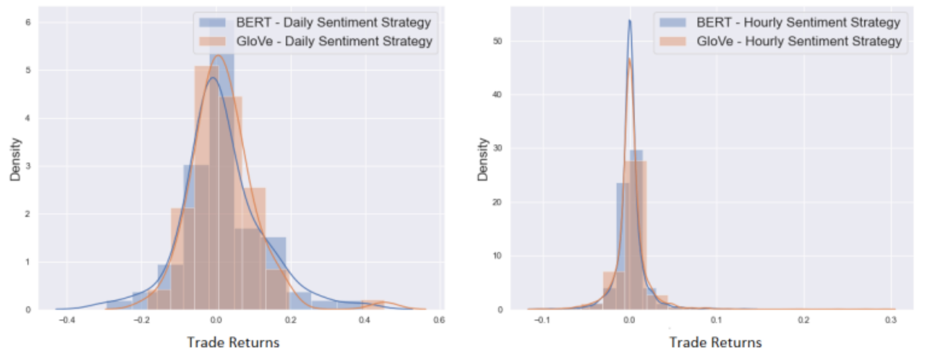


Figure 7.11: Trade returns distribution for strategies of daily (left) and hourly (right) timeframes.

$$HWM_t = \max_{s \leq t} P_s$$

The decline from the HWM is called the relative drawdown. This is an important indicator as it describes the conditionality amongst the returns increments. Investors often withdraw their money when a hedge fund drops too far below the high-water mark, so values closer to zero are an indication of a better strategy. The relative drawdown at time t is described as follows:

$$DD_t = \frac{HWM_t - P_t}{HWM_t}$$

P_t: Price at time t

And the maximum of all past relative drawdown values is called the maximum drawdown, which looks for the greatest downward movement from a high point to a low point before a new peak is achieved. Maximum drawdown is defined below:

$$MDD_t = \max_{s \leq t} D_s$$

We can visualize High-Water Mark, Relative Drawdown, and Maximum Drawdown on each equity curve as presented in Figures 7.12 - 7.15. In each case we see the maximum drawdown occurring during the bear market of 2018. Also while the relative drawdown seems to be high for the strategies, if we into account the severity of the decrease in Bitcoin's value during the crash, the strategies seem to hold up decently.

The last ratio we will look at is the Calmar Ratio, which incorporates the maximum drawdown into its calculation. Specifically, the Calmar ratio measures the annual expected return of the strategy over its maximum drawdown and is widely used as a measure of performance of investment funds.

$$Calmar\ ratio = \frac{R}{MDD}$$

R: annual return of the strategy
MDD: maximum drawdown of the strategy

Similar to the other risk-reward ratios, high Calmar ratio values are considered to be better. As hinted in the previous section, I tried to optimize the parameters of each strategy by adjusting their buy threshold and maximum allowed position number. For that, I implemented strategies with several different combinations of these values and evaluated them on their Calmar ratio. In Figures 7.16 and 7.17 the heatmaps of this evaluation for each signal are presented. On a first look, the fact that the Calmar ratio changes gradually with the changes in these parameters is an

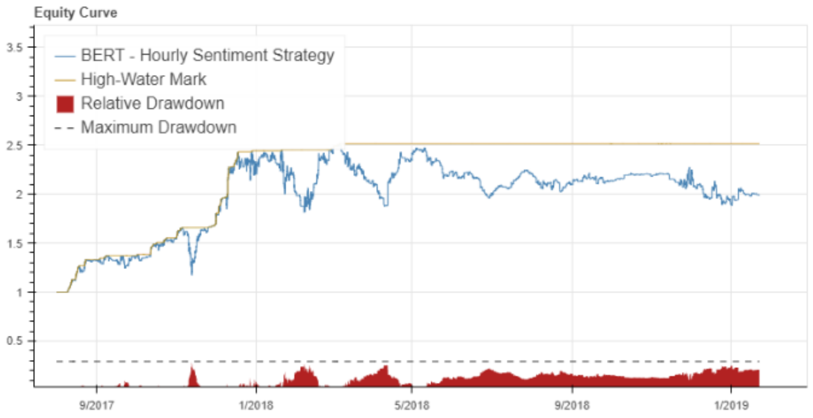


Figure 7.12: High-Water Mark and Drawdown of BERT - Hourly Sentiment Strategy

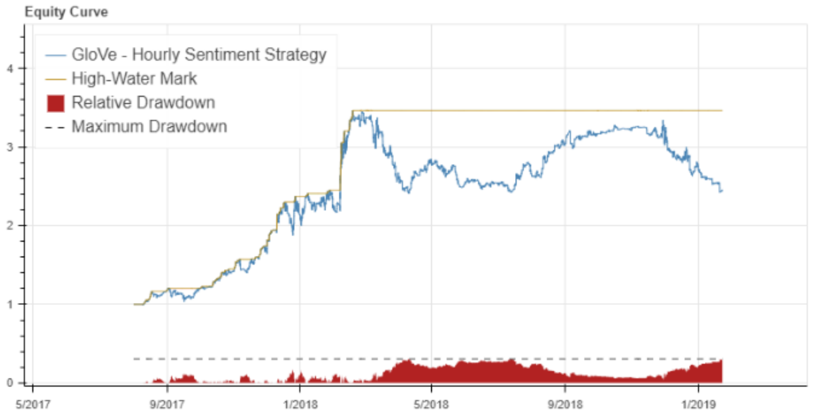


Figure 7.13: High-Water Mark and Drawdown of GloVe - Hourly Sentiment Strategy

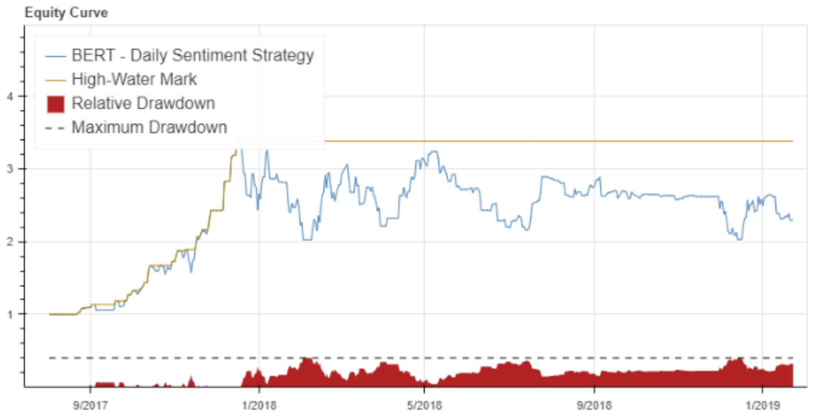


Figure 7.14: High-Water Mark and Drawdown of BERT - Daily Sentiment Strategy

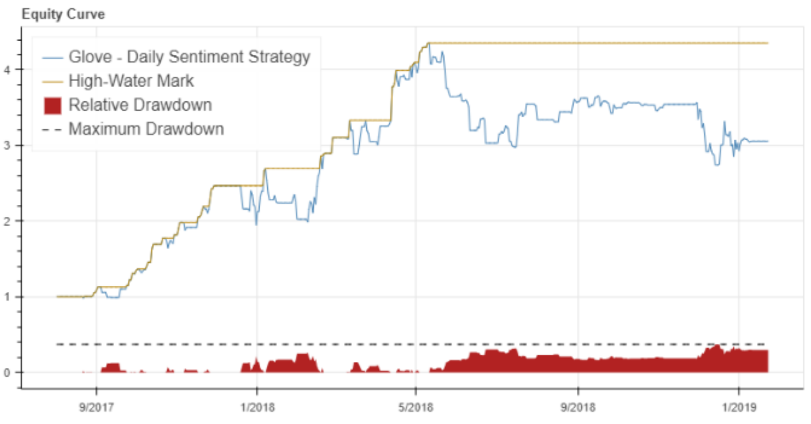


Figure 7.15: High-Water Mark and Drawdown of GloVe - Daily Sentiment Strategy

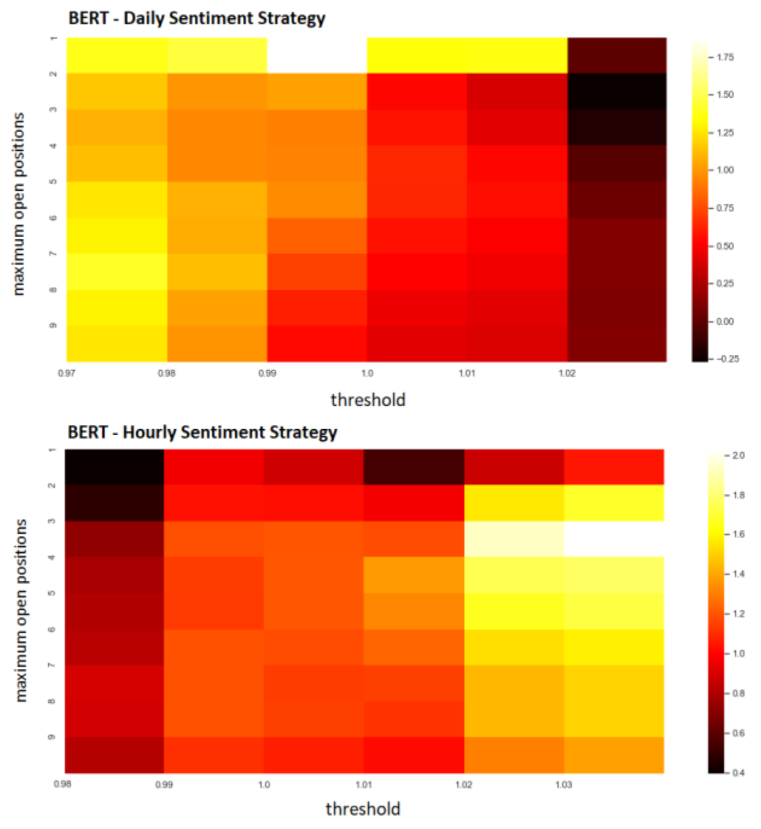


Figure 7.16: Calmar ratio for different threshold and maximum open positions combinations for BERT signal Strategies

encouraging sign for the validity of our strategies.

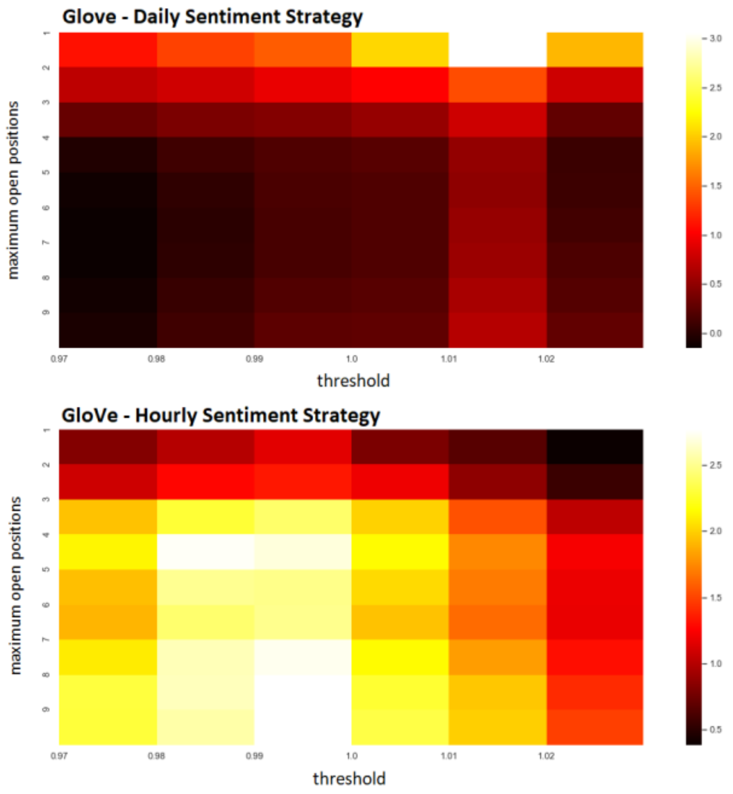


Figure 7.17: Calmar ratio for different threshold and maximum open positions combinations for GloVe signal Strategies

As we see the best BERT-based strategies have a Calmar ratio close to 2.0 which is considered good. The optimal BERT - Daily Sentiment Strategy invests the entire capital at once when the daily sentiment is higher than the 99% of its 20-day moving average and exits otherwise. The BERT - Hourly Sentiment Strategy allows for 3 open positions at maximum each day, with a size equal to 1/3 of portfolio value each and the optimal threshold 3% above the 200-hour average sentiment.

The optimal GloVe strategies perform even better, with the GloVe - Daily Sentiment Strategy scoring a Calmar ratio of value 3.0 which is exceptional. This strategy also invests the entire capital when the daily sentiment exceeds its 20-day moving average by at least 1%. Lastly, the optimal GloVe - Hourly Sentiment Strategy with an equally high Calmar ratio allows for a maximum of 4 concurrent open buy positions of size equal to 1/4 of portfolio value each, while the optimal threshold is 2% below the 200-hour average sentiment.

The last metric I will check for the implemented strategies is their Capital Usage Ratio. This metric shows what percent of our total portfolio value is being traded each day on average. Higher values of the ratio are considered to be better since it is generally preferred to have most of our portfolio invested at all times. For our strategies, however, this is absolutely true. Since we only open long positions on Bitcoin, it is generally a better strategy to avoid investing during periods of falling prices, like most of the year 2018.

All of the aforementioned measures are presented in Table 7.2 for the four implemented strategies. From these results, we can understand that the two strategies based on the GloVe produced signal perform better than those on BERT. The Sharpe Ratio is a good indication of that, which with a value of around 1.6 makes the two GloVe strategies tradeable. Sortino ratio of values close to 2 is considered good, while the value of 2.54 in the case of GloVe based hourly strategy is great. The maximum drawdown (MDD) is higher than we would like it to be, but given that Bitcoin price experienced a downfall of over 80% during the period of testing, it is acceptable.

Metrics	**BERT Hourly**	**GloVe Hourly**	**BERT Daily**	**GloVe Daily**
VaR 95	0.031	0.032	0.059	0.046
ES 95	0.048	0.045	0.078	0.066
VaR 99	0.054	0.053	0.096	0.067
ES 99	0.083	0.058	0.102	0.094
Sharpe Ratio	1.287	1.556	1.190	1.642
Sortino Ratio	1.806	2.542	1.442	1.845
Calmar Ratio	2.028	2.741	1.877	3.059
MDD	-26.93%	-30.05%	-40.22%	-37%
Capital Usage	22.489%	26.053%	15.277%	14.682%
Daily Returns				
mean	0.001578	0.00191	0.00217	0.002468
std	0.023005	0.02311	0.034386	0.028397
min	-0.12357	-0.068269	-0.112211	-0.112713
max	0.150933	0.225005	0.158404	0.134674
Total Returns	98.95%	145.41%	129.34%	205.66%

Table 7.1: Summary of metrics for the four implemented strategies

The last evaluation technique of our trading strategies we are going showcase in this section uses Monte Carlo simulation. The incentive is to look at the potential evolution of the equity curves over time from their daily returns, on the assumption the latter follow a normal distribution. Having the mean and volatility of daily returns for each strategy we can simulate thousands of possible equity curves with the same parameters on their daily returns. For these parameters, we can use their values estimated during the strategy testing by assuming past mean returns and volatility levels will continue into the future. After simulating 10,000 equity curves for each of the implemented strategies, I plot the distribution of the simulated portfolios final values after 540 days (since it is the original strategies testing period) as shown in Figure 7.18.

The distributions show that all strategies are profitable on average, while the weight of their right tails indicates that they can reach extremely high returns with a relatively high probability. In Table 7.2 we can see that the expected final portfolio value of each strategy for the period of testing is over 2 units, starting with the unit portfolio. Especially for the daily strategies, this number exceeds the 200% in total returns.

Metrics	**BERT Hourly**	**GloVe Hourly**	**BERT Daily**	**GloVe Daily**
Average Final Portfolio Value	2.32	2.81	3.23	3.77
5% quantile	0.83	1.01	0.64	1.02
95% quantile	4.90	5.93	8.73	9.07

Table 7.2: Summary of metrics for the Monte Carlo simulated distributions.

7.4 Comparison with Benchmarks

In this section, I will compare the four implemented strategies with the most widely used benchmarks. The first and most standard evaluation is the comparison of our portfolio value with that of the S&P 500 during the same period. This will tell us if our strategy is better than simply undertaking market risk and buying the market. The results are displayed in Figure 7.19 on a

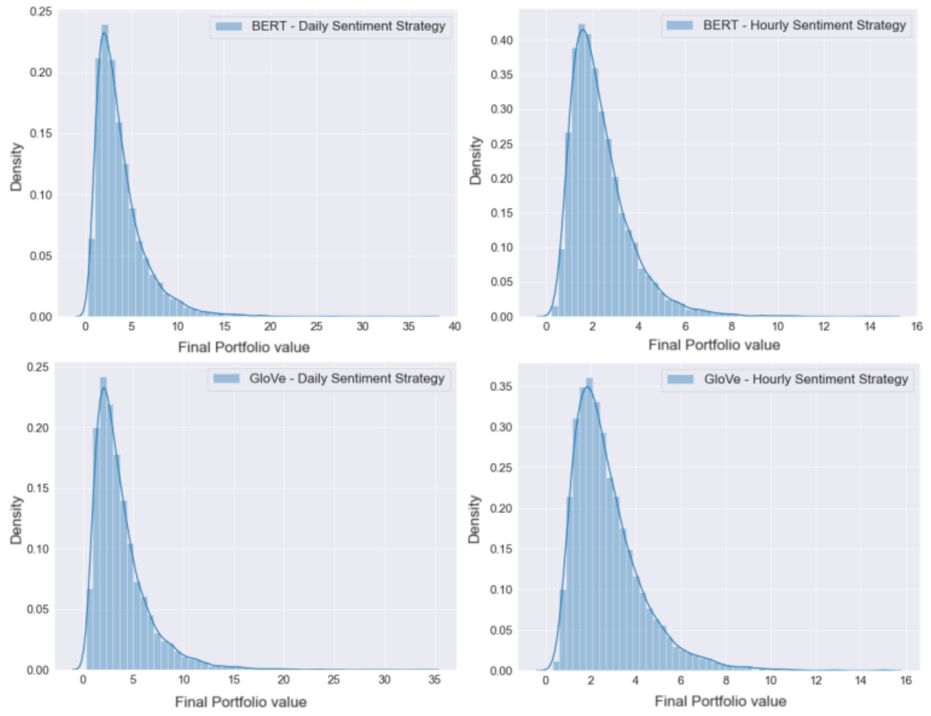


Figure 7.18: Final portfolio value distribution of 10,000 simulated equity curves for each implemented strategy.

logarithmic scale to allow for better visualization. The curves reveal the big difference in riskiness of our strategy compared to the market index, with both the upside and the downside being significantly amplified. However, starting with a unit portfolio our strategies were able to produce substantially higher profits than the buy and hold on S&P 500. The results however change if we introduce some commission costs to our strategies, as I will show in the appendix.

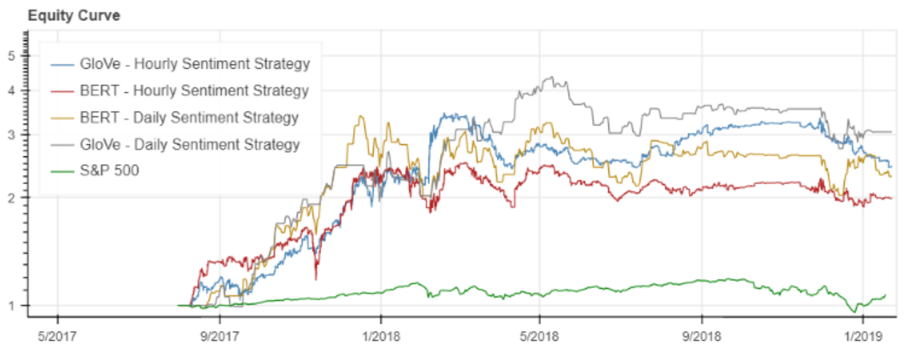


Figure 7.19: Comparison of our Trading Strategies with the Buy and Hold on S&P 500 (Logarithmic Scale).

The contrary is true, however, if we compare our strategies with the Buy and Hold strategy in Bitcoin for the same period as shown in Figure 7.20. In some cases, it might be simply better to buy and hold a position on the underlying asset than trading on it, which is also cheaper in terms of trading costs. In this case, however, while our strategies weren't able to benefit in the fullest from the bull market in Bitcoin prior to the crash, they still performed significantly better than Bitcoin during the bear market of 2018.

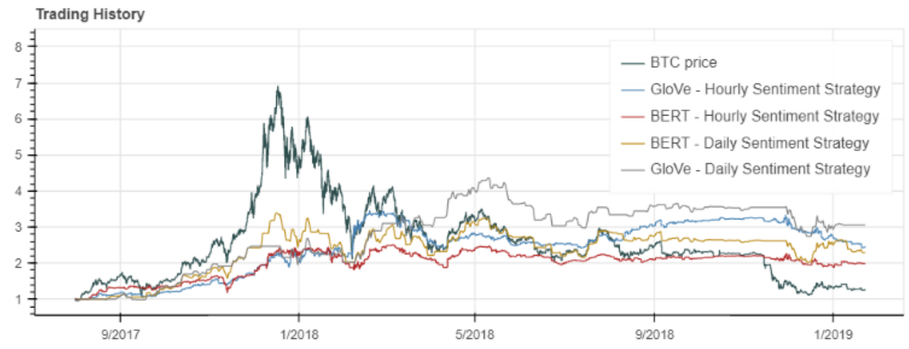


Figure 7.20: Comparison of our Trading Strategies with the Buy and Hold on Bitcoin.

Chapter 8

Conclusion

In this project, I examined the usefulness of social media sentiment in predicting future Bitcoin price movements. For that, some of the more advanced Natural Language Processing Models were utilized and used to classify 2 million tweets regarding Bitcoin as either positive or negative. The calculated sentiment values were incorporated into a Trading Strategy is a signal based on which trades on the Bitcoin were made. I constructed four strategies in total distinguished by the time-frame on which they traded (daily or hourly) and by the signal they used for decision-making (signal calculated by the BERT model or signal calculated by the GloVe model). The strategies were simple buy-only strategies that would buy Bitcoin when they observed positive sentiment values above a rolling average. After running the strategies for the period 2017-2019 most of the strategies managed to double their initial portfolio in value while one of them tripled its value. Furthermore, each strategy was able to benefit from the bull market of 2017 in Bitcoin while they sustained the 2018 period decently where Bitcoin lost 80% of its value. This speaks in favor of our initial hypothesis that Twitter sentiment can be employed for predicting future Bitcoin prices. However, in order to solidify this idea some more testing is required possibly on more recent data.

The analysis conducted in this project involved the parsing and analysis of textual data, their spam filtering, and transformation, the development and training of neural networks, and the adaptation of state of art Language model to the needs of our classification task. The choice of Bitcoin as the underlying asset was made as it is a popular topic amongst Twitter users. I leave it as future work to apply the same models and ideas for tweets related to stocks and test the predictability of the calculated signal on the stock market. While evaluating the implemented sentiment classifiers it was shown that each model relies on different language characteristics to make its classification. Thus another step forward could be the combination of sentiment values calculated with different language models to enhance the reliability of the final signal. The final step of this analysis will be to construct a trading bot, that connected to Twitter's API will parse data, classify them construct the final signal in an online fashion, and depending on the calculated signal, will trade the underlying asset in real-time.

Appendix A

Appendix

A.1 Other methods for calculating the trading signal

When constructing the trading signal, more complex method can be employed that also utilize other information from our tweets data set . One such methods involves taking into account the sentiment history of the user that is posting each tweet. This can be done by holding ongoing average sentiment scores for each user (from their past tweets) and using those to weigh each tweet. The idea is that a positive message (negative resp.) has much more gravity when coming from a pessimistic user (optimistic resp.) on average rather than from an optimistic (pessimistic resp.) one. When implementing this scheme, all users are assumed as neutral at the start (initial sentiment score 0.5) while the "weighting" is simply

$$new\ sentiment–past\ average\ sentiment$$

which broadens the sentiment value range to (-1,1).

Putting the above example in numbers:

	Average Sentiment	New tweet sentiment	Final sentiment (weighted)
User 1	0.8	0.1	0.1 – 0.8 = - 0.7
User 2	0.3	0.1	0.1 – 0.3 = - 0.2

The final tweet sentiment of User1 is much lower than that of User2, even though they have posted the same tweet, due to their past scores. The whole process of calibrating the weighted scores is presented in pseudocode in Figue A.1.

With this technique I was able to avoid some sudden spikes for the final signal that occurred with the simpler methods. However since it involves storing and accessing a table of every user in each evaluation, it slows down the classification process significantly. Thus, I went with the much simpler method of averaging per user per period which operates in a similar fashion.

A.2 Visualizing the trading history

In this section the visualization of the two daily sentiment strategies is presented by pinpointing the opening and closing of positions on the Bitcoin price chart. The Figures A.2 and A.3 summarize the entire trading history of the strategy. In these figures we can see reduced trading to occur during the major drops in Bitcoin's price which is a positive sign for the validity of our strategies.

A.3 Introducing trading costs

As explained in the Chapter 7 comparing our strategies with the Buy and Hold in Bitcoin and S&P 500 doesn't reveal the true differences of the approaches. Here I backtested the GloVe daily strategy for the same period but, this time I included fixed comissions of 1% of the position size. Then I repeated the comparison with the S&P 500 and Bitcoin Buy and Hold strategies and the

Algorithm 1 User sentiment weighted scores calculation

```
**Initialize:** $U \leftarrow \{\}$                      ▷ dictionary for users' past scores
**get** $u, s$                                   ▷ username and sentiment score of tweet
**while** $u \neq None$ and $s \neq None$ **do**
//Update
    **if** $u \notin U$ **then**
        $s_n \leftarrow s$
        $t_n \leftarrow 1$
        $U[u] \leftarrow (s_n, t_n)$
    **else**
        $(s_p, t_p) \leftarrow U[u]$
        $s_n \leftarrow s_p + s$
        $t_n \leftarrow t_p + 1$
        $U[u] \leftarrow (s_n, t_n)$
    **end if**
//Calculation
    $(s_n, t_n) \leftarrow U[u]$
    $s_{weighted} \leftarrow s - \frac{s_n}{t_n}$
    **yield** $s_{weighted}$
    **get** $u, s$
**end while**
```

Figure A.1: User sentiment weighted scores implementation psuedocode

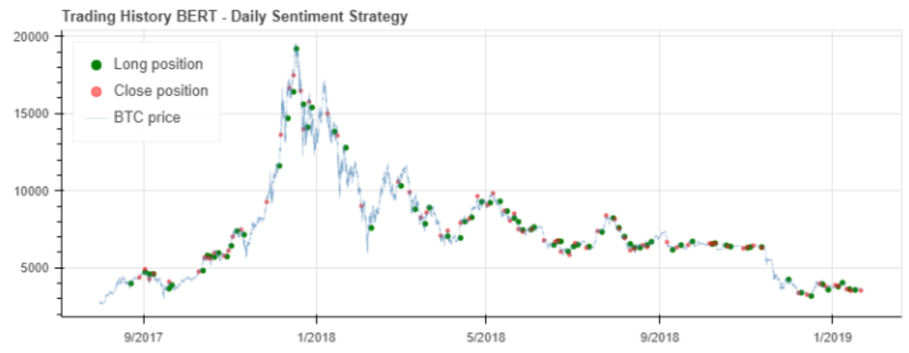


Figure A.2: Trading History of BERT - Daily Sentiment Strategy visualized.

results are presented in Figures A.4, A.5. The results are significantly worse, but the strategies still managed to end up more profitable than the two benchmark strategies.

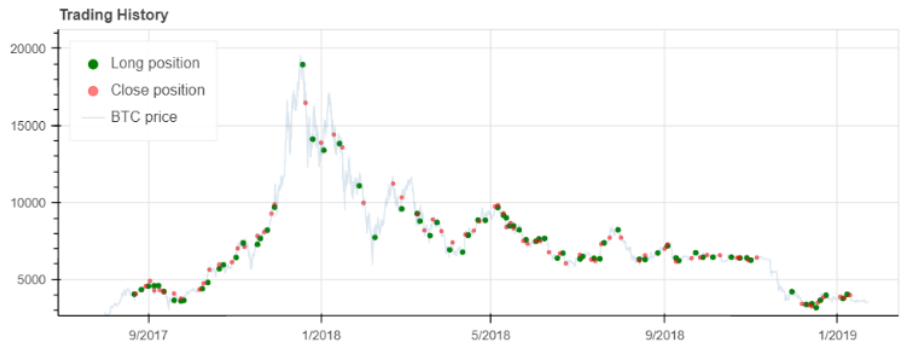


Figure A.3: Trading History of Glove - Daily Sentiment Strategy visualized.

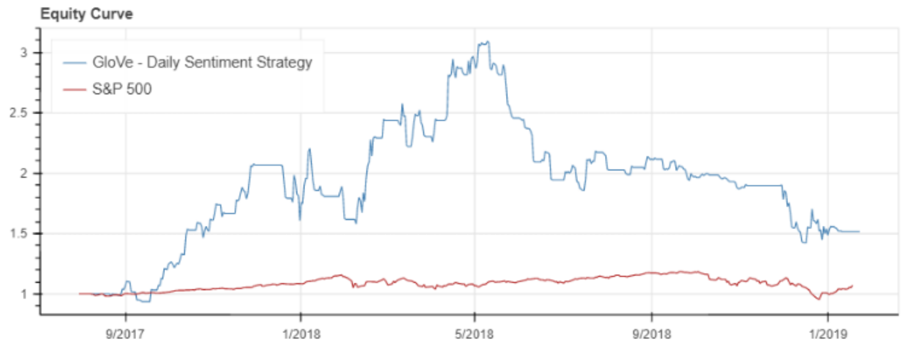


Figure A.4: Comparison of Glove - Daily Sentiment Strategy with 1% comissions with S&P 500 Buy and Hold strategy.

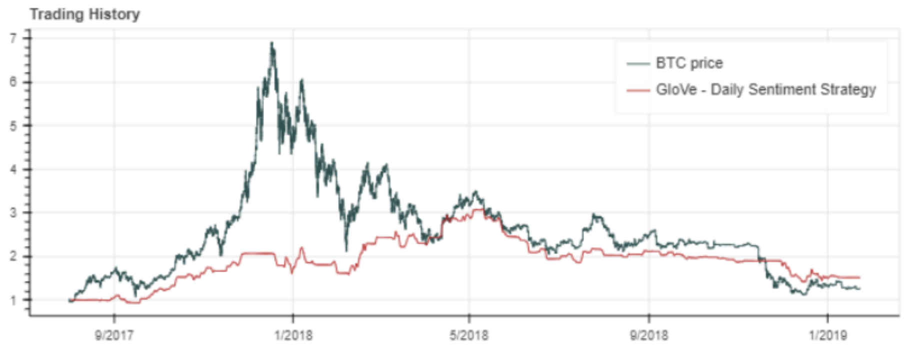


Figure A.5: Comparison of Glove - Daily Sentiment Strategy with 1% comissions with Bitcoin Buy and Hold strategy.

Bibliography

[1] Jim D.says: et al. *How Many People Use Social Media in 2021? (65 Statistics)*. Available online at: https://backlinko.com/social-media-users, last accessed on 09.06.2021. Sept. 2021.

[2] Lasse Heje Pedersen. "Game on: Social networks and markets". In: *Available at SSRN 3794616* (2021).

[3] H. Sul, A. Dennis, and L. Yuan. "Trading on Twitter: Using Social Media Sentiment to Predict Stock Returns". In: *Decis. Sci.* 48 (2017), pp. 454–488.

[4] Johan Bollen, Huina Mao, and Xiaojun Zeng. "Twitter mood predicts the stock market". In: *Journal of Computational Science* 2.1 (2011), pp. 1–8. ISSN: 1877-7503. DOI: https://doi.org/10.1016/j.jocs.2010.12.007. URL: https://www.sciencedirect.com/science/article/pii/S187775031100007X.

[5] Jason Brownlee. *Why One-Hot Encode Data in Machine Learning?* Available online at: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/, last accessed on 09.06.2021. June 2020.

[6] Jason Brownlee. *A Gentle Introduction to the Bag-of-Words Model*. Available online at: https://machinelearningmastery.com/gentle-introduction-bag-words-model/, last accessed on 09.06.2021. Aug. 2019.

[7] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: Jan. 2013, pp. 1–9.

[8] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[9] *Implementing Deep Learning Methods and Feature Engineering for Text Data: The Continuous Bag of Words (CBOW)*. Available online at: https://www.kdnuggets.com/2018/04/implementing-deep-learning-methods-feature-engineering-text-data-cbow.html, last accessed on 09.06.2021.

[10] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[11] Matthew E. Peters et al. "Semi-supervised sequence tagging with bidirectional language models". In: *CoRR* abs/1705.00108 (2017). arXiv: 1705.00108. URL: http://arxiv.org/abs/1705.00108.

[12] Bryan McCann et al. "Learned in Translation: Contextualized Word Vectors". In: *CoRR* abs/1708.00107 (2017). arXiv: 1708.00107. URL: http://arxiv.org/abs/1708.00107.

[13] Matthew E. Peters et al. "Deep contextualized word representations". In: *CoRR* abs/1802.05365 (2018). arXiv: 1802.05365. URL: http://arxiv.org/abs/1802.05365.

[14] Jason Brownlee. *How to Develop a Bidirectional LSTM For Sequence Classification in Python with Keras*. Available online at: https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/,last accessed on 09.06.2021. Jan. 2021.

[15] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 3104–3112.

[16] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *CoRR* abs/1409.0473 (2015).

[17] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *CoRR* abs/1508.04025 (2015). arXiv: `1508.04025`. URL: `http://arxiv.org/abs/1508.04025`.

[18] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: `1706.03762`. URL: `http://arxiv.org/abs/1706.03762`.

[19] Jay Alammar. *The Illustrated Transformer*. Available online at: `https://jalammar.github.io/illustrated-transformer/`, last accessed on 09.06.2021.

[20] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: `1810.04805`. URL: `http://arxiv.org/abs/1810.04805`.

[21] Alec Radford and Karthik Narasimhan. "Improving Language Understanding by Generative Pre-Training". In: 2018.

[22] Andrew M. Dai and Quoc V. Le. "Semi-supervised Sequence Learning". In: (2015). arXiv: `1511.01432 [cs.LG]`.

[23] Jay Alammar. *The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)*. Available online at: `https://jalammar.github.io/illustrated-bert/`, last accessed on 09.06.2021.

[24] M. Chew et al. "Using Natural Language Processing Techniques for Stock Return Predictions". In: *Econometric Modeling: Capital Markets - Forecasting eJournal* (2017).

[25] Xiao Ding et al. "Deep learning for event-driven stock prediction". In: *Twenty-fourth international joint conference on artificial intelligence*. 2015.

[26] Alec Go, Richa Bhayani, and Lei Huang. "Twitter Sentiment Classification using Distant Supervision". In: ().

[27] Efthymios Kouloumpis, Theresa Wilson, and Johanna D. Moore. "Twitter Sentiment Analysis: The Good the Bad and the OMG!" In: *ICWSM*. 2011.

[28] Federico Neri et al. "Sentiment Analysis on Social Media". In: *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. 2012, pp. 919–926. DOI: `10.1109/ASONAM.2012.164`.

[29] Chong Oh and Olivia Sheng. "Investigating Predictive Power of Stock Micro Blog Sentiment in Forecasting Future Stock Price Directional Movement." In: vol. 4. Jan. 2011.

[30] Jasmina Smailovic et al. "Stream-based active learning for sentiment analysis in the financial domain". In: *Inf. Sci.* 285 (2014), pp. 181–203.

[31] Hong Kee Sul, Alan R Dennis, and Lingyao Yuan. "Trading on twitter: Using social media sentiment to predict stock returns". In: *Decision Sciences* 48.3 (2017), pp. 454–488.

[32] Hai Liang and King-wa Fu. "Testing Propositions Derived from Twitter Studies: Generalization and Replication in Computational Social Science". In: *PLoS ONE* 10 (Aug. 2015), e0134270. DOI: `10.1371/journal.pone.0134270`.

[33] Gordon V Cormack. "Email spam filtering: A systematic review". In: (2008).

[34] Ioannis Kanaris et al. "Words versus character n-grams for anti-spam filtering". In: *International Journal on Artificial Intelligence Tools* 16.06 (2007), pp. 1047–1067.

[35] Thiago S Guzella and Walmir M Caminhas. "A review of machine learning approaches to spam filtering". In: *Expert Systems with Applications* 36.7 (2009), pp. 10206–10222.

[36] Chao Chen et al. "6 million spam tweets: A large ground truth for timely Twitter spam detection". In: *2015 IEEE International Conference on Communications (ICC)*. 2015, pp. 7065–7070. DOI: `10.1109/ICC.2015.7249453`.

[37] Trevor Hastie, Jerome Friedman, and Robert Tisbshirani. *The Elements of statistical learning: data mining, inference, and prediction*. Springer, 2017.

[38] Jason Brownlee. *A Gentle Introduction to Long Short-Term Memory Networks by the Experts*. Available online at: `https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/`, last accessed on 09.06.2021. July 2021.

[39] Sarang Narkhede. *Understanding AUC - ROC Curve*. Available online at: `https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5`, last accessed on 09.06.2021. June 2021.

[40] Paul van der Laken. *ROC, AUC, precision, and recall visually explained*. Available online at: `https://paulvanderlaken.com/2019/08/16/roc-auc-precision-and-recall-visually-explained/`, last accessed on 09.06.2021. May 2020.

TSOULIAS_KONSTANTINOS_02007404

GRADEMARK REPORT

FINAL GRADE

/0

GENERAL COMMENTS

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49

PAGE 50

PAGE 51

PAGE 52

PAGE 53

PAGE 54

PAGE 55

PAGE 56

PAGE 57

PAGE 58

PAGE 59

PAGE 60

PAGE 61

PAGE 62

PAGE 63

PAGE 64