Opinion Mining and Social Media Sentiment Analysis in the Prediction of Cryptocurrency Prices

School of Mathematical, Physical and Computational Sciences
Indivdual Project - CS3IP16

Student: Andrew Sotheran
Student Number: fr005432
Supervisor: Kenneth Boness
Word Count: Place Holder

Submission date: Place Holder

1 Abstract

The volatility of the stock markets is an aspect that is both hard to predict and to mitigate particularly when relating to the cryptocurrency market. Commodities such as cryptocurrencies are profoundly volatile and have attracted investors in an attempt to make quick profits on the market. These financial commodities are subject to the whim of public confidence and platforms such as Twitter and Facebook are most notably utilised to express opinions. Extrapolating sentiment from such platforms has been used to gain insight into topics across industries, thus applying it to crypto-market analysis could serve to show a relationship between public opinion and market change.

This project looks into public perception of the cryptomarket, by analysing Bitcoin-related tweets per hour for sentiment changes that could indicate a correlation to market fluctuations in the near future. This is achieved by training a recurrent neural network on the severity changes of historical sentiment and price over the past year every hour. The predictions are then shifted forward in time by 1 hour to indicate the corresponding Bitcoin price interval.

2 Acknowledgements

I would like to express my gratitude to Dr. Kenneth Boness for his continued support and guidance throughout this project.

Secondly, I want to express gratitude to PhD. Jason Brownlee, of Machine Learning Mastery for having clear and thorough explanations of machine learning concepts and metrics.

I would also like to thank my family for their support during the development of this project.

3 Glossary

Bull(ish)/Bear(ish) Markets - Relates to a trend of the market price increasing and decreasing respectively

Highs/Lows - The highest and lowest trading price of a giving period

Fiat Currency - A currency without intrinsic value that has been established as money

BTC - Bitcoin's stock symbol

Twitter - Online social media platform, which allows users to post information or express opinions through messages called "Tweets"

Tweets - The name given for messages posted on the Twitter platform, which are restricted to 280 characters.

Hashtag - Is a keyword or phrase used to describe a topic and allows the tweets to be categorised.

Fomo (Fear of Missing Out) - Is used to describe buying behaviour when stocks are moving suddenly and more buyers appear to enter all of a sudden.

Shorting - Or short sale, is the sale of an asset that the investor buys shares and immediately sells them, hoping to make a profit from buying later at a lower price.

Doubling Down - Is to take further risk on a stock by doubling effort/investment in a hope and attempt to raise the price

RNN - Recurrent Neural Network

LSTM - Long-Short Term Memory Neural Network

Contents

4 Introduction

The premise of this project is to investigate whether the sentiment expressed in social media has a correlation to the prices of cryptocurrencies and how this could be used to predict future changes in the price.

The chosen cryptocurrency that will be of focus for this project will be the currency Bitcoin (BTC), due to having the largest community and backing and has been known to lead other fiat currencies. Bitcoin is seen as one, if not the first cryptocurrency to bring a broader following to the peer-to-peer token transaction scene since 2009. Although it was not the first token to utilise blockchain technology, it allowed investors to openly trade a public cryptocurrency which provided pseudonymous means of transferring funds through the internet. Thus it has been around longer than most of the other fiat currencies and is the most popular crypto-token due to it's more extensive community base.

Most financial commodities are subject to the whim of public confidence and are the core of its base value. A platform that is frequently used for the public to convey their opinions on a commodity is that of Twitter which provides arguably biased information and opinions. Whether the opinions present a basis in facts or not, they are usually taken at face value and can influence the public opinion of given topics. As Bitcoin has been around since 2009 the opinions and information on the commodity are prevalent through the platform. In the paper *Sentiment Analysis of Twitter Data for Predicting Stock Market Movements* by *Majhi et al.* [1] 2.5 million tweets on Microsoft were extracted from Twitter, sentiment analysis and logistical regression performed on the data yielded 69.01% accuracy for a 3-day period on the increase/decrease in stock price. These results showed a "*good correlation between stock market movements and the sentiments of the public expressed in Twitter*".

The background of this project is in response to the volatility of the cryptocurrency market, which can fluctuate at a moments notice and can be seen to be social media driven. The history of the price of Bitcoin and what was being discussed on the currency around it's most volatile period to-date, Nov-2017 to Feb-2018, shows a strong bullish trend which saw Bitcoin reach a $19,500 high in mid-Dec. While social media, such as Twitter, during that period was had an extremely positive outlook on the cryptocurrency. The trend was short lived and saw the market crash only a month later, with only a couple of sell-offs, expected for the holidays rush, accompanied by negative outlooks posted on social media turned the market against itself which saw the longest bearish market in Bitcoin's history and is still trying to recover today.

Due to how volatile the crypto-market can be, there is a need to either mitigate or to anticipate where the markets are heading. As the crypto-market and Bitcoin are affected by socially constructed opinions, either through Twitter, news articles or other forms of media, there is a way to perform the latter, where the prices of Bitcoin could be predicted based on the sentiment gathered from social media outlets.

This project aims to create a tool that gathers tweets from Twitter, obtains the overall sentiment score of the given text while gathering historical price data for the period gathering occurs. Features are then extracted from the gathered data and used in a neural network to ascertain whether the price of the currency can be predicted from the correlation between the sentiment and price history of the data.

This report will discuss the justifications for the project and the problems it will be attempting to resolve, the stakeholders that would benefit the most from this system and what this project will not attempt to accomplish. Similar tools will be critiqued and examined for their feature set and credibility in the literature review along with current sentiment analysers, algorithms, natural language processing techniques and neural networks in their respective topics and comparing their accuracy for this project purpose. The solution approach will discuss the decisions and reasoning behind choosing the techniques and tools used for this project and will outline the requirements for this project. Implementation of the chosen techniques and tools, with the discussion of essential functions of the system will formulate the implementation section of this report with an in-detail explanation of the function's use and data flow of the system.

5 Problem Articulation

5.1 Problem Statement

The fundamental problems this project attempts to address are that of, an open-source system available to the public that aids in the analysis and prediction of BTC. The accuracy of open-source tools and technology when applied to the trading market scene and to identify whether there is a correlation between Twitter sentiment and BTC price fluctuation. While there are existing tools, only a few are available to the public and only provide basic functionality, while others are kept in-house of major corporations who invest in this problem domain.

The other issue presented here is that assuming perfect accuracy can be achieved is naive. As this project will only be using existing tools and technologies; thus, there are limitations to the accuracy of what can be obtained. One of that being the suitability of the tools, there are no open-source sentiment analysers for stock market prediction, thus finding a specifically trained analyser for the chosen domain in highly unlikely. In relation, finding the most suitable machine learning or neural network is equally important as this will determine the accuracy of the predictions. Due to being a regression problem, machine learning techniques and neural networks that focus around this and forecasting should be considered.

The accuracy and suitability of various machine learning methods and neural networks are a known issue in their respective domains. This investigation should be carried out to determine their suitability for their needed use in this project and will be detailed in the literature review.

This project will focus on the investigation of these technologies and tools to justify whether it is feasible to predict the price of BTC based on historical price and the sentiment gathered from Twitter. Limitations of the system and it's accuracy in predictions should be investigated and discussed to determine the implemented solution is the more suitable compared to other methods.

5.2 Stakeholders

The main stakeholders of this system would be those looking to invest in the cryptocurrency markets, in this projects regard, specifically into Bitcoin.

- Public Investors - These are investors from the general public. These investors can decide to either actively or passively invest in the markets but are essential for the general use of a given cryptocurrency. This type of investor would benefit the most from an open-source system such as this, as it will aim to provide a basis for decisions for buying or selling Bitcoin. Additionally, due to the lack of any open-source tools available, these stakeholders could be seen as being left in the dark when it comes to predicting the direction of Bitcoin where Businesses and Enterprises will have a one up, due to having an internal system for predictions.

- Speculators - These stakeholders can be both public and business, who aim for the chance of the possibility fast. They actively invest at points where a market is an impending rise in price and tend to sell after a market makes them a reasonable amount of money before it possibly drops. These stakeholders would benefit from such a system as it will provide a means to identify and predict short term gains in the Bitcoin market, and if taken into decisions could make a profit.

- Business Investors: These will be investors of a business who would be investing on the behalf of a company. A system such as that this project will provide may benefit such a stakeholder, but the information would be used as a collective with others to justify the investment. Additionally, this system may not benefit this stakeholder as the company they are investing for may have an equivalent or better system.

- Prospect Investors: These are new investors to the cryptomarket scene who are looking to get into the market and are generally looking for initial information of the market movement. This system will benefit such a stakeholder in their initial decisions in market investment, but not as much as a generally more active investor. This is due to the extent to which a new investor invests compared to a establish active investor.

- Developer - Andrew Sotheran: The developer responsible for this project by developing a solution that satisfies the problem and objective defined in the *Technical Specification*. As the sole developer of this project it should be ensured that the system is developed on time and the project runs smoothly.

- Project Supervisor - Kenneth Boness: Is the projects supervisor whom will oversee the development through weekly project meetings. Weekly feedback will be given on the progress and direction of development, and will offer advice to ensure the quality of the solution.

5.3 Project Motivation

The motivation behind the project stems from a range of points, from personal and public issues with the volatility if the crypto-market, and how losses specifically could be mitigated. The personal motivations behind the conceptualisation of this began two years ago during the crash of late 2017-2018, which saw new investors blindly jump into the trend that was buying cryptocurrencies. During this period of November to December 2017 saw Bitcoin's price reach \$20,000 from \$5,000, new public investors jumped on the chance to buy into the trend of possibly making quick profits and the fear of missing out (FOMO). In late December, a few holiday sell-offs occurred from business and big investors, this coupled with a few negative outlooks posted on social media by news outlets caused the market to implode causing investors to panic sell one after the other and posting negativity on social, thus causing more decline in the market. As a result, this caused personal monetary loss and distress as being a long-term investor.

Another motivation is that at the time of writing, there are no publically available systems that combine sentiment analysis with a historical price to forecast the price of Bitcoin or any other cryptocurrency. There are papers and a few code repositories that implement a similar concepts [2] - *Use of a Multi-layer Perceptron network for moving averages in Bitcoin price*, [3] - *Predicting Bitcoin price fluctuation with Twitter sentiment analysis*, [4] - *Predict Tomorrows Bitcoin (BTC) Price with Recurrent Neural Networks* but are not operational. A system such as [1] hosted on Coingecko, a popular cryptocurrency track site, provides a tool for basic sentiment analysis but doesn't give an evaluated indication of the direction of the market as a prediction. This leaves the public to the whim of volatility of the market without a means to know what the next, say an hour, could entail to possibly reduce losses if the market drops. Such systems are usually kept in-house of major corporations who invest significant time into tackling such a problem. Additionlly, this could be seen as a positive for major investors, as such a system could cause panic selling if public investors solely trusted such a system.

5.4 Technical Specification

This project will need to follow a specification to ensure that the quality and the problem statement is met. This section will outline what this project should include, what it will not consist of and will guide the development of this project.

General:

- To investigate into the use of lexicon-dictionary based sentiment analyser approach in for sentiment analysis and it's customisability for a given topic domain
- To create a system that can predict the next hour of Bitcoins price when given the price and sentiment for the past hour
- To investigate into natural language data pre-processing techniques and how these could be used to filter out unwanted data
- To investigate into the use of a neural network, specifically an LSTM for forecasting price data
- Ultimatly, to investigate into how the use of sentiment effects the prediction of price for the next hour

Natural Language pre-processing (Spam and language detection filtering)

- To produce a system that processes the historical and live tweets, removing unwanted characters, removing urls and punctuation.
- To produce a system for spam filter using probability likelihood for processed tweets. A naive Bayes approach may be suitable for this given task
- To produce a language detection and filtering system that removes all tweets that are not of the English language or containing non-basic-latin characters
- To provide a means for stemming, tokenisation and stopword removal to aid in data pre-processing for language detection and spam filtering

Neural Network

- To produce a neural network which trains on collected, historical and live data, to forecast the future price of Bitcoin, based on price and sentiement

- To produce a neural netowork which accomplished the same as the other above, but with out use of sentiment
- To produce metrics to justify accuracy of the model
- To produce data files containing, the current time of predictions alongside current hour price and sentiment. This should also include a suggested action based on a threshold for the price difference between hours.
- To produce data files containing the true and predicted price values of every hour for trained data, and another for current reoccuring predictions.

Interface

- To produce a basic interface which displays the predicted values alongside true price values with a time interval step of an hour. This can be displayed as both a table consisting of:

  Date of prediction, predicted price of next hour, current hour price and sentiment, and a suggested action based on a threshold for the price difference between hours.

  To produce charts displaying the true and predicted price values for every hour, from both start of new predictions made, and from training predictions
- To display a table of performance metrics of the trained model

Server

- This system, both prediction system and interface, should be deployed to a server due to the need to be constantly running

This project will not attempt to justify the accuracy of the chosen algorithm or tools over other algorithms. It will be discussed in the solution approach the justifications made on why the chosen algorithm and tools have been used for this project over the others, but accuracy will not be directly compared.

This project will only be coded to predict an hour ahead as the model will be trained on an hourly basis as the data is gathered per hour. Predictions for further in the future can be modelled but will be seen as a future improvement to the system.

The detail of a interface may be subject of change through this project due to time contraints and the focus being the investigation of the impact social media has on market predictions.

5.5 Project Constraints

The following constraints are recognised in this project

- ...

6 Quality Goals

7 Literature Review

7.1 Existing Tools

An aspect that this project will be attempting to address is that, at the time of writing, there are a limited amount of systems available to the public that either provide sentiment analysis or predictions of the crypto-market. Additionally, none known that combine both sentiment and price analysis to make said predictions on the direction of the market.

Such tools are usually provided by exchanges which correspond the amount of positive and negative sentiments with a suggestion to buy and sell. These tools, however, are vague in their suggestions as they don't provide any further analysis on when the best time would be to conduct an action on the market, and simply display the number of tweets per sentiment level. A well-known cryptocurrency tracking site,Coingecko provides a basic sentiment analysis tool for their top 30 ranking cryptocurrencies tracked on the site. This tool shows the sentiment analysis of tweets from Twitter every hour for a given cryptocurrency. This is displayed as a simple pill on the page showing the ratios of positive, neutral and negative tweets. *See Appendix C for visual representation*

7.2 Related research

There has been an abundant amount of research conducted in this problem domain. Many theses globally have been published in recent years on the topic of cryptocurrency market predictions and analysis, and even more, research conducted on general stock markets further back.

The thesis written by *Evita Stenqvist and Jacob Lonno* of the *KTH Royal Institute of Technology* [3] investigates the use of sentiment expressed through micro-blogging such as Twitter can have on the price fluctuations of Bitcoin. Its primary focus was creating an analyser for the sentiment of tweets more accurately *"by not only taking into account negation, but also valence, common slang and smileys"*, than that of former researchers that *"mused that accounting for negations in text may be a step in the direction of more accurate predictions."*. This would be built upon the lexicon-based sentiment analyser VADER to ascertain the overall sentiment scores were grouped into time-series for each interval from 5 minutes to 4 hours, along with the interval prices for Bitcoin. The model chosen was a naive binary classified vectors of predictions for a certain threshold to *"ultimately compare the predictions to actual historical price data"*. The results of this research suggest that a binary classification model of varying threshold over time-shifts in time-series data was "lackluster", seeing the number of predictions decreasing rapidly as the threshold changed. This research is a reasonable basis of starting research upon, as it suggests tools such as VADER for sentiment analysis and that the use of a machine learning algorithm would be a next step in the project that would yield better more accurate results.

Another thesis written by *Pagolu, Venkata Sasank and Reddy Kamal Nayan, Panda Ganapati and Majhi, Babita* [1] on "Sentiment Analysis of Twitter Data for Predicting Stock Market Movements" 2.5 million tweets on Microsoft were extracted from Twitter, sentiment analysis and logistical regression performed on the data yielded 69.01% accuracy for a 3-day period on the increase/decrease in stock price. These results showed a "*good correlation between stock market movements and the sentiments of the public expressed in Twitter*". Using various natural language pre-processing tweets for feature extraction such as N-gram representation the sentiment from tweets were extrapolated. Both Word2vec and a random forest classifier were compared for accuracy, Word2vec being chosen over the machine learning model. Word2vec, being a group of related shallow two-layer neural network models to produce word embeddings.

A topic that reoccurs in various papers and theses is that of the use and focus of regression techniques and machine learning methods. Few implement a fully fledged neural network; the above paper attempts to use a simple network to achieve predictions of classification of sentiment for stock market movement then correlated this with historical data of prices. An article posted on "Code Project" by Intel Corporation [5] compares the accuracy of three machine learning algorithms; Random Forest, Logistic Regression and Multi-Layer Perceptron (MLP) classifiers on predicting the price fluctuations of Bitcoin with embedded price indices. Results showing *"that using the MLP classifier (a.k.a. neural networks) showed better results than logistic regression and random forest trained models"*. This assumption can be backed up by the results from a thesis posted on IEEE [6] which compares a Bayesian optimised recurrent neural network and a Long Short Term Memory (LSTM) network - showing the LSTM model achieving *"the highest classification accuracy of 52% and a RMSE of 8%"*. With interest in neural networks personally and with little papers utilising them for this purpose a neural network will thus be implemented, and the accuracy of one's predictions with use of sentiment analysis data analysed and discussed.

7.3 Data Collection

7.3.1 Twitter and Twitter API

Twitter is a micro-blogging platform that was launched in 2006 and provides it's users the ability to publish short messages of 140 characters. The messages published could be of any form, from news snippets, advertisement, or the prevalent publication of opinions which allowed a platform of extensive diversity and knowledge wealth. As of the time of writing, the message character limit was increased to 280 characters, the platform has over 300 million monthly active users, and around 1 million tweets are published per day. Due to the length restriction and the primary use of the platform to express opinions Twitter is seen as a gold mine for opinion mining.

The Twitter API has an extensive range of endpoints that provide access from streaming tweets for a given hashtag, obtaining historical tweets for a given time-period and hashtag, posting tweets on a users account and to change settings on a user account with authentication. The exhaustive range of features provided by these endpoints makes data collection from Twitter straight forward as one can target a specific endpoint for the required data. Due to Twitter being the target for opinion mining within this project the Twitter API will ultimately need to be utilised. This can either be used for the gathering of historical tweets or streaming current tweets for the #Bitcoin hashtag.

There are, however, limitations and rate limits imposed on users of the API. Twitter employs a tiering system for the API - Standard, Premium and Enterprise tiers, each of which provides different amounts of access for data collection. If the API were to be used to capture historical data for a span of 3 months, each tier is allowed to obtain varying amounts of data for different durations; [7]

- A Standard user would be able to capture 100 recent tweets for the past 7 days
- A Premium user would be allowed to capture up to 500 tweets per request for a 30-day span and will have access to a full-archive search to query up to 100 tweets per request for a given time period, with a 50 request limit per month
- An Enterprise user would be able to capture up to 500 tweets per unlimited requests for a 30-day span and will be able to query the full-archive of tweets for a given hashtag up to 2000 tweets per unlimited amount of requests for a given time period

Each tier has individual costs while the standard user negating this as a basic tier. Due to only being elegable for the Premium tier for educational purposes, historical data gathering will be limited to 100 tweets per request with a limitation of 50 requests per month. Furthermore streaming tweets is an Enterprise feature which rules out the the Twitter API for use of streaming current real-time data [8].

7.3.2 Tweepy Python Package

Tweepy is a python package for accessing the Twitter API. It fundamentally accomplishes the same means if one to conduct a GET request to the Twitter API, except it simplifies this into a simple to use API that is easier to implement and automate in python [9]. Consequently, it builds upon the existing Twitter API to provide features such as automated streaming of provided hashtags to the API. It realises this by initialising a listener instance for a provided set of API credentials, handling authentication, connections, creating and destroying sessions. Due to Twitter's streaming API being only available to Enterprise users [7], using Tweepy to stream data for a given hashtag will provide the real-time data needed. The streaming of current data by Tweepy is accomplished by setting up a stream which listens for new data for a given hashtag, which bypasses the need for the Enterprise tweet tracker provided by the Twitter API.

7.4 Sentiment Analysis

In short, sentiment analysis is the process and discovery of computationally identifying and categorising the underlining opinions and subjectivity expressed in written language. This process determines the writer's attitude towards a particular topic as either being positive, neutral or negative in terms of opinion, known as polarity classification.

7.4.1 Natural Language Processing

Polarity classification is the focus of sentiment analysis and is a well-known problem in natural language processing that has had significant attention by researchers in recent years [1][3][6][10]. Traditional approaches to this have usually been classified to dictionary-based approaches that use pre-constructed sentiment lexicons such as VADER or usually confined to machine learning approaches. The latter requires an extensive amount of natural language pre-processing to extrapolate vectors and features from the given text; this is then fed into a machine learning classifier which attempts to categorise words to a level of sentiment polarity. Natural language pre-processing techniques, supported by the NLTK (Natural Language Toolkit) python package that would be required for this approach would consist of;

- Tokenisation: The act of splitting a stream of text into smaller units of typographical tokens which isolate unneeded punctuation.
- Removal of domain specific expressions that are not part of general purpose English tokenisers - a particular problem with the nature of the language used in tweets, with @-mentions and #-hashtags
- Stopword removal: Are commonly used words (such as "the","in","a") that provide no meaning to the sentiment of a given text
- Stemming: Is used to replace words with common suffixes and prefixes, as in "go" and "goes" fundamentally convey the same meaning. A stemmer will replace such words with their reduced counterparts
- Term Probability Identification and Feature Extraction: This is a process that involves identifying the most frequently used words in a given text, by using a probability type approach on a pre-defined dataset which classifies a range of

texts as with overall negative or positive a machine learning algorithm is trained to classify these accordingly.

- Ngrams: Are a contiguous sequence of n items from a given sample of text. The use of Ngrams in natural language processing can improve the accuracy of classification. For example: Good and Not Good have opposite meanings. By only using 1 token (1gram) not good (not and good) can be incorrectly classified. As the english language contains a significant amount of 2gram type word chains using 2gram can improve the accuracy of classification.

The former, seen and has been proven to provide higher accuracy than traditional machine learning approaches [11], and need little pre-processing conducted on the data as words have a pre-defined sentiment classification in a provided lexicon. Although these lexicons can be complex to create, they generally require little resources to use and alter.

7.4.2 Valence Aware Dictionary and sEntiment Reasoning

VADER is a combined lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media and works well on texts from other domains. It is capable of detecting the polarity of a given text - positivity, neutrality, and negativity [12], and also calculate the compound score which is calculated by summing the valence scores of each word in the lexicon. VADER uses a human-centric approach to sentiment analysis, combining qualitative analysis and empirical validation by using human raters to rate the level of sentiment for words in its lexicon. Vader also has emoticon support which maps these colloquialisms have pre-defined intensities in its lexicon, which makes VADER specifically suitable for the social media domain were the use of emoticons, utf-8 emojis and slang such as "Lol" and "Yolo" are prevalent within the text. Additionally, VADER is provided as a lexicon and a python library under the MIT license, this means that it is open-source software. This means that the lexicon can be altered and added to abling it to be tailored to specific topic domains.

VADER was constructed by examining and extracting features from three pre-existing well-established and human-validated sentiment lexicons [12] - (LIWC) Linguistic Inquiry and Word Count, (ANEW) Affective Norms for English Words, and (GI) General Inquirer. This is supplemented with additional lexicon features *"commonly used to express sentiment in social media text (emoticons, acronyms and slang)"* [12] and uses "wisdom-of-the-crowd" approach [13] to establish a point of estimations of sentiment valance for each lexical feature candidate. This was evaluated for the impact of grammatical and syntactical rules and 7,500+ lexical features, with mean valence *"¡¿ zero, and SD ¡= 2.5"* as a human-validated "gold-standard" sentiment lexicon. [12]*Section 3.1*

VADER is seen as a "Gold Standard" for sentiment analysis, in the paper for VADER, [12] *A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text*, it was compared against 11 other *"highly regarded sentiment analysis tools/techniques on a corpus of over 4.2K tweets"* for polarity classification across 4 domains. Results showing VADER, across Social media text, Amazon reviews, movie reviews and Newspaper editorials, consistently outperforming other sentiment analysis tools and techniques showing a particular trend in performing significantly higher on analysis of sentiment in tweets. [12] *Section 4: Results*

7.5 Neural Networks

A neural network is a set of perceptrons modelled loosely after the human brain that is designed to recognise patterns in whatever domain it is intended to be trained using. A neural network can consist of multiple machine perceptrons or clustering layers in a large mesh network, and the patterns they recognise are numerical which are contained in vectors. Pre-processed data, confined and processed into pre-defined vector labels, are used to teach a neural network for a given task. While this differs from how an algorithm is coded to a particular task, neural networks cannot be programmed directly for the task. The requirement is for them to learn from the information by use of different learning strategies; [14][15]

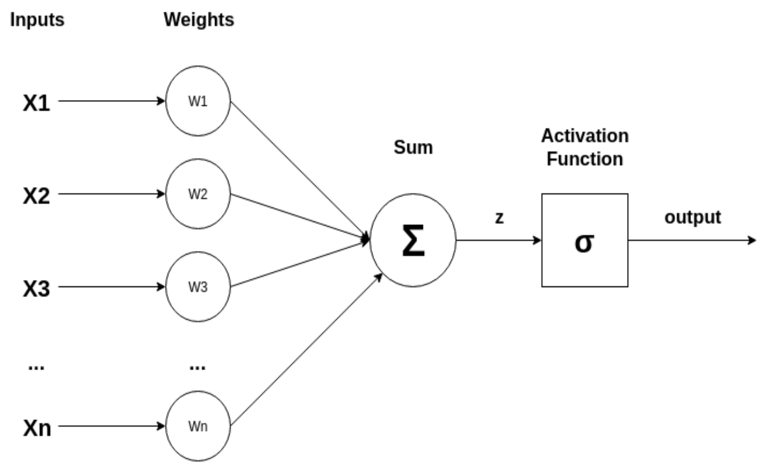


Figure 1: Basic perceptron layout

- Supervised learning: Simplest of the learning forms, where a dataset have been labeled which indicate the correct classified data. The input data is learned upon until the desired result of the label is reached [16]
- Unsupervised learning: Is training the with a dataset without labels to learn from. The neural network analyses the dataset with a cost function which tells the neural network how far off target a prediction was. The neural network then adjusts input weights in attempt to increase accuracy. [15]

- Reinforced learning: The neural network is reinforced with positive results and punished for negative results causing a network to learn over iterations.

7.5.1 Recurrent Neural Network (RNN)

The type of neural network that is of focus for this project will be that of a Long-Short Term Memory (LSTM); however, it is crucial to understand how this is an extension of a Recurrent Neural Network (RNN) and how the underlying network works.

Recurrent Neural Networks (RNN) are a robust and powerful type of neural network and is considered to be among the most encouraging algorithms for the use of classification, due to the fact of having internal memory. RNNs are designed to recognise patterns in sequences of presented data or most suitably, time-series data, genomes, handwriting and stock market data. Although RNNs were conceptualised and invented back in the 1980s [17], they've only really shown their potential in recent years, with the increase of computational power due to the level of sequencing and internal memory store to retrain. Due to this 'internal' memory loop, RNNs are able to remember data and adjust neurons based on failures and alternating parameters. The way this is accomplished, knowing how a standard neural network such as a feed-forward network, should initially be understood. [18]

A standard, feed-forward neural network has a single data flow with an input layer, through hidden computational layers, to an output layer. Therefore any node in the network will never see the same data again. However, in an RNN data is cycled through a loop over the same node, thus two inputs into the perception. Decisions are influenced by previous data that it has previously learned from if any, which in turn affects output and the weights of the network. [19]

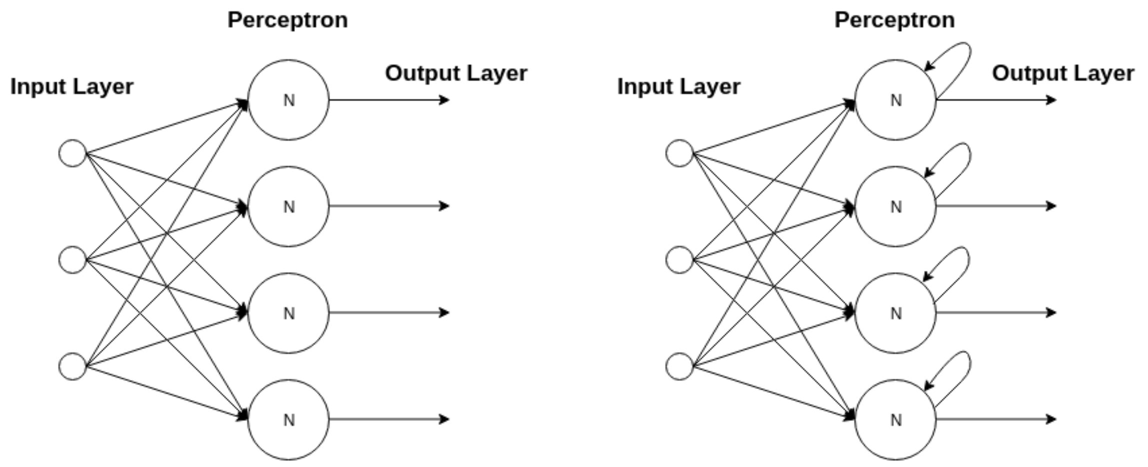


Figure 2: Feed-forward network (left) vs Recurrent Neural network (right)

The act of tweaking weights to alter the processing of the next iteration of data in an RNN is called backpropagation, which in short means going back through the network to find the partial derivatives of the error with respect to the weights after output has occurred. Along with gradient descent, an algorithm that adjusts the weights up or down depending on which would reduce the error. There are however a few obstacles of RNNs;

- Exploding Gradients: Is when gradient decsent assigns high importance to the weights. As in the algorithm assigns a ridiculously high or low value for the weights on iteration which can cause overlow and result in NaN values [20]
- Vanishing Gradients: Is when the values of a gradient are small enough that weights cannot be altered and the model stops learning. [21]

These issues are overcome by the concept of Long-Short Term Memory neural networks, coined by *Sepp Hochreiter and Juergen Schmidhuber, 1997* [22].

7.5.2 Long-Short Term Memory (LSTM)

LSTMs are an extension of recurrent neural networks capable of learning long-term dependancies and were conceptualised by *Sepp Hochreiter and Juergen Schmidhuber, 1997* [22]. LSTMs were explicity designed to avoid long-term dependancy problems such as exploding and vanishing gradients. As they are an extension of RNNs they operating in almost the exact same manner, but stores the actual gradients and weights in memory which allows for LSTMs to read, write and alter the values. A way of explaining how this works is seeing the memory block as a gated cell, where 'gated' is that the cell decides whether or not to store or alter data in it's memory based input data and the importance assigned to it. In a sense it learns over time of which values and data is important.

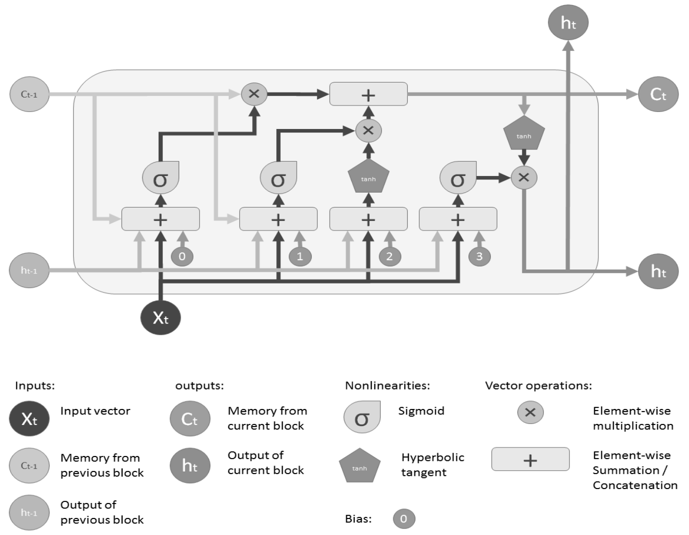


Figure 3: A conceptual design of an LSTM cell bank - from Medium article by Shi Yan: Understanding LSTM and its diagrams[23]

The network takes in three initial inputs, the input of the current time step, output from the previous LSTM unit if any, and the memory of the previous unit. Outputs, H_t - output of current network, and C_t - the memory of the current unit. [23]

The various steps of the network decide what information is thrown away from the cell state, through use of a 'forget gate' which is influenced by the calculations of sigmoid memory gates which influence how much of old and new memory is used C_{t-1}, H_{t-1} and X_t, and merged based upon importance. The section of the cell that controls the outflow memory H_t and C_t determines how much of the new memory should be used by the next LSTM unit. *For a more in-detailed explanation of exactly how the calculations are made see* [22],[23] and [24].

As mentioned in the foremost section of related work the use of an LSTM network would be optimal for the given problem domain over the use of machine learning algorithms, for time-series data. As detailed above, LSTMs are widely used for time-series data forecasting due to being able to remember previous data and weights over long sequence spans[22][25]. The flexibility of LSTMs such as many-to-many models, useful *"to predict multiple future time steps at once given all the previous inputs"* due to use of look-back windows and control of multiple 3D input parameters.[25]

7.5.3 Keras and TensorFlow

TensorFlow is an open-source numerical math computational library framework for dataflow differentiable programming, primarily used for machine and deep learning applications such as neural networks. TensorFlow bundles various machine learning and deep learning models and algorithms into one package for the Python language, but executes as byte code executed in C++ for performance. TensorFlow provides a range of conveniences to developers for the types of algorithms it supports such as debugging models and modifying graph operations separately instead of constructing and evaluating all at once, and compatibility to execute on CPUs or GPUs [26]. However, TensorFlow's implementation and API, although provides an abstraction for development for machine and deep learning algorithms and simplifies implementation, it isn't all too friendly to programmers to use, especially new developers to the field of machine and deep learning.

Keras is a high-level built to run atop of deep learning libraries such as Tensorflow and Theanos - another deep learning library similar to Tensorflow. It is designed to further simplify the use and application of such deep learning libraries thus making implementing a neural network and similar supported algorithms friendlier to developers working in Python. It accomplishes this by being a modular API; neural layers, cost functions, optimisers, activation functions, and regularisation schemes are all standalone features of the API that can be combined to create functional or sequential models. Due to being a high-level API for a more refined and more natural development of deep learning libraries, it does not provide these low-level operations and algorithms; Keras relies on a back-end engine such as TensorFlow and supports a wide range of others.

7.5.4 Optimisers

There are three distinct optimisers used for LSTM networks; ADAgrad optimizer, RMSprop, Adam. The role of an optimiser All three of which is a type of Stochastic Gradient Descent, which θ (weights of LSTM) is changed according to the gradient of the loss with respect to θ. Where α is the learning rate and L is the gradient loss. [27]

$$\theta_{t+1} = \theta_t - \alpha \delta L(\theta_t)$$

This is primarily used in recurrent LSTM neural networks to adjust weights up or down depending on which would reduce the error, *see RNN section for non LSTM limitations.* The concept of using momentum μ in stochastic gradient decent helps to negate significant convergance and divergance during calculation of the weights and dampens the oscillation, by increasing the speed of the learning rate upon each iteration. [28]

$$\theta_{t+1} = \theta_t + v_{t+1}$$

where

$$v_{t+1} = \mu v_t - \alpha \delta L(\theta_t)$$

[28]

- Adagrad (Adaptive Gradient): Is a method for adaptive rate learning through adaptively changing the learning parameters. This involves performing more substantial updates for infrequent parameters and smaller updates for frequent parameters. This algorithm fundamentally eliminates the need to tune the learning rate of the neural network manually and is well suited with sparse data in a large scale network. [28]

$$\theta_{t+1} = \theta_t + v_{t+1} \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot g_t$$

(G_t is the sum of the squares of the past gradients to θ)

- RMSProp (Root Mean Square Propagation): Aims to resolve Adagrads radically diminishing learning rates by using a moving average of the squared gradient. Thus utilises the magnitude of the recent gradient descent to normalise it, and gets adjusted automatically by choosing different learning rate for each parameter. [29]

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{(1-\gamma)g_{t-1}^2 + \gamma g_t + \epsilon}} \cdot g_t$$

(γ - decay that takes value from 0-1. g_t - moving average of squared gradients)

[30]

- Adam (Adaptive Moment Estimation): Also aims to resolve Adagrads diminishing learning rates, by calculates the adaptive learning rate for each parameter. Being one of the most popular gradient descent optimisation algorithms, it estimates from the 1st and 2nd moments of gradients. Adam implements the exponential moving average of the gradients to scale the learning rate of the network and keeps an average of past gradients. [31]

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2$$

The algorithm updates the moving averages of the gradient (m_t) and the squared gradient (v_t) which are the estimates of the 1st and 2nd moments respectively. The hyperparameters β_1 and β_2 control the decay rates of the moving averages. These are initialised as 0 as a biased estimations for the initial timesteps, but an become bias-corrected by counteracting them with;

$$\vec{m}_t = \frac{m_t}{1-\beta_1^t}$$

and

$$\vec{v}_t = \frac{v_t}{1-\beta_2^t}$$

Thus the final formula for the Adam optimiser is;

$$\theta_{t+1} = \theta_t - \frac{\eta \vec{m}_t}{\sqrt{\vec{v}_t} + \epsilon}$$

Diederik P. Kingma, Jimmy Lei Ba - Adam: A method for Stochastic Optimization[30]

7.6 Machine Learning

7.6.1 Naive Bayes

To get an understanding of both how probability works and how a neural network will predict the next hour value based on the concepts of probability, using a well-established probability algorithm will aid in this understanding.

Bayes theorem works on conditional probability and is the probability of how often an event will happen given that that event has already occurred. There are numerous variations of the theorem such as Multinomial, which supports categorical features where each conforms to a multinomial distribution, and Gaussian naive Bayes, which support continuous-valued features each of which conforming to a Gaussian (normal) distribution. The classical multinomial Bayes theorem is defined as; [32]

$$P(H \cap A) = \frac{P(A \cap H) * P(H)}{P(A)}$$

and incase H and A are independant

$$P(H \cap A) = P(H) => P(H \cap A) = P(H)P(A)$$

where:

- $P(H)$ is the probability of hypothesis being true
- $P(A)$ is the probability of evidence
- $P(A \cap H)$ is the probability of the evidence such that the hypothesis is true
- $P(H \cap A)$ is the probability of the hypothesis given the occurance of evidence of the probability

The naive approach assumes the features that are used in the model are independent of one another, such that, changing the value of a feature doesn't directly influence the value of the other features used in the model. When such features are independent, the Bayes algorithm can be expanded:

$$P(H \cap A) = \frac{P(A \cap H) * P(H)}{P(A)}$$

Becomes

$$P(H \cap A_1...A_n) = \frac{P(A_1 \cap H) * P(A_2 \cap H)... * P(A_n \cap H) * P(H)}{P(A_1) * P(A_2)... * P(A_n)}$$

$$Probability\ of\ Outcome \cap Evidence = \frac{Probability\ of\ Likelihood\ of\ evidence * Prior}{Probability\ of\ Evidence}$$

The naive Bayes approach has many applications, especially for the topic of this project in classifying the probability occurrence of the next price. Although it is a robust algorithm has its drawbacks which make it not as suitable as a neural network for the given need of this project. The naive Bayes trap is an issue that may occur due to the size of the dataset that will be used. There are however other scenarios this algorithm could be used such as classification of spam data.[32]

7.7 Bag Of Words

The Bag Of Words algorithm counts the occurance ('Term-Frequency') of a word in a given text or document. The counts allow the comparison for text classification and is used prior to TF-IDF (detailed below) to aid in identifying the probability of words in a given text and classify accordingly. [33]

$$P(w)\ and\ P(w|spam) = \frac{Total\ number\ of\ occurrences\ of\ w\ in\ dataset}{Total\ number\ of\ words\ in\ dataset}$$

7.8 TF-IDF

Stands for Term Frequency-Inverse Document Frequency is another way similar to Bag of Words that are used to judge the topic of a given text. Each word is given a weight (relevance not frequency) for how many times it occurs in the given text [33]. Term-frequency measures the number of times that a word appears in the text, but due to words such as 'and', 'the' and 'a' can frequently appear in text Inverse Document Frequency is used to change the weight of words that appear the most. Therefore words that appear the most are signalled to be less important and valuable, and therefore will not be used for classifications when used with such models as Naive Bayes for a given purpose. [33]

IDF is defined as:

$$IDF(w) = log \frac{Total\ number\ of\ messages}{Total\ number\ of\ messages\ containing\ w}$$

TF-IDF is thus defined as both:

$$P(w) = \frac{TF(w) * IDF(w)}{\sum_{a\ words\ x\ \epsilon\ train\ dataset} TF(x) * IDF(x)}$$

$$P(w|spam) = \frac{TF(w) * IDF(w)}{\sum_{a\ words\ x\ \epsilon\ train\ dataset} TF(x|spam) * IDF(x)}$$

[34]

7.9 Addictive Smoothing

Used alongside Bag Of Words, is a method of handling data that is in the test data but not in the training dataset. In case of $P(w)$ it would evaluate to 0, which will make the $P(w|spam)$ undefined as it will not be able to classify the word. Addictive smoothing tackles this by adding a number α to the numerator, and adding *alpha* time the number of classes over the probability that is found in the denominator. [34]

For TF-IDF:

$$P(w|spam) = \frac{TF(w) * IDF(w) + \alpha}{\sum_{a\ words\ x\ \epsilon\ train\ dataset} TF(x|spam) * IDF(x) + \alpha \sum_{a\ words\ x\ \epsilon\ train\ dataset}}$$

8 Solution Approach

This section will outline the solution intended to solve the problem that the problem statement identifies, with justification and reference to the research conducted in the literature review. This will lay out the development process for the project and will tools and technologies will be explained for the particular use case in this project.

8.1 Data gathering

This will be the part of the system that will gather price data and tweets from relevant sources, Twitter and cryptocurrency exchanges.

Price data

Historical price data can be collected in a number methods, one being that of the exchange APIs, another through a historical price tracker who creates a CSV consisting of all prior historical data. Both have their merits and reliability for granting the needed data; however, a historical tracker who has been tracking the price every hour since the start of Bitcoin would be the better option. This is due to a couple of factors, the data in some historical trackers are an average unbiased price for Bitcoin - they track the price of all or a select few exchanges and average the hourly price. Whereas if the historical data was obtained directly from an exchange this would be biased and might not represent the true price of the currency, and thus would need averaging with other hourly prices from other exchanges. By using a historical tracker, all the data is unbiased and averaged and readily available and doesn't require any requests to an API or coding needed to process data.

Live price data can be collected through the same methods, a historical price tracker and an exchange API. However, this doesn't work the same way; unfortunately, a historical price tracker is not updated as frequently as exchange APIs thus wouldn't provide on the hour accurate data. Therefore exchange APIs will be utilised in this case and multiple to give an unbiased average for the hourly price. Three exchanges will provide a sufficient average, and the exchanges most likely to be used would be the more popular exchanges such as Coinbase, Bitfinex and Gemini.

Tweets

Historical tweets can be obtained through the Twitter API, and however is not a feature of the Tweepy package - *not mentioned or method on official Tweepy Documentation* [35]. The Twitter API, as explained in the Literature review, allows for historical tweets to be extracted from the platform, 100 per request and a maximum of 50 requests per month. This proposes an issue with not providing enough data, where the sentiment will need to be calculated per hour. Simply put, for a year of hourly price data, there will be 9050 records. Therefore the equivalent will be required for sentiment; however, the sentiment will be the average the sentiment per hour of tweets. Using a single request with 100 tweets per hour, per hour; 905,000 tweets will need to be extracted to provide the data required. A solution to this issue could be to use and create multiple accounts and manually extract data from the API and merge. Another option is the pay for the data from 3rd party companies who have access to the Enterprise API and can pull more data, 2000 per request [7][8]. Due to the price for data of these 3rd parties the former could be a suitable, but more time-consuming option.

Live tweets can be collected by two methods from Twitter, from the Twitter API and using Twitter Python package such as Tweepy, detailed in the Literature review. Additionally, the limitations of the Twitter API are also discussed in the literature review which states how the Twitter API has a tiering system: Standard, Premium and Enterprise. Each tier has different levels of access to the API and can extract varying amounts of data from the platform. Thus concluding the section in the Literature review, the Twitter API will not be used for the extraction and streaming of live tweets due to it being restricted to Enterprise users. Therefore, Tweepy will be used to set up a looping authenticated streaming solution with the Twitter API which will allow the access of current recurring data. Natural language pre-processing will be apart of most systems in this project. Techniques such as tokenisation, stemming, stopword removal and character filtering will be prevalent, as these will be used to remove unwanted data and to sanitise the data for classification.

8.2 Data pre-processing

Natural language pre-processing will be apart of most systems in this project. Techniques such as tokenisation, stemming, stopword removal and character filtering will be prevalent, as these will be used to remove unwanted data and to sanitise the data for classification.

8.3 Spam Filtering

This part of the system will aim to detect whether or not the streamed data or the historical data is spam - unwanted tweets that serve no purpose in determining the opinion of the public. These types of tweets can be from advertisement - usually labelled with *#Airdrop* and can contain *"tickets here" and "Token Sale"*, to job advertisements - usually containing word such as *Firm, hire, hiring, jobs and careers.* It is essential to filter out and remove such data from the network as these can be seen as outliers of the true data and will skew predictions will invalid sentiment.

The spam filter will use a probability-based algorithm such as Naive Bayes, other algorithms such as Random Forest could be used, but due to this being a probability related problem using an algorithm such as Naive Bayes would be more suitable. This classifier will be trained on a hand created dataset containing both spam and ham (*wanted data*) tweets, and should not be exclusive to either category.

8.4 Language Detection

Before performing any natural language pre-processing and spam filtering, non-English tweets will need to be reduced. This can be introduced through various language detection filtering using techniques such as ngrams alongside other natural language pre-processing techniques to filter out non-English characters. Fortunately, both Tweepy and the Twitter API have methods for specifying the desired language to receive tweets in - *filter=['en']* for the Tweepy streaming method and *query={...,language='en',...}* on the JSON parameters for the Twitter API. This provides a simple means of filtering out non-English tweets, but this only filters based on region and user settings which indicate the users desired language. Thus if a user has their region set to *'en'* or has their desired language set also as *'en'* the tweet will be classified as English but may contain non-English characters.

As is the case, a suitable language detection system will be implemented to identify any tweets that contain non-English characters. Some tweet will regrettably make it past the initial API filters; thus such a system will be implemented that will drop the tweets if it predominately contains non-English characters. If however, the majority of the text in English but includes some non-English characters, these will be removed from the tweet.

8.5 Sentiment Analysis

As mentioned in the Litrature review, the VADER sentiment analysis performs exceptionally well on the social media domain when compared to idividual human rates and 10 other highly regarded sentiment analysers, stated in the results section of the paper *VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text* [12].
Extraction of results from paper [12]:

Analyser	Overall Precision	Overall Recall	Overall F1 Score
Ind. Humans	0.95	0.75	0.84
VADER	**0.99**	**0.94**	**0.96**
Hu-Liu04	0.94	0.66	0.77
SCN	0.81	0.75	0.75
GI	0.84	0.58	0.69
SWN	0.75	0.62	0.67
LIWC	0.94	0.48	0.63
ANEW	0.83	0.48	0.60
WSD	0.70	0.49	0.56

Analysis of Social Media Text (4,200 Tweets)[12]

Due to the suitability for the given domain of social media and with the customisability, due to VADER's lexicon-dictionary based approach, makes this sentiment analyser most suitable for use in this project. This analyser will be utilised as the sentiment analyser of this project due to its feature set and need for little data pre-processing before polarity classification of the provided text. [11] *"is a widely used approach to sentiment analysis in the marketing research community, as it does not require any pre-processing or training of the classifier."*.

This will be an intermediate system between the neural network and the data collection pre-processing system, as the later will provide the cleaned processed data for analysis and the former to feed in the classified polarity of each tweet alongside price data for model learning.

8.6 Neural Network

The *Neural Network* section in the literature review details how Recurrent Neural networks work alongside how a Long-short term memory networks build upon and overcome limitations and known issues with a standard RNN network. A recurrent neural network is the focus of this project, and this is due to:

- Nature of an RNN - Allows for backpropagation to find partial derivatives of the error with respect to the weights after an output has occurred, to tweak the current weights of the LSTM cell. In short, allows the tweaking of weights of the network based on previously seen data by looping the same node thus influencing decisions made on current data based on old weights and errors from previous.
- Nature of an LSTM over RNN - LSTMs are extensions of RNNs [22] that were designed to avoid long-term dependency problems such as exploding and vanishing gradients. Weights are not only just reused but are stored in memory and are propagated through the network.

- Lack of use for the project's purpose - Other papers tend to focus on machine learning techniques, other neural networks such as Multi-layer Perceptron (MPL) and standard Recurrent Neural Networks, with use of time-series data. Especially with the use of a standard RNN, not overcoming its common issues with gradient descent. Stated in related research section of the literature review, [5] - *"using the MLP classifier (a.k.a neural networks) showed better results than logistical regression and random forest trained models"*

- Prior use for time-series data and data forecasting - Although RNN LSTM networks have been used for the prediction of Bitcoin price there are a few papers on this [25]. Regardless, LSTMs have been notably used with use for time-series data forecasting due to being able to remember previous data and weights over long sequence spans [25] - *"adds a great benefit in time series forecasting, where classical linear methods can be difficult to adapt to multivariate or multiple input forecasting problems"*.

Therefore, a recurrent long-short-term memory neural network will be used for this project to predict the next hour interval of Bitcoin price based on previous historical prices and hourly sentiment. This system will read in historical data, both price and sentiment - depending on the network for prediction with and without sentiment, this data will be merged, split and used to trained and test the network model for use for forecasting prices. The relative sizes for the training and test data can be decided upon system creation, but the standard sizing for training neural networks is 75:25 respectively.

Tensorflow will be used for the network implementation, and the Keras API use upon it to make development more straight forward. Other tools are comparable to TensorFlow that are also supported by Keras.

Framework	**Pros**	**Cons**
TensorFlow	Supports reinforcement learning and other algorithms	Doesnt support matrix operations
	Offers computational graph abstraction	Doesn't have pertained models
	Faster compile time than Theano	Drops to Python to load each new training batch
	Data and model parallelism	Doesn't support dynamic typing on large scale projects
	Can be deployed over multiple CPUs and GPUs	
Theano	Computational Graph Abstraction	Is low-level
	Has multiple high-level wrappers similar to Keras	Can only be deployed to a single GPU
		Much slower compile times on large models than competition
		Unhelpful and vague error messages
		Development ceased in 2017
Pytorch	Graph definition is more imperative and dynamic than other frameworks	Not as widley adopted as TensorFlow
	Graph computation defined at runtime, allowing standard popular IDEs to support it	Visualisation is not as robust as TensorBoard
	Natively support common python deployment frameworks such as Flask	Not as deployable as TensorFlow, doesn't supper gRPC

Comparison between TensorFlow, Theano and Pytorch[36]

Due to the continued support and development of TensorFlow, the board community and support of a high-level wrapper - Keras, this library will be used for this project. Although, Pytorch is a good alternative it is not as easy to use as implement when compared to TensorFlow using Keras.

8.7 Price Forecasting

This part of the system will be responsible for prediction the next time-step of Bitcoin's price for the next hour based on past data. It will use the trained model from the neural network to predict the future hour price when given live hourly data, price and sentiment. The system will also have a look back of 5 which will allow it to see historical data to aid in the predictions. This will occur on the hour every hour when new data is received and processed, this data will also be merged and the split into training and testing data. The sizing can be decided upon system creation, but the standard sizing for training is 75:25, training and testing respectively.

8.8 Frontend Application

The frontend application will display the predicted data to the stakeholders and users of the system, along with charting True hourly prices against Predicted, for both with and without sentiment embedded in the predictions. The interface will display this data in both tabular and chart form to provide variety to the user. Performance metrics will also be displayed at the bottom of the application to show the accuracy of the model. Due to this project focusing around the backend, how the predictions are made and the accuracy of the model, the interface will be somewhat of a second thought. It will aim to display the information in a clear and concise manner which will start to solve the problem of providing a system to the public to aid in investment decisions. The design will not be complicated but more basic and functional. Therefore a basic webpage coded in HTML with Jquery to plot data, and Ajax requests to obtain and load data, will be sufficient.

8.9 With reference to Initial PID

Both the problem and solution have changed considerably from the original project initiation document (PID), which outlines the initial ideas, objectives and specification for the project. The reason for this was due to a change in direction which was caused by a number of factors; one being a change in passion after initial research into machine learning techniques and neural networks, instead of creating an application that just performed sentiment analysis the direction turned towards how this could be used to predict future prices. This change does still loosely keeps in-line with the initial idea of wanting to create a platform that will aid in investor decision making but takes it a step further by directly giving them predictions on market direction price as a basis for these decisions rather than just identifying opinion direction of the market. Another point was the simplicity of the initial idea, which consisted of focusing more work on the design of the frontend application to display opinion data and general price data on a range of cryptocurrencies which will only by consuming exchange APIs. Both the developer and project supervisor concluded that this initial idea was too simple and a more sophisticated approach needed forming. The initial PID did, however, give an initial basis to base ideas and initial research from and was the beginning drive of this project.

8.10 Solution Summary

The overall solution, concerning the problem statement, is to create a system mainly consisting of; a frontend application that will display plotting, predicted and true, performance metric data to the user as a clear and concise form. The backend system behind the price forecasting will consist of various subsystem responsible for data collection, filtering, data pre-processing, sentiment analysis, network training, validation and training and future price predictions. Each stage will consist of relevant tools and techniques for performing their required task.

8.11 Data flow Overview

To get an understanding of how the system will be put together, a dataflow diagram is a useful method for view how systems are integrated and how data could possibly flow through a system.

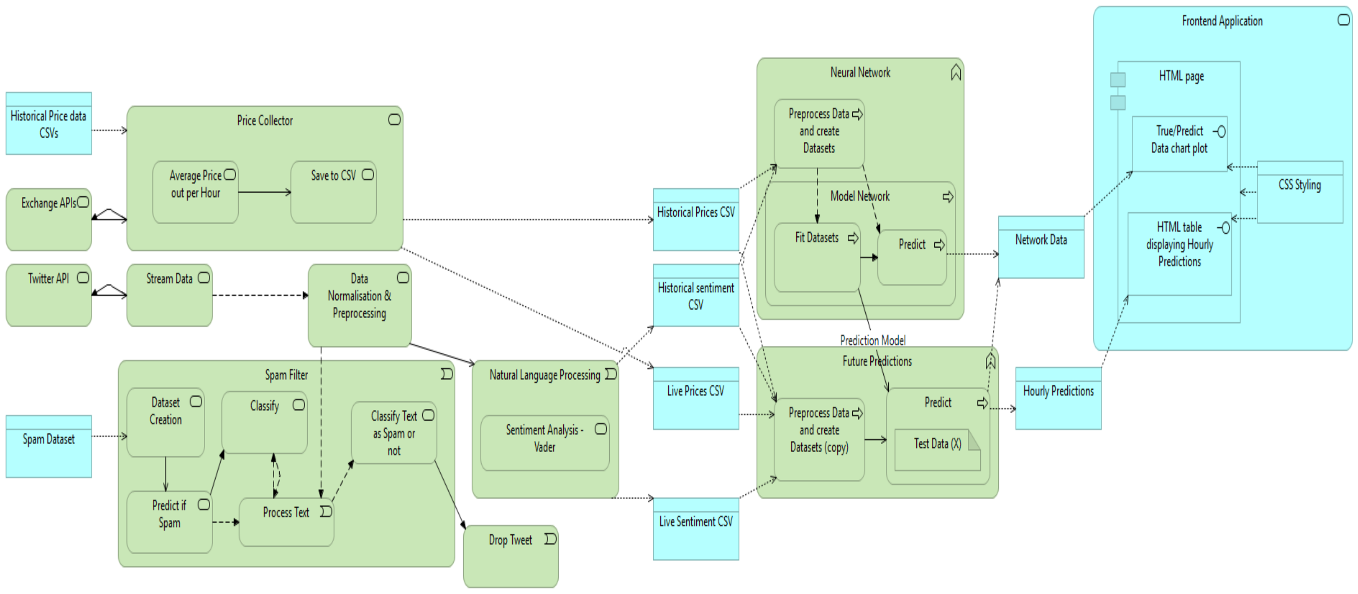


Figure 3: Basic Dataflow diagram of systems in the project and how data could possibly flow

9 System Design

9.1 Dataflow Designs

This section will describe and outline how the system will be formed and will work with each component; a useful way of displaying this is as a dataflow diagram. A dataflow is a way of representing the flow of data through a process or system; as a result, it also provides information about how inputs and outputs of each component work and how they function with other components. It can also give either broad or in-depth overview of the specific workings of each component through how the data is processed and manipulated.

Dataflow overview of entire system:

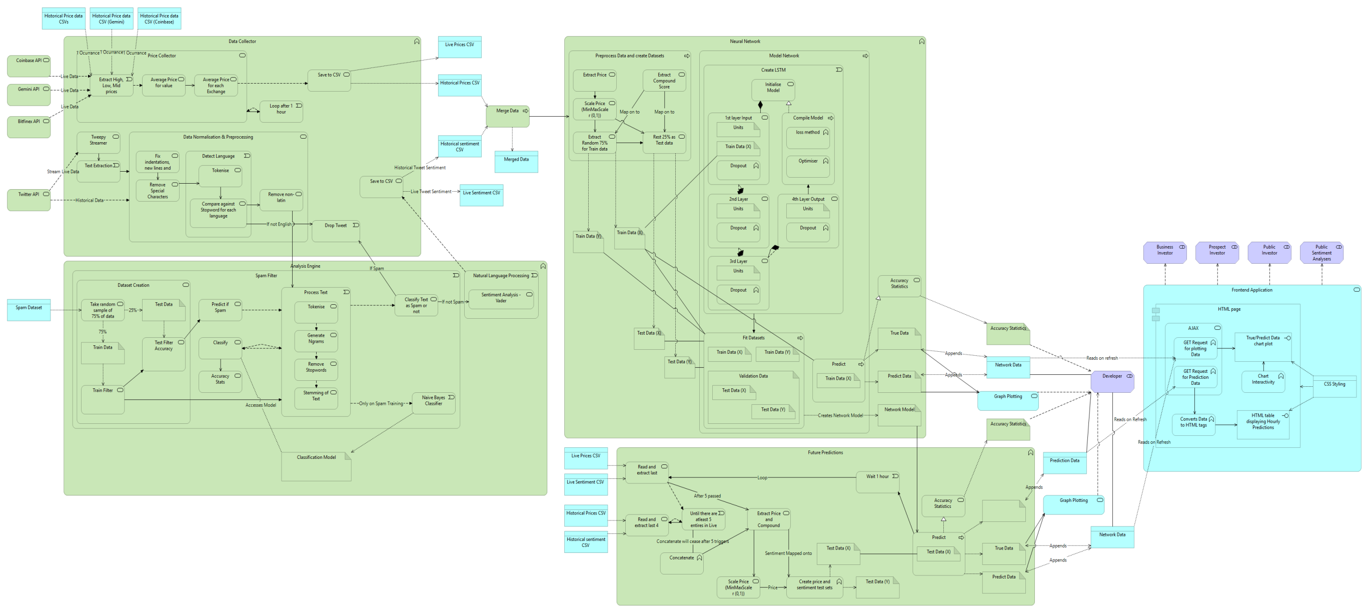


Figure 4: Overall Dataflow diagram of the entire system

This dataflow diagram shows the overall concept of how the data is intended to flow through the system, from being processed and manipulated through each component and what the outputs are of each. Due to the size, this will be broken up and individually explained.

Data collector

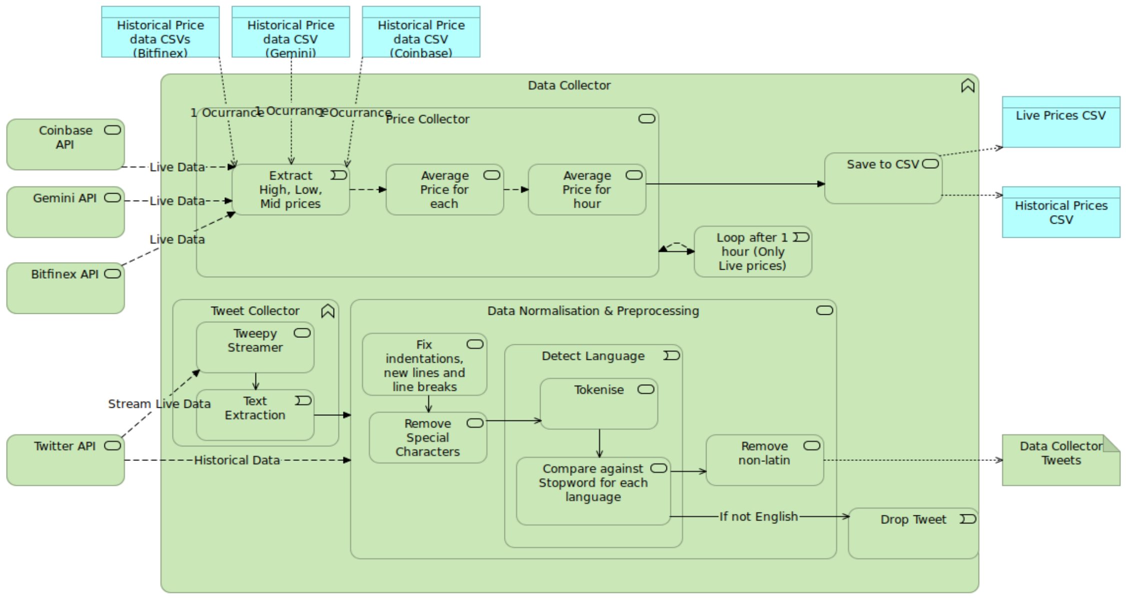


Figure 5: Data collector Dataflow diagram

This dataflow diagram shows the part of the system responsible for the collection and processing of both historical data. This is split into three parts: Price collector, Tweet collector and tweet normalisation and natural language pre-processing.

- Price Collector - Processes two forms of data, Historical and Live price data.

  Historical data is extrapolated from three CSVs that contain the historical price every hour for the past year, from a historical price tracker. At this point in the project, it was identified that historical price trackers do not average the price data from exchanges as previously identified; therefore this data will need to be merged and averaged to create the unbiased hourly price needed.

  Live data is extracted directly from the three exchanges APIs shown through REST endpoint requests.

  Data from both, as separate processes independent from one another, are averaged by extracting the *High*, *Mid* and *Low* hourly prices. This averaged price per hour for each exchange are then averaged together to obtain an unbiased hourly average. The price is then saved to a CSV of historical or live prices respectively. The difference in the flow of data is that of Live prices, in which the process is looped every hour to extract the new hourly prices.

- Tweet Collector - Streams tweets from Twitter using Tweepy, historical tweets are manually collected directly from the Twitter API. Both are fed through the normalisation and data pre-processing stage.

- Data pre-processing - This involves cleaning the initial data by removing line breaks and new lines that occur in the data, removal of special characters that are standard in tweets (*’#’,’’ and urls*). The data is then fed into a language detection system which tokenises and compares stopwords in the text to NLTK package supported languages. Depending on whether the text is identified as being predominately English or not determines whether or not the tweet is dropped and not used in the network. If the majority is in English, non-English characters are removed as these can still be present in the text.

Analysis Engine

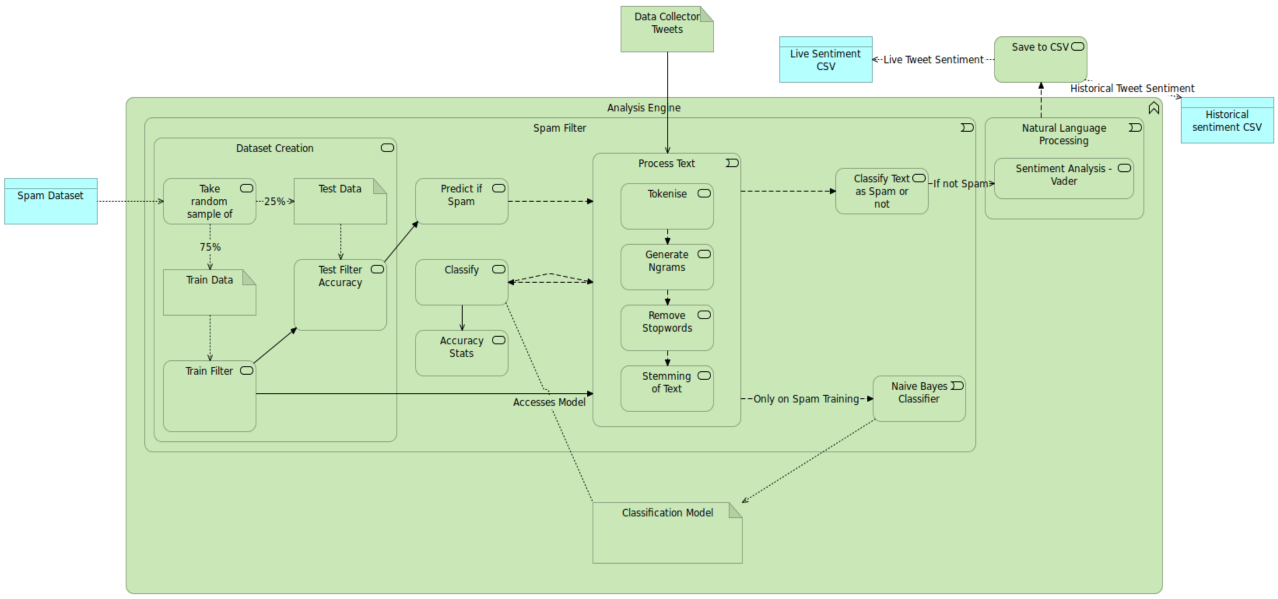


Figure 6: Analysis Engine Dataflow diagram

This dataflow diagram shows the part of the system that is responsible for training a spam filter, creating the model that’ll be used to identify if the tweets from the data collector are unwanted - spam. This system is also responsible for assigning the polarity classification to the tweet through sentiment analysis conducted by the VADER package [12].

- Spam filter training - The initial step in this system is to train the Naive Bayes Classifier using the pre-labelled spam dataset which contains an unbiased amount of either spam or ham tweets with their respective labels.

  This data is split into two samples, training and test sets 75:25 respectively and the Naive Bayes classifier trained and validated against these datasets after pre-processing of the data occurs on the data to prepare it.

- Data pre-processing - The tweets from both training and testing the filter and from live and historical tweets are processed through this section.

  This section of the system is primarily used to process the tweets for the filter to classify the data and doesn't directly modify the live and historical tweets. The data is processed through various natural language processing techniques such as; Tokenisation, Ngram generation, stopword removal and stemming.

- Classifier Modelling and Model creation - Once the data is pre-processed, the data is classified, and the prediction model created, which later used to classify the historical and live tweets.

- Sentiment Analysis (VADER) - On a separate route from the spam filter training, using the past and live tweets, the sentiment analyser VADER performs analysis on the tweets and assigns a polarity classification to each text (*Negative, Neutral, Positive* and calculates the compound score which is the difference between the negative and positive ratings *compound*).

- Storage - The polarity classification and tweets are then saved to their relevant CSV files for historical and live data.

Neural Network

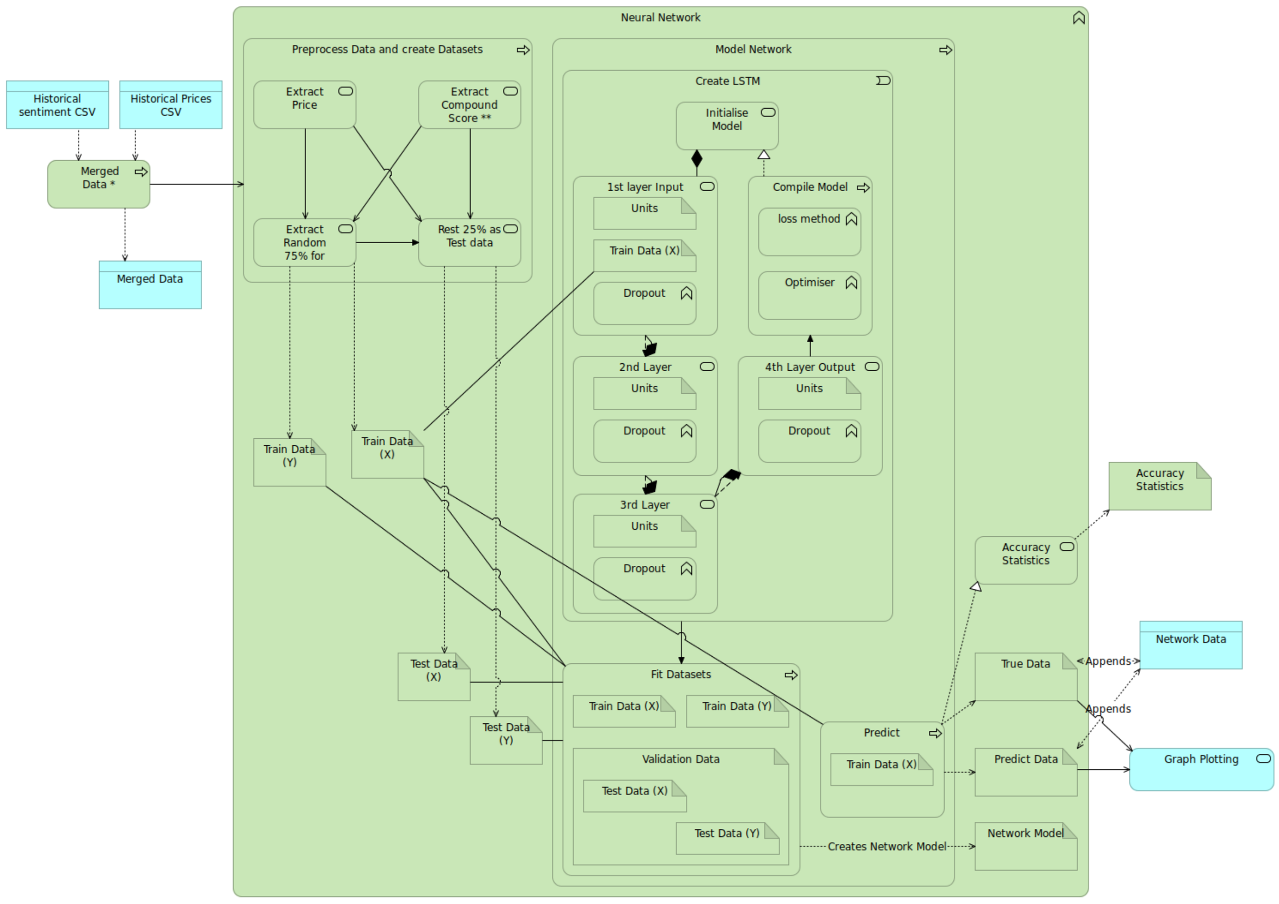


Figure 7: Neural Network layout Dataflow diagram

The dataflow diagram in *figure 7* shows the part of the system that is responsible for training and creating the neural network model. The dataflow diagram shows how the network will be trained, and the layers of a possible solution to the network. The model shows four layers which may not be the solution that will be implemented but is there to show a representation of a number of layers that could be applied.

- Merging of Datasets - Data from both historical datasets are merged to create one dataset with mapped price and sentiment for each hour. *This is a specific process that is different from the system that does not include sentiment for predictions, the merge process doesn't occur in that system/model.
- Training and Testing - Data is split into two samples of training and testing, 75:25 respectively. **This also doesn't occur in the system that doesn't model with the sentiment.

- Training network - The training sets, X and Y coordinates are used to train the network.
- Testing network - The testing sets, X and Y coordinates of 25% of the initial data are used to test the validation and accuracy of predictions as these contain the true data of what the predictions should be.
- Outputs - Accuracy Statistics, true price data and predicted next hour prices are outputted to respective files for use on the front-end application. The model is then later used for hourly forecasting.

Future Price Forecasting

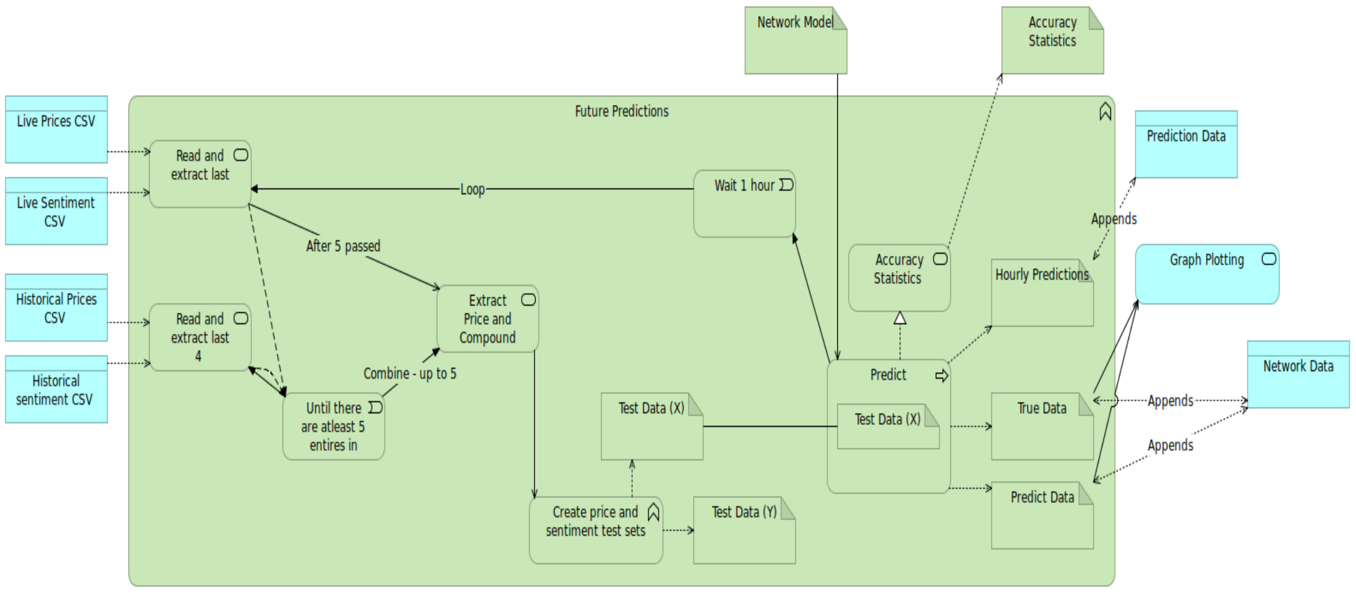


Figure 8: Price Forecasting Dataflow diagram

The dataflow diagram in *figure 8* shows how the forecasting system would be implemented. This dataflow shows how it will read live data of both sentiment and price data, merge, split and conduct regression using the trained neural network model to predict the next hour price.

- Data merging - (Doesn't occur with the system that doesn't include sentiment in price predictions). Data is consolidated from both historical and live data up to 5 iterations. This is due to after the initial hour there will only be a singular record of price and sentiment data, in which no prediction could be made from this as there isn't a sufficient amount of data.
- Prediction - This data is then fitted to the neural network model and predictions for the next time-step hour are made.

- Hour Loop - This will then proceed to loop every hour to make the hourly predictions. Historical price data will cease to be used when there are 5 or more live price records.
- Outputs - Accuracy Statistics, true price data and predicted next hour prices are outputted to respective files for use on the front-end application for charting.

Front-end Application

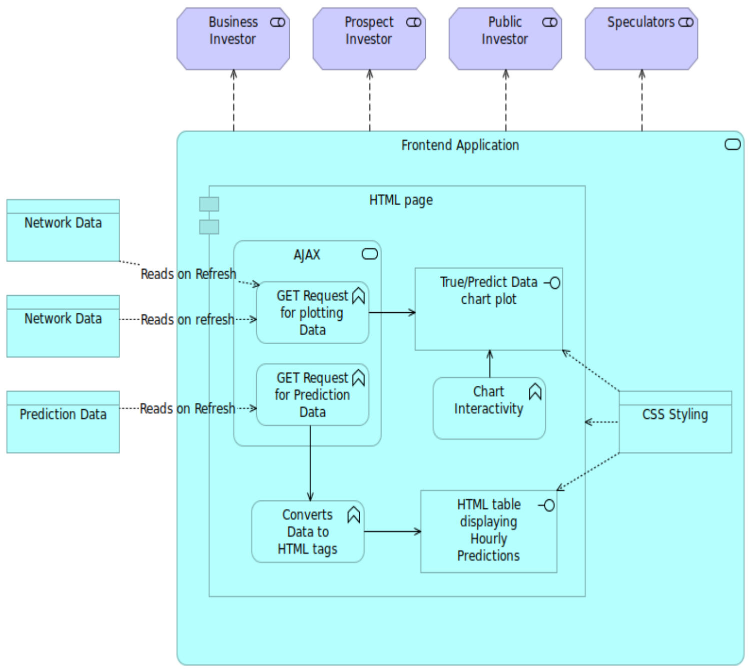


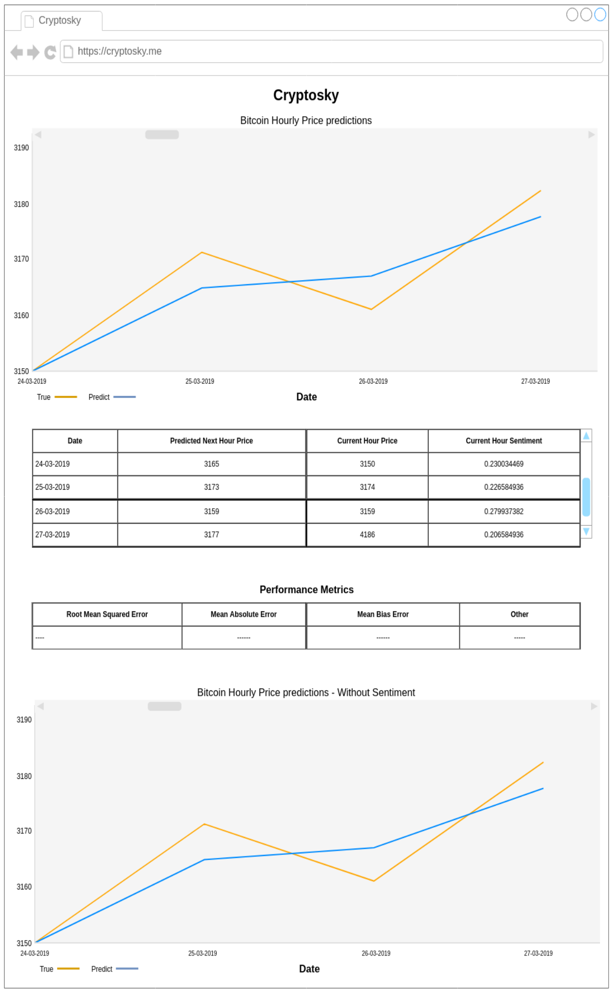
Figure 9: Front-end Application Dataflow diagram

The above dataflow diagram shows the data flow for the front-end application and how the data is read into the system from the data files generated by the backend application (Neural network).

- Ajax Requests - These are API file requests for files hosted on the server in which the system is running on. This loads the data files into the application for use.
- CSS Styling - Contains design styling for page and charts, loaded upon loading of a webpage.
- Charting and Tables - Accesses the loaded data from the Ajax requests and plots the data. Prediction data, only with sentiment and prices are plotted into a table. There will be separate charts and tables displaying the data from the backend that hasn't used sentiment in predictions to aid in establishing a correlation between sentiment and price and whether it affects the hourly price (Aiming to solve the problem statement)

- Stakeholders - There will be the four stakeholders, outlined in the problem articulation section, that would be the primary users of this application.

9.2 Interface Design



Cryptosky

Bitcoin Hourly Price predictions

Date	Predicted Next Hour Price	Current Hour Price	Current Hour Sentiment
24-03-2019	3165	3150	0.230034469
25-03-2019	3173	3174	0.226584936
26-03-2019	3159	3159	0.279937382
27-03-2019	3177	4186	0.206584936

Performance Metrics

Root Mean Squared Error	Mean Absolute Error	Mean Bias Error	Other
----	------	------	-----

Bitcoin Hourly Price predictions - Without Sentiment

Figure 10: Interface design

Figure 10 above shows the basic idea of the interface design that will be presented to the stakeholders and aims to be the interface that these stakeholders will use to aid in their market decisions of Bitcoin. The interface, although simplistic, provides all the necessary information that any of these stakeholders would need, it also provides information to allow visual comparison on how sentiment affects the hourly price of Bitcoin, represented as the two charts. The comparison will aid in solving the problem statement later in the conclusion of the project.

10 Implementation

This section will outline the method and process of development of this system to satisfy the chosen solution, technical specification and the problem statement. Each section of the system will be outlined and discussed with relevant codes snippets of essential methods from the system to highlight the processing of data throughout.

10.1 Data collection

10.1.1 Price Time-Series Historical Data

Historical price data were extracted from a CSV historical price tracker, *Bitcoin Charts* [37]. This tracker provided the historical data from the three exchanges used for Live price collection - Coinbase, Bitfinex and Gemini, since the exchanges supported the cryptocurrency. The data used spans from *2018-01-06* to *2019-01-06*.

```python
...
coinbase = pd.read_csv('coinbase_btcusd.csv')
bitfinex = pd.read_csv('bitfinex_btcusd.csv')
gemini = pd.read_csv('gemini_btcusd.csv')

coinbase.drop(columns=["Currency", "24h Open (USD)", "24h High (USD)", "
    24h Low (USD)"], axis=1, inplace=True)

coinbase.columns = ["timestamp", "price"]

coinbase['timestamp'] = pd.to_datetime(coinbase['timestamp'])

coinbase = coinbase.set_index('timestamp').resample('1D').mean().resample
    ('1H').mean()
... # similar code for the other 2 exchanges

data.set_index(coinbase['timestamp'])
for i in data:
  data['price'] = (coinbase['price'][i] + gemini['price'][i] + bitfinex['
    price'][i])/3

data = data.fillna(method='backfill')
data = data.round(3)

```

Listing 1: Historical price collection and averaging per exchange

Due to each of the hourly prices in each CSV for each exchange were averaged from the *'high'*, *'mid'* and *low* prices, the data from each exchange only needed to be averaged together. This data is averaged and then saved to a CSV containing historical prices of Bitcoin for the past year.

10.1.2 Price Time-Series Live Data

Live price data, as described in the solution approach, were extracted every hour from three exchanges - Coinbase, Bitfinex and Gemini were chosen for providing this data due to being the most popular exchange platforms that provide an API for retrieving live price data.

Key packages used:

```python
import requests

from coinbase.wallet.client import Client

from dotenv import load_dotenv
from pathlib import Path
env_path = Path('.')/'data_collector/prices/config/coinbase.env'
load_dotenv(dotenv_path=env_path)

```

Requests was used to make the API endpoint calls to obtain the response that contained the three prices for the hour needed.

The ***Coinbase*** package was mandatory for establishing a connection with the Coinbase API, and regardless this exchange was still used as it is regarded as the most popular exchange to the general public with one of the highest flow of traffic through the site to purchase cryptocurrencies.

Both the ***dotenv*** and ***pathlib*** packages were used to extract the API keys - access and secret keys, from the relevant *'.env'* file used alongside the Coinbase package for connection to the Coinbase API.

The *'high'*, *'mid'* and *low* prices were extracted from the endpoint response and averaged to provide an overall hourly price per exchange.

```python
def coinbase():

  api_key = keys().api_key
  api_secret = keys().api_secret

  try:
    client = Client(api_key, api_secret)
    repsonse = client.get_spot_price(currency_pair = 'BTC-USD')
    price = (float(repsonse['amount']))
    price = round(price, 3)
    return price
  except KeyError as e:
    print("Error: %s" % str(e))
    sys.stdout.flush()
    price = 0
    return price

```

```
def bitfinex():

  try:
    response = requests.request("GET", "https://api.bitfinex.com/v1/
   pubticker/btcusd")
    response = json.loads(response.text)

    price = (float(response['low'])+ float(response['mid']) + float(
   response['high']))/3
    price = round(price, 3)
    return price
  except KeyError as e:
    print("Error: %s" % str(e))
    sys.stdout.flush()
    price = 0
    return price

def gemini():
  ... # Exact code to bitfinex()

```

Listing 2: Extraction of Price from exchanges

The above code shows how this was implemented as a system for the price extraction from the APIs.

These functions are called every hour by a master function which uses the averaged price from each exchange to average and creates a fair, unbiased hourly price, which is the saved to a CSV containing the live unbiased price for the hour along with the time of creation. The below code shows how this is implemented:

```
def collector(priceCSV, fieldnames):

  now = datetime.now()

  coinbase_P = coinbase()
  bitfinex_P = bitfinex()
  gemini_P = gemini()

  if coinbase_P == 0 or bitfinex_P == 0 or gemini_P == 0:
    if coinbase_P and bitfinex_P == 0:
      averagePrice = gemini_P
      return
    elif coinbase_P and gemini_P == 0:
      averagePrice = bitfinex_P
      return
    elif bitfinex_P and gemini_P == 0:
      averagePrice = coinbase_P
      return
    averagePrice = (coinbase_P + bitfinex_P + gemini_P)/2
  else:
    averagePrice = (coinbase_P + bitfinex_P + gemini_P)/3

  averagePrice = round(averagePrice, 3)

```

Listing 3: Creation of the unbiased hourly price

10.1.3 Historical Tweet Collection

Historical tweets were obtained directly from the Twitter API through a simple Curl command for the given date range of the past year. Multiple accounts were created to obtain the amount of data needed, as detailed in the data gathering section under the solution approach. Due to the vast amount need, 5 tweets averaged per hour for the past year would require 1.2 requests per day (40320 total to get a whole year's worth), totalling 9,050,000 tweets. As this was highly unfeasible with the API access available for this project, 1 tweet per hour (25 per day, 1 request per 4 days) was obtained rather than the average, which resulted in only 92 requests needed to get the required data.

```
curl --request POST \
  --url https://api.twitter.com/1.1/tweets/search/fullarchive/boop.json \
  --header 'authorization: Bearer TOKEN' --header 'content-type:
    application/json' \
  --data '{"query": "bitcoin", "maxResults":100, "fromDate
    ":"201904050000", "toDate":"201904050200"}' -o data_collector/twitter/
    temp_hist_tweets.json \
  && python3 data_collector/twitter/sift_text.py

```

Listing 4: Sample Curl request - data saved to json and python scripted called to process data

These tweets are processed through the spam filter to detect if they were included unwanted text, cleaned and a polarity classification assigned to each for each hour. The process of how both the spam classification, pre-processing of the data and polarity classifications work will be detailed in their relevant sections of the system below.

```
import tweet_collector  ## pre-processing functions
import spam_filter      ## spam filter classification
import analysis_engine.sentiment_analysis as sentiment_analysis
## Sentiment analysis and polarity classification (symbolic link to file)

def processTweet(tweet, tweetFilter):

  now = datetime.datetime.now()

  #Data preprocessing
  removedLines = tweet_collector.utilityFuncs().fixLines(tweet)
  removedSpecialChars = tweet_collector.utilityFuncs().cleanTweet(
    removedLines)
  removedSpacing = tweet_collector.utilityFuncs().removeSpacing(removedSpecialChars[0])
  tweetLength = tweet_collector.utilityFuncs().checkLength(removedSpacing)

  if tweetLength == True:
  ## Drop tweet if too short

    ##Check if the tweet is predominantly English
    checkIfEnglish = tweet_collector.utilityFuncs().detectLaguage(removedSpecialChars[0])


    if checkIfEnglish == True:
      ## Remove non-English Characters
      tweetText = tweet_collector.utilityFuncs().remove_non_ascii(removedSpacing)
      print("Cleaned Tweet: ", tweetText)
      sys.stdout.flush()

      cleanedTweet = tweetText+' '+removedSpecialChars[1]

      ## Check with spam filter - drop if classified as spam
      classification = tweetFilter.testTweet(cleanedTweet)

      if classification == False:
        ## Perform Sentiment Analysis
        ovSentiment, compound = analyser.get_vader_sentiment(cleanedTweet)

        try:
          ## Save to historical tweets file
          with open('data_collector/historical_tweets.csv', mode='a') as csv_file:
            writer = csv.DictWriter(csv_file, fieldnames=['created_at', 'tweet', 'sentiment', 'compound'])
            writer.writerow({'created_at': now.strftime("%Y-%m-%d %H:%M"), 'tweet': cleanedTweet, 'sentiment': ovSentiment, 'compound': compound})
            return True
        except BaseException as exception:
          print("Error: %s" % str(exception))
          sys.stdout.flush()
          return False
      else:
  .... # other finished else statements with print statements

```

Listing 5: Sift-text python script - used alongside Curl command in Listing 4

As detailed in the comments for the code, this function conducts multiple methods on the data, all of which are predefined in other files. These are not redefined in this function to reduce code duplication throughout the system and hence are imported at the beginning of the file. Due to the nature of spam filtering tweets were inevitably removed; therefore a few hours were missing data. This resolved by making another request for that specific hour and averaging the sentiment for the given hour to fill missing data.

10.1.4 Live Tweet Collection

Live tweets were obtained through the use of the Tweepy package to stream current tweets per hour from the Twitter API. Spam filter detection,, data pre-processing and language detection are also conducted on this data and are defined within this python script *'tweet_collector.py'*, these functions will be described in the relevant sections in Data processing section.

When this script, *'tweet_collector.py'*, is ran it firstly initialises the CSV files for storing tweets and tweets that have been assigned polarities by the VADER. More importantly it initialises the spam filter and trains it based on the pre-labelled spam dataset.

```
## In __main__ when script is first ran
...

 tweetFilter = filterSpam(training_set)
 tweetFilter.trainFilter()
 ## Initialise with loaded training_set and train

 prediction = tweetFilter.testData_Prediction()
 # test classification model with test tweets

 tweetFilter.filterStatistics(prediction)
 # Print metric accuracys for test data

 tweetFilter.testPrediction()
 # Test classifier with hard specified tweets - to check if it correctly
    classifies

```

Listing 6: Spam filter initialisation and training functions

Said functions relate to a function defined under the *filterSpam* class which are used to create the training and test datasets. This function will be described in the Spam Filtering section below.

The streaming of tweets are handled by the Tweepy package and is first initialised upon starting of the python script. The streaming method works by establishing a listener and authenticated with the Twitter API; it then listens on that connection for data. This streamer can also filter on language and a specified hashtag which is loaded from a *'.env'* file also containing the API keys for authentication.

```python
...# in __main__ #Code ran first on script run
  twitter_streamer = Streamer()
  twitter_streamer.stream_tweets(tweets_file, temp_tweets, hashtag,
   tweetFilter, analyser)

#==========================================
  class Streamer():

    def __init__(self):
      pass
      # Initialise stream object

    def stream_tweets(self, tweets_file, temp_tweets, hashtag,
   tweetFilter, analyser):
      listener = Listener(tweets_file, temp_tweets, tweetFilter, analyser
   )
      auth = OAuthHandler(keys().api_key, keys().api_secret)
      # Load API keys from env file and set auth

      print("Console: ", "Authorising with twitter API")
      sys.stdout.flush()

      auth.set_access_token(keys().access_token, keys().access_secret)
      # Set access keys

      print("Console: ", "Streaming Tweets")
      sys.stdout.flush()

      stream = Stream(auth, listener, tweet_mode='extended')
      stream.filter(languages=["en"], track=hashtag)
      ## Execute streamer and filter for only English region tweets and
   by specified hashtag ('Bitcoin')

```

Listing 7: Tweepy Streamer setup

Once the listener and streamer are declared, and Tweepy begins listening all data is processed through the *on_data* method. In this function, the tweet is extracted from the response and performs data pre-processing, language detection, spam classification and sentiment analysis on the data. Additionally, there is an initial time interval that checks for a time limit - this is used to ensure that the script runs for just under an hour and restarts every hour. This allows the average of the gathered tweets' sentiment to be summed for that hour and then used for the network price predictions.

The tweet text can be nested in multiple attributes in the response; this depends on a few factors of what the tweet is and how it was posted on Twitter. If a user retweeted the tweet, the text of the tweet would be nested under *'retweeted_status'* in the JSON response, also there is a check to see if the tweets are above the original twitter tweet character limit (140 characters). This is a possible legacy parameter in the Twitter API but is checked upon data response. If an attribute *'extended_tweet'* exists the character limit for the tweet exceeds 140 but is under the 280 characters hard limit of Twitter, this exact filtering is the same if it in a non-retweeted tweet.

```python
import spam_filter
import analysis_engine.sentiment_analysis as sentiment_analysis
from tweepy import OAuthHandler
from tweepy import Stream
from tweepy.streaming import StreamListener
import csv
...
def on_data(self, data):
   ## Check time limit for under an hour - if limit reached kill script
    if (time.time() - self.start_time) < self.limit:

      now = datetime.now() + timedelta(hours=1)
      ## Sets current time, add 1 hour due to script finished before the
     completed hour is finished

      data = json.loads(data)

      # Tweet Extraction from response
      try:
        # Check if tweet is a retweet
        if 'retweeted_status' in data:
          if 'extended_tweet' in data['retweeted_status']:
          #if tweet is over the 140 word limit
            text = data['retweeted_status']['extended_tweet']['full_text']
            print("Uncleaned Tweet:", text)
            sys.stdout.flush()
          else:
            text = data['retweeted_status']['text']
            print("Uncleaned Tweet:", text)
            sys.stdout.flush()
        else:
          # Else if a normal Tweet
          if 'extended_tweet' in data:
            # If tweet is over 140 word limit
            text = data['extended_tweet']['full_text']
            print("Uncleaned Tweet:", text)
            sys.stdout.flush()
          else:
            # Else if not found in nested attributes look in top-level
            text = data['text']
            print("Uncleaned Tweet: ", text)
            sys.stdout.flush()

        # Data cleaning and pre-processing prior to polarity classification
        removedLines = utilityFuncs().fixLines(text)
        removedSpecialChars = utilityFuncs().cleanTweet(removedLines)
        removedSpacing = utilityFuncs().removeSpacing(removedSpecialChars
     [0])

    tweetLength = utilityFuncs().checkLength(removedSpacing)

    # Check if tweet is long enough to perform polarity classification on (> 5 words (checked through tokenisation))
    if tweetLength == True:
        checkIfEnglish = utilityFuncs().detectLaguage(removedSpecialChars[0])
        # Check if the text in tweet is predominantly English, if not drop
        if checkIfEnglish == True:
            tweetText = utilityFuncs().remove_non_ascii(removedSpacing)
            print("Cleaned Tweet: ", tweetText)
            sys.stdout.flush()

            # re-combine emojis onto end of tweet (Due to VADER supporting emoticon sentiment assignment)
            cleanedTweet = tweetText+' '+removedSpecialChars[1]

            ## Check if spam, drop if classified as such
            classification = self.tweetFilter.testTweet(cleanedTweet)

            if classification == False:
                ## Perform Sentiment Analysis using VADER
                ovSentiment, compound = self.analyser.get_vader_sentiment(cleanedTweet)

                # Save date/hour, tweet text, highest sentiment score from Positive or Negative and compound score
                try:
                    # temp file which is used at end of hour streaming to average sentiment for hour
                    with open(temp_tweets, mode='a') as csv_file:
                        writer = csv.DictWriter(csv_file, fieldnames=temp_fieldnames)
                        writer.writerow({'created_at': now.strftime("%Y-%m-%d %H:%M:%S"), 'tweet': cleanedTweet, 'sentiment': ovSentiment, 'compound': compound})
                except BaseException as exception:
                        print("1 Error: %s" % str(exception))
                        sys.stdout.flush()

                # Save date/hour, tweet text, highest sentiment score from Positive or Negative and compound score
                try:
                    # tweet file for storing all collected tweets from every hour
                    with open(tweets_file, mode='a') as csv_file:
                        writer = csv.DictWriter(csv_file, fieldnames=fieldnames_tweet)
                        writer.writerow({'created_at': now.strftime("%Y-%m-%d %H
:%M:%S"), 'tweet': cleanedTweet, 'sentiment': ovSentiment, 'compound': compound})
            except BaseException as exception:
              print("2 Error: %s" % str(exception))
              sys.stdout.flush()
          else:
            print("Console: ", "Tweet is spam. Not storing tweet in dataset")
            sys.stdout.flush()
        ...
      ... # Closing Else statments with print statments for when the tweet doesn't meet criteria
    ...

```

Listing 8: Tweepy Stream: 'on_data' method

As for the key facts about this function; the length of the tweet is checked to be above 5 (tokenised) due to any tweets with fewer words will not contain enough information to be given a proper polarity classification and almost always returns as 100% neutral, which is of no use and will have no effect on the hours average sentiment. The entire code in the function is encapsulated in a try-catch to check if data was received and handles non-responses and missing data. If there was no data the issue is ignored unless a connection between the streamer and API is broken it otherwise exits the script.

10.2 Data pre-processing

Various techniques and tools have been utilised throughout the development of the system to process the data appropriately so it can be parsed by VADER, spam filter and neural network. This section will cover the crucial functions that provide such functionalities and that are called throughout the system, as seen in some of the above code snippets.

10.2.1 Tweet Filtering

Various 'Utility Functions' have been used to initially filter out unwanted data from tweet text. These functions called by, both live tweet (*tweet_collector.py*) and historical tweet (*sift_text.py*) processing, prior any polarity classification or storing of tweet data to CSV files.

```
import re
import emoji as ji
## Key packages used
...
class utilityFuncs():

  def cleanTweet(self, text):
    # Function to clean tweets, removes links and special characters
    return re.sub(r'([^0-9A-Za-z \-\%\  \$ \t])|(@[A-Za-z0-9]+)|(http\S+)', '', text), ' '.join(c for c in text if c in ji.UNICODE_EMOJI)
    # Also removes emojis from text - later re-added due to VADER supporting emoticons

  def removeSpacing(self, text):
    return re.sub(r'( +)', ' ', text)
    # Removes extra spacing that may be left between words

  def fixLines(self, text):
    return re.sub(r"([\r\n])", " ", text)
    # Removes line breaks and new lines from text

  def remove_non_ascii(self, text):
    return ''.join(i for i in text if ord(i)<128)
    # User after language detections and removes non-English characters from text

  def checkLength(self, text):
    tokens = text.split()
    if len(tokens) <= 5:  # Tokenisation
      return False
    else:
      return True

```

Listing 9: Basic data filtering and processing function - defined in 'tweet_collector.py'

Due to VADER being a lexicon-based sentiment analyser little data pre-processing needs conducting on the tweet text. The functions above primarily remove unnecessary text from the tweet that will either provide no insight into public opinion or can obstruct a proper classification of the sentiment - such as the existence of URLs in the given text. Additionally, the 'clean_tweet' function removes the emojis in the given text if any are presently using the emoji package - which in turn is another lexicon that compares the given text to any emoticon contained within the lexicon. These are removed at this stage but are later re-added back to the text as VADER support emoticon classification. The last function in 'utility functions', 'checkLength' splits the text up into individual words (tokens - a process of tokenisation), this is used to check the total length of a tweet. If the tweet is less than five words it is dropped from classification, this is due to

10.2.2 Language detection filtering

This feature of the system is used as an additional filter for filtering out non-English tweets. As discussed in the solution approach, Tweepy/Twitter API provides a means to filter out non-English based tweets, this, however, doesn't work if the user has settings on Twitter set to be English as a prefered language and the region 'en'. Due to this non-English characters can still be within collected tweets; thus these are detected and filtered with the below function.

```
def detectLaguage(self, text):
  """
  Calculate the probability of given text is written in several languages
  Using nltk stopwords and comparing to all supported languages

  There are other ways to identify this - TextBlob.detect_language and
   Ngrams
  """
  language_ratios = {}

  # Split words up into tokens - tokenisation
  tokens = wordpunct_tokenize(text)

  # Shift to lower case
  words = [word.lower() for word in tokens]

  # Compute per language in nltk number of stopwords in text
  for language in stopwords.fileids():
    stopwords_set = set(stopwords.words(language))
    words_set = set(words)
    common_elements = words_set.intersection(stopwords_set)

    language_ratios[language] = len(common_elements)
    # Form ratio scores for each language detected from stopword
    comparison

  ratios = language_ratios
  highest_ratio = max(ratios, key=ratios.get)
 # Extract highest ratio language used in given text

  print("Console: Text is - ", highest_ratio)
  sys.stdout.flush()

  if highest_ratio == 'english':
    return True
  else:
    return False
 # If text is not predominately English drop tweet

```

Listing 10: Language detection and filter function [38]

This function uses several natural languages pre-processing techniques to identify the most predominant language for a given text. It accomplishes this by first tokenising the text into tokens and converting them to lower case - this is so that the stopwords can be identified. For each of the languages supported by the Natural Language Toolkit Python package, the stopwords are identified in the text and compared to the stopwords in the language corpus' in NLTK. The ratios for the individual languages are formed, and then the predominant language identified. If the language is not predominantly English, the tweet is dropped. There is however an issue with this approach, if a tweet contains too many special characters - characters that are allowed, the tweet occasionally is not classified as English even when it predominantly is upon visual inspection; therefore the tweet is dropped and not processed. This isn't a significant issue as about 3000 tweets can be collected in an hour, and some of these would be filtered out by the spam filter regardless.

Additionally, an n-grams method could be used to distinguish the language of a given text and may perform more accurately than the word-based approach that was implemented [39]. This could be a later improvement as the word-based approach is sufficient and requires a corpus for each language to compare against to be presented. Therefore it could be used as a comparison between approaches and seen as a possible improvement.

10.2.3 Spam filter - Tokenisation, Ngrams, Stopword removal and Stemming

Prior to any text being processed to both train the Naive Bayes classifier of the spam filter or to classify live tweets, the data needs to be pre-processed to extract the features from the text so that the classifier can identify the probability of each word in the given text. The explanation of how this classifier functions will be detailed in the 'Spam Filtering' Section.

```python
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
...

def processTweet(tweet, gram = 2):
  tweet = tweet.lower() # convert to lower case

  words = word_tokenize(tweet)    # Tokenise words in text
  words = [w for w in words if len(w) > 2]
  # remove words that are not greater than 2 characters

  if gram > 2:    ## Increasing grams can increase accuracy
    w = []
    for i in range(len(words) - gram + 1):
      w += [' '.join(words[i:i + gram])]
    return w

  # Remove stopwords
  sw = stopwords.words('english')
  words = [word for word in words if word not in sw]
  # Create new dict without stopwords

  stemmer = PorterStemmer()            # Stem words
  words = [stemmer.stem(word) for word in words]
  # Create new dict of stemmed words

  return words

```

Listing 11: pre-processing of data prior to being used by the spam filter

The actions performed on the text consist of:

- Convert to lower case: This is due to that 'DROP' and 'drop', and likewise words convay the same meaning thus these are simply converted to all lower case.
- Tokenise words: This splits the text into individual words. A dictionary is then created from the tokens that are above the length of 2 - due to words that are of less 'is', 'he' and 'if' will not contribute to spam detection and are seen as generic words in the English language
- Ngrams: This is implemented to provide richer word sequences for the spam filter classification, as explained in the litrature review use of ngrams can increase accuracy.
- Stop words Removal: This removes stopwords such as 'this', 'we' and 'now' from the text. Due to these common words carrying less importance for sentiment analysis.

- Stemming: Reduces words down to their smaller form, as in it remove suffixes from inflected words - ’studying’ become ’study’ [40]. The Porter Stemmer works by removing the suffixes from the text - ’going’ becomes ’go’, however, this applies to other words such as ’leaves’ becomes ’leav’ which is not a word. However, this method will be applied equally to all words containing such suffixes so all variations will become so, thus still allowing the probability classifications to occur on the word as all variations will be the same.

As discovered from [40], lemmatisation could be an alternative and arguably a better solution to stemming. Lemmatization works fundamentally the same as stemming but reduces the inflected words properly ensuring that a root word belongs to a language. Using the same words that are used to describe stemming, lemmatisation reduces ’goes’ to ’go’ and ’leaves’ to ’leaf’ - removing the suffixes down to create the actual root word. Although lemmatisation will provide the classifier with an actual English word, regardless stemming will still reduce the words down to the same form, this added with a lemmatiser needing a corpus for classifying the words to their root words and additional computational time to do so, the former of using a stemmer is sufficient.

10.3 Spam Filtering

This section of the implementation will describe how the spam filter is initialised in the *tweet_collector*, how it is trained and how it classifies tweets as being either spam or ham (wanted data).

Listing 12 shows the initalisation and method functions used within the *tweet_collector*, that creates the training and testing datasets, and tests classifier on hard specified tweets and checks their classification.

```python
import pandas as pd
import spam_filter
import numpy as np
...

class filterSpam(object):

  def __init__(self, training_set):
    self.training_set = training_set
    ## initialises function and globalises training set for use in every
    function where needed

  def trainFilter(self):
    self.dataset()    ## Split dataset 75:25
    self.train()    ## Train based on training dataset

  def dataset(self):
    self.data = pd.read_csv(self.training_set)

    self.data['class'] = self.data['classes'].map({'ham': 0, 'spam': 1})
    # Remap labels of 'Spam' and 'Ham' to 1:0 respectively

    self.data.drop(['classes'], axis=1, inplace=True)
    # Drop old labels

    self.trainIndex, self.testIndex = list(), list()
    for i in range(self.data.shape[0]):
      if np.random.uniform(0, 1) < 0.75:  # Random shuffle data of 75%
        self.trainIndex += [i]  # Create training index
      else:
        self.testIndex += [i] # Create testing index
    self.trainData = self.data.loc[self.trainIndex]
    self.testData  = self.data.loc[self.testIndex]
    # Define datasets by getting values from first 75% and then 25%

    self.trainData.reset_index(inplace=True)
    self.testData.reset_index(inplace=True)
    # Reset indexes

    self.trainData.drop(['index'], axis=1, inplace=True)
    self.testData.drop(['index'], axis=1, inplace=True)
    # Drop old index

def train(self):
  self.spamFilter = spam_filter.classifier(self.trainData)
  # Initialise the spam filter with the 75% dataset

  self.spamFilter.train()
  # Train

def testData_Prediction(self):
  # Classify data from test dataset
  prediction = self.spamFilter.predict(self.testData['tweet'])
  return prediction

def testPrediction(self):

  # Test Spam/Ham tweets - should return True and False respectivly
  spam = spam_filter.processTweet("Earn more than 0015 btc free No deposit No investment Free Bitcoins - Earn $65 free btc in 5 minutes bitcoin freebtc getbtc")

  ham = spam_filter.processTweet("Bitcoin closed with some gains in month of February")
  # Process Tweets - Tokenise and Stem


  hamTweet = self.spamFilter.classify(ham)
  spamTweet = self.spamFilter.classify(spam)
  # Classify both tweets

  print("Console: ", "Spam Tweet -- ", spamTweet)
  sys.stdout.flush()
  print("Console: ", "Ham Tweet -- ", hamTweet)
  sys.stdout.flush()

def filterStatistics(self, prediction):
  # Get performance metrics for prediction data compared to actual test data
  spam_filter.metrics(self.testData['class'], prediction)

def testTweet(self, tweet):
  # Used for live tweets classification
  processed = spam_filter.processTweet(tweet)
  classified = self.spamFilter.classify(processed)

  return classified

```

Listing 12: Spam filter training Class - *tweet_collector.py*

- filterSpam - __init__: is called when the *tweet_collector* script is first executed which initialises the object, first described in the 'Live Tweet Collection' section above.

- trainFilter: is a function that calls the dataset function which created the training and testing dataset, followed by the train function which trains the initialised classifier. This function's sole purpose is to serve as a parent function that only needs to be called to perform the child functions once.

- dataset: This function loads the pre-labelled spam dataset, remaps the labels to integers 0:1 to ham:spam respectively, creates a dictionary with an index of 75% of the original data for the training dataset and 25% for the testing dataset. This function does this by extracting the data at the set point from the spam dataset into the relevant new datasets which resetting indexes and dropping old columns to form appropriate data.

- train: Is used to call the classifier function defined in the *spam_filter* script and passes the training data for it to initialise then train on.

- testData_Prediction: Is a function similar to the 'train' function, but calls the 'predict' function defined in *spam_filter* to test the classifier on the test data and returns the predictions made, which is used later on in the 'filterStatistics' function to calculate the accuracy of the classifier.

- testPredictions: This function is used to test the accuracy of the trained classifier with pre-defined tweets that are assumed to be either spam or ham. The primary goal of this function is to ensure that the classifier correctly classifies the two tweets as either spam or ham appropriately. The text is processed through the 'processTweet' function previously described to transform the tweets into tokens ready for classification.

- filterStatistics: Is used by the 'testData_Predictions' function to calculate the accuracy of the classification model using the test data and prediction data. The 'metrics' function is defined in the *spam_filter* script.

- testTweet: Is a function used on the live tweets by the 'on_data' function also outlined previously to process the tweets data and classify it as either being spam or not, the 'on_data' function then handles the result accordingly.

10.3.1 Naive Bayes model

The spam filter classifier, using a Naive Bayes model, was coded from scratch. Ultimately unneeded as the Scikit-learn python package comes with four inbuilt Naive Bayes classification models (Bernoulli, Complement, Multinomial, Gaussian)[41]. The Naive Bayes model implemented was a multinomial Bayes model as the data used for classification was of multinomial distribution and categorical. This algorithm was not compared to the Scikit-learn's inbuilt model for accuracy as this was not the focus of this project. The model was coded from scratch due to finding information on how this would be done with techniques such as TFIDF and Additive Smoothing as detailed in the literature review, the tutorial that helped the greatest *Spam Classifier in Python from scratch* [34] [42]. For an explanation of how the maths work behind this classifier see Literature review sections 'Bag Of Words', 'TF-IDF' and 'Addictive Smoothing'.

```python
class classifier(object):
  def __init__(self, trainData):
    self.tweet = trainData['tweet']
    self.labels = trainData['class']

  def TF_and_IDF(self):
    noTweets = self.tweet.shape[0]
    self.spam = self.labels.value_counts()[1]
    self.ham = self.labels.value_counts()[0]
    self.total = self.spam + self.ham

    # Initialise spam vars
    self.spamCount = 0
    self.hamCount = 0
    self.tfSpam = dict()
    self.tfHam = dict()
    self.idfSpam = dict()
    self.idfHam = dict()

    # Bag Of Words implementation - pro

    for entry in range(noTweets):
      processed = processTweet(self.tweet[entry])
      count = list()
      #To keep track of whether the word has ocured in the message or not. IDF count

      for word in processed:
        if self.labels[entry]:
          self.tfSpam[word] = self.tfSpam.get(word, 0) + 1
          self.spamCount += 1
          ## If label for data is spam then add words to spam list
        else:
          self.tfHam[word] = self.tfHam.get(word, 0) + 1
          self.hamCount += 1
          # If label for data is ham then add words to ham list
        # Addictive Smoothing - if current word is not seen add count list
        if word not in count:
          count += [word]
      for word in count:
        # Loop unseen word list
        if self.labels[entry]:
          self.idfSpam[word] = self.idfSpam.get(word, 0) + 1
        else:
          self.idfHam[word] = self.idfHam.get(word, 0) + 1

  def TF_IDF(self):
    self.probSpam = dict()
    self.probHam = dict()
    self.sumSpam = 0
    self.sumHam = 0

    # Calculate probability of word being spam or ham based on occurance in text compared to counted sets along with relevant keys
    for word in self.tfSpam:
      self.probSpam[word] = (self.tfSpam[word]) * log((self.spam + self.ham) / (self.idfSpam[word] + self.idfHam.get(word, 0)))
      self.sumSpam += self.probSpam[word]

    for word in self.tfSpam:
      self.probSpam[word] = (self.probSpam[word] + 1) / (self.sumSpam + len(list(self.probSpam.keys())))

    for word in self.tfHam:
      self.probHam[word] = (self.tfHam[word]) * log((self.spam + self.ham) / (self.idfSpam.get(word, 0) + self.idfHam[word]))
      self.sumHam += self.probHam[word]
    for word in self.tfHam:
      self.probHam[word] = (self.probHam[word] + 1) / (self.sumHam + len(list(self.probHam.keys())))

    # Calculate total amount of spam words identified
    self.probSpamTotal, self.probHamTotal = self.spam / self.total, self.ham / self.total

```

Listing 13: classifer class of spam_filter.py

10.3.2 Classification

This function aims to classify the pre-processed tweet data as either spam or ham based on the term-frequency and probabilities calculated in the 'TF_IDF' function. This conducted for each word in the processed tweet is identified if the word is contained in the spam set, based on the level of occurrence the probability is assigned a weight (The more it occures, the more likely it is a generic word), this is also identified for the level of occurrence in the ham set. Totals for the probability are formed, and the total count for both spam and ham are added to the spam and ham probabilities for the processed tweet. If the spam probability *pSpam* is higher than the ham probability *pHam* based on the level of occurrence of each word in the modelled respective sets, a boolean is returned based on which probability is higher - which identifies if the tweet is predominantly spam or ham (*True* or *False*).

```python
def classify(self, processed):
  pSpam, pHam = 0, 0

  for word in processed:
    if word in self.probSpam:
      pSpam += log(self.probSpam[word])
    else:
      pSpam -= log(self.sumSpam + len(list(self.probSpam.keys())))
    if word in self.probHam:
      pHam += log(self.probHam[word])
    else:
      pHam -= log(self.sumHam + len(list(self.probHam.keys())))
    pSpam += log(self.probSpamTotal)
    pHam += log(self.probHamTotal)
  return pSpam >= pHam

```

Listing 14: Classify Function of Parent classifier class of spam_filter.py

10.3.3 Predict

The predict function under the classify parent class used by the *tweet_collector* to test the trained classifier on the test dataset. For each tweet in the dataset, the data is processed through the *processTweet* function previously described, this returns a dictionary of words in the text which then used in the *classify* function described above to identify whether or not each tweet is predominantly spam or ham, the result of all tweets are returned. The *tweet_collector then uses the returned array* in the *filterStatistics* function, also previously described, to calculate the performance and accuracy of the trained model.

```python
def predict(self, testData):
  result = dict()
  for (i, tweet) in enumerate(testData):
    processed = processTweet(tweet)
    result[i] = int(self.classify(processed))
  return result

```

Listing 15: Predict function of parent classifier class of spam_filter.py

10.3.4 Metrics

The metrics function calculates the F-score, precision, recall and accuracy (Suitable performance metrics for classification models) of the model when comparing the predicted class labels to the real class labels of the test dataset used in testing the model.

By using these metrics, the performance of the model can be evaluated and later compared to a competitor model - for this reason, the metrics are calculated. A discussion of what these metrics show and the level of accuracy of the model are discussed in the Testing section later.

```python
def metrics(labels, predictions):
  true_pos, true_neg, false_pos, false_neg = 0, 0, 0, 0

  # Identify the true pos/negs and false pos/negs of the predicted model of predicted values compared to the actual true values of the test dataset class labels
  for i in range(len(labels)):
    true_pos += int(labels[i] == 1 and predictions[i] == 1)
    true_neg += int(labels[i] == 0 and predictions[i] == 0)
    false_pos += int(labels[i] == 0 and predictions[i] == 1)
    false_neg += int(labels[i] == 1 and predictions[i] == 0)
  precision = true_pos / (true_pos + false_pos)
  recall = true_pos / (true_pos + false_neg)
  Fscore = 2 * precision * recall / (precision + recall)
  accuracy = (true_pos + true_neg) / (true_pos + true_neg + false_pos + false_neg)

  print("Precision: ", precision)
  print("Recall: ", recall)
  print("F-score: ", Fscore)
  print("Accuracy: ", accuracy)

```

Listing 16: Metrics function for calculating the performance and accuracy of the model

10.4 Sentiment Analysis

This section of the implementation outlines how VADER sentiment analyser is implemented and performs with the rest of the system. The *get_sentiment* class and its *__init__* function are called in the *tweet_collector* script upon starting and by the *historical tweets* script to initialise the analyser from the VADER package. Both scripts then call *get_vader_sentiment* when needed to give polarity classification to a tweet.

```python
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
...
class get_sentiment(object):

  def __init__(self):
    ## Initialse object and analyser as global objects
    self.analyser = SentimentIntensityAnalyzer()
    self.sentiment = {}
    self.compound = {}

  def get_vader_sentiment(self, sentence):

    # Calculate the polarity scores of the provided tweet
    score = self.analyser.polarity_scores(sentence)

    # Split dict into overall sentiment and compound
    sentiment = list(score.values())
    compound = sentiment[3:]
    compound = compound[0]
    sentiment = sentiment[:3]

    # Compare and find overall sentiment
    score = max(sentiment)
    pos = [i for i, j in enumerate(sentiment) if j == score]

    if pos[0] == 1:
      print("Console: ", "Tweet is overal Neutral - Score: ", score)
      # return neg or pos which ever is higher
      if sentiment[0] > sentiment[2]:
        score = sentiment[0]
      else:
        score = sentiment[2]
      return score, compound
    else:
    return score, compound

```

The *get_vader_sentiment* function provides the polarity scores for the provided tweet. The scores are split into polarity and compound to compare the positive and negative scores to identify the overall greater sentiment in the given tweet. By doing so helps to identify if the tweet was overall negative or positive. The compound score is separated and used separately.

10.5 Recurrent Neural Network - LSTM

This section of the implementation describes and discusses how the LSTM neural network is configured, trained, tested and used to create the model later used for price forecasting for both neural networks - with and without hourly sentiment embedded in datasets. Its performance metrics that were calculated to verify the accuracy for the model, appropriate to regression models and K-fold validation implementation are discussed.

Packages used for both neural networks, with and without hourly sentiment embedded with price data:

```python
import pandas as pd
import numpy as np
from math import sqrt
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import mean_squared_error, classification_report
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout
import keras.backend as K
from sklearn.model_selection import StratifiedKFold, cross_val_score

from time import sleep
from datetime import datetime, timedelta

import csv, sys, json

from tqdm import tqdm
from keras_tqdm import TQDMCallback

```

Listing 17: LSTM packages

Additionally, this section only shows code from the neural network that has the sentiment embedded in the datasets; comments are made in the code snippets with the reduced code that consists of the neural network and is due to each neural network having almost the same code. The reasons behind not implementing both networks in the same Python script was down to perform. Due to Python executing code synchronously and due the neural networks needing to ran on the dot of an hour and at the same time the code was divided and executed individually. This also reduced the need to recode most of the functions to loop and perform tasks for each network at every given stage of the network even if the majority of the code was duplicated.

10.5.1 Script Execution

```
import pandas as pd
import csv
...

class Network(object):

  def __init__(self, merged_lstm_data):
    self.lstm_data = merged_lstm_data

...
# Merges hourly and price data into one CSV
### Occurs for both neural networks but only specific columns are used in
    each
merged = pd.merge(left=price_file, right=tweet_file, how="inner")
print("merge length", len(merged))
merged.to_csv('merged_lstm_data.csv')

# Initialise network and pass merged data
network = Network(merged)

# Calls the dataset creation function from the network class
network.data()

# Calls the future trading function which starts forecasting the price
    for next hour
network.future_trading(live_price, live_sentiment, predictions_file)


```

Listing 18: Start of execution of the LSTM script

10.5.2 Dataset Creation

Listing 19 shows how the datasets for training (train_X and train_Y) and testing (test_X and test_Y) are formed and shaped for model training. A look back of 2 is used to create a timestep of one record to ensure predictions are forecasted for the next record. Prices are also scaled between 0 and 1 due to sentiment ranging in the same values and is a standard for model creation to speed up regression and model training as the data is of smaller values - using the scikit-learns MinMaxScaler function.

```
def data(self):
  self.preprocess()
  # Pre-process and extract required data from dataset

  loopback = 2
  # Set lookback for dataset creation - for 1 record timestep

  train_X, train_Y = self.create_sets(self.price_train, loopback, self.sentiment_data[0:self.price_train_size])
  test_X, test_Y = self.create_sets(self.price_test, loopback, self.sentiment_data[self.price_train_size:len(self.scaledPrice)])
  ## Create datasets (!sentiment parameters are not passed into the neural network the doesn't embedded the sentiment alongside price data!)

  train_X = np.reshape(train_X, (train_X.shape[0], 1, train_X.shape[1]))
  test_X = np.reshape(test_X, (test_X.shape[0], 1, test_X.shape[1]))

  self.model_network(train_X, train_Y, test_X, test_Y)
  # Call network function to train network

def preprocess(self):

  self.model_data = self.lstm_data[['price','compound']].groupby(self.lstm_data['created_at']).mean()
  #Extract price and compound columns from dataset

  self.sentiment_data = self.model_data['compound'].values.reshape(-1,1)
  self.price_data = self.model_data['price'].values.reshape(-1,1)
  ## Reshape data to column-wise

  # convert types to float32 for consistancy
  self.sentiment_data = self.sentiment_data.astype('float32')
  self.price_data = self.price_data.astype('float32')

  self.scale = MinMaxScaler(feature_range=(0,1))
  self.scaledPrice = self.scale.fit_transform(self.price_data)
  # Scale price to values between 0 and 1

  self.price_train_size = int(len(self.scaledPrice) * 0.7 )
  # use 70% of dataset for training and 30% for testing
  self.price_test_size = len(self.scaledPrice) - self.price_train_size

  # Get said train data on size

  self.price_train = self.scaledPrice[0:self.price_train_size:]
  self.price_test = self.scaledPrice[self.price_train_size:len(self.scaledPrice):]
  #set sizes of dataset to be mapped later


def create_sets(self, data, lookback, sentiment):
  data_X, data_Y = [], []
  for i in range(len(data) - lookback):
    if i >= lookback:
      # Sets timestep if data by a record
      pos = data[i-lookback:i+1, 0]
      pos = pos.tolist()

      # Append sentiment at position of hours price
      pos.append(sentiment[i].tolist()[0])
        ## Above pos is not conducted on neural network with no sentiment
    embedded pos.append(0) occurs instead
      data_X.append(pos)
      data_Y.append(data[i + lookback, 0])
  return np.array(data_X), np.array(data_Y)

```

Listing 19: Dataset creation and preprocessing

10.5.3 Training and Testing Model

The neural network is set up with four layers each of which configured with 100 LSTM cells, with a dropout of 0.2 each, and returning sequences to each other layer. A dropout was used to ensure that the data would not be overfitted, by setting the dropout to 0.2 probability, 80% of the data on each layer would be retained for the next layer. Return sequences allow for the returning of the hidden state output for each time step and ensures the next LSTM layer has 2 inputs that carry over from the previous layer, which are the old weights and value outputs from the previous layer.

```
self.model = Sequential()

## 1st layer - input layer
self.model.add(LSTM(100, input_shape=(train_X.shape[1], train_X.shape
  [2]), return_sequences=True))
self.model.add(Dropout(0.2))

## 2nd Layer
self.model.add(LSTM(100, return_sequences=True))
self.model.add(Dropout(0.2))

## 3rd Layer
self.model.add(LSTM(100, return_sequences=True))
self.model.add(Dropout(0.2))

## 4th Layer without sequences
self.model.add(LSTM(100))
self.model.add(Dropout(0.2))

self.model.add(Dense(1))
self.model.compile(loss='mean_squared_error', optimizer='adam', metrics
  =['mse', 'mae', 'mape'])

self.model.summary()
# Model summary of params and dropout at each layer

```

```
    history = self.model.fit(train_X, train_Y, epochs=200, batch_size=1000,
        validation_data=(test_X, test_Y), verbose=0, shuffle=False, callbacks
        =[TQDMCallback()])

    yhat = self.model.predict(test_X)

    scale = self.scale
    scaledPrice = self.scaledPrice

    yhat_inverse_sent = scale.inverse_transform(yhat.reshape(-1, 1))
    testY_inverse_sent = scale.inverse_transform(test_Y.reshape(-1, 1))

```

Listing 20: LSTM model creation layering compiling and fitting

As per the discussion in the literature review, the Adam optimiser was used for the compilation of the model. The loss was calculated using the mean squared error and the metrics calculated and returned to present the accuracy of the model in prediction were: mean squared error, root mean squared error, mean absolute error and mean absolute percentage error. Both the metrics and prediction made are saved to a CSV that is then presented to users in the server-hosted UI.

The model was fitted on the training sets (X, Y) over 200 epochs with a batch size of 1000 on about 11000 records - which was the total amount of data used to train for a year. Predictions are then made using the test set resulting in the predictions of 'yhat' which is inverted and rescaled to get original price values to save to a CSV and also displayed on the user interface.

10.6 Future Prediction Forecasting

```
def remodel(self, price_file, previous_sent, live_price, live_sentiment,
    predictions_file):
    price = pd.read_csv(live_price)
    sentiment = pd.read_csv(live_sentiment)

    price_tail = price.tail(5)
    sentiment_tail = sentiment.tail(5)
   ## Get last 5 live prices and predict on them

    price_tail.index = price_tail['created_at']
    sentiment_tail.index = sentiment_tail['created_at']

   ## Example gets last 5 records for some reason

    price = price_tail['price'].values.reshape(-1,1)
    sentiment = sentiment_tail['compound'].values.reshape(-1,1)

    price_scale = self.scale.fit_transform(price)

```

```
testX, testY = self.create_sets(price_scale, 2, sentiment)

testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

yhat = self.model.predict(testX)
yhat_inverse = self.scale.inverse_transform(yhat.reshape(-1, 1))
testY_inverse = self.scale.inverse_transform(testY.reshape(-1, 1))

rmse_sent = sqrt(mean_squared_error(testY_inverse, yhat_inverse))
print('Test RMSE: %.3f' % rmse_sent)
## Calculate RMSE for prediction

difference = ((yhat_inverse[0][0]-self.previous_val)/self.previous_val)
 *100
# Caclulate difference between hour predictions for threshold action
 prediction (below)

...
## Suggest market action based on 0.25 threshold (2.5%)

if difference >= self.threshold:
  print("Buy")
  self.state = 'BUY'
elif difference < self.threshold:
  print("Sell")
  self.state = 'SELL'
else:
  print("Prediction Error!")

...

self.test_Y_updating = np.concatenate((self.test_Y_updating,
 testY_inverse))
self.yhat_updating = np.concatenate((self.yhat_updating, yhat_inverse))

## Output plots to json for display on UI
...
with open('data/updating.json', mode='w') as file:
  for x in range(len(cat)):
    xs[x] = {'index' : x, 'testY_inverse': cat[x][0], 'yhat_inverse' :
 cat[x][1]}
  json.dump(xs, file, indent=3)

...

...
## Output prediction made for hour to CSV file for use in UI
try:
  with open(predictions_file, mode='a') as csv_file:
    ...
    writer.writerow({'created_at': now.strftime("%Y-%m-%d %H:00:00"), '
 next_hour_price': hour, 'current_price': current, 'current_sentiment':
  senti, 'state': self.state})
except Exception as e:
  print("Error: %s" % str(e))
  sys.stdout.flush()

# Set the predicted value of current hour
self.previous_val = yhat_inverse[0][0] ##THE NEXT PREDICTED VALUE IN AN
  HOUR

```

10.7 User Interface

11 Testing Metrics and Accuracy

mean bias Error

k-fold cross validation was attempted, but issues with continuous data

12 Project Evaluation

Reflection

Quality

13 Discussion: Contribution and Reflection

13.1 Limitations

14 Conclusion and Future Improvements

14.1 Conclusion

14.2 Future Improvements

Shifting the intial data by and hour and sequencing over previous data - will also allow proper use of look-back windows

Another could be to predict the hour of sentiment and create a threshold for it.

Identify whether or not use of ngrams improved accuracy of spam classification

Identify whether use lemmatisation would change how spam classification occured

References

[1] V. S. Pagolu, K. N. Reddy, G. Panda, and B. Majhi, "Sentiment analysis of twitter data for predicting stock market movements," in *2016 international conference on signal processing, communication, power and embedded system (SCOPES)*, IEEE, 2016, pp. 1345–1350. [Online]. Available: https://arxiv.org/pdf/1610.09225.pdf.

[2] N. Indera, I. Yassin, A. Zabidi, and Z. Rizman, "Non-linear autoregressive with exogeneous input (narx) bitcoin price prediction model using pso-optimized parameters and moving average technical indicators," in *Journal of Fundamental and Applied Sciences. Vol.35, No.35*, University of El Oued, 2017, pp. 791–808. [Online]. Available: https://www.ajol.info/index.php/jfas/article/viewFile/165614/155073.

[3] J. L. Evita Stenqvist, "Predicting bitcoin price fluctuation with twitter sentiment analysis," Diva, 2017. [Online]. Available: http://www.diva-portal.org/smash/get/diva2:1110776/FULLTEXT01.pdf.

[4] O. G. Yalcin, "Predict tomorrows bitcoin (btc) price with recurrent neural networks," Towards Data Science, 2018. [Online]. Available: https://towardsdatascience.com/using-recurrent-neural-networks-to-predict-bitcoin-btc-prices-c4ff70f9f3e4.

[5] Intel-Corporation, "Stock predictions through news sentiment analysis," Code Project, 2017. [Online]. Available: https://www.codeproject.com/Articles/1201444/Stock-Predictions-through-News-Sentiment-Analysis.

[6] S. C. Sean McNally Jason Roche, "Predicting the price of bitcoin using machine learning," in *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, IEEE, 2018, pp. 344–347. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8374483.

[7] Twitter, "Search tweets," Twitter Developers, 2018. [Online]. Available: https://developer.twitter.com/en/docs/tweets/search/overview.

[8] ——, "Consuming streaming data," Twitter Developers, 2018. [Online]. Available: https://developer.twitter.com/en/docs/tutorials/consuming-streaming-data.html.

[9] J. Roesslein, "Streaming with tweepy," Tweepy, 2009. [Online]. Available: http://docs.tweepy.org/en/v3.4.0/streaming_how_to.html.

[10] S. N. Mehrnoush Shamsfard, "Using linked data for polarity classification of patients experiences," in *Journal of Biomedical Informatics*, Elsevier, 2015, pp. 6–19. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1532046415001276.

[11] L. P. T. Chedia Dhaoui Cynthia M. Webster, "Social media sentiment analysis: Lexicon versus machine learning," in *Journal of Consumer Marketing, Volume 34. Issue 6*, Emerald Insight, 2017. [Online]. Available: https://www.emeraldinsight.com/doi/pdfplus/10.1108/JCM-03-2017-2141.

[12] C. Hutto and E. Gilbert, "Vader: A parsimonious rule-based model for sentiment analysis of social media text," in *Eighth International Conference on Weblogs and Social Media (ICWSM-14)*, Ann Arbor, MI, 2014. [Online]. Available: https://www.aaai.org/ocs/index.php/ICWSM/ICWSM14/paper/download/8109/8122.

[13] W. Kenton, "Wisdom of crowds," Investopedia, 2018. [Online]. Available: https://www.investopedia.com/terms/w/wisdom-crowds.asp.

[14] Skymind, "A beginner's guide to neural networks and deep learning," in *A.I. Wiki*, Skymind, 2018. [Online]. Available: https://skymind.ai/wiki/neural-network.

[15] J. DeMuro, "What is a neural network," in *World of tech*, techradar, 2018. [Online]. Available: https://www.techradar.com/uk/news/what-is-a-neural-network.

[16] F. Bach, "Supervised dictionary learning," in *Advances in neural information processing systems*, NIPS Proceedings, 2009, pp. 1033–1040. [Online]. Available: http://papers.nips.cc/paper/3448-supervised-dictionary-learning.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, 1985. [Online]. Available: https://apps.dtic.mil/docs/citations/ADA164453.

[18] Skymind, "A beginner's guide to lstms and recurrent neural networks," in *A.I. Wiki*, Skymind, 2018. [Online]. Available: https://skymind.ai/wiki/lstm.

[19] N. Donges, "Recurrent neural networks and lstm," Towards Data Science, 2018. [Online]. Available: https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5.

[20] P. Jason Brownlee, "A gentle introduction to exploding gradients in neural networks," Machine Larning Mastery, 2017. [Online]. Available: https://machinelearningmastery.com/exploding-gradients-in-neural-networks/.

[21] S. D. S. Team, "Recurrent neural networks (rnn) - the vanishing gradient problem," Super Data Science, 2018. [Online]. Available: https://www.superdatascience.com/blogs/recurrent-neural-networks-rnn-the-vanishing-gradient-problem.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," in *Neural computation, Volume 9. 8*, MIT Press, 1997, pp. 1735–1780. [Online]. Available: https://www.bioinf.jku.at/publications/older/2604.pdf.

[23] S. Yan, "Understanding lstm and its diagrams," Medium, Mar 13, 2016. [Online]. Available: https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714.

[24] C. Olah, "Understanding lstm networks," 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs.

[25] R. Kompella, "Using lstms to forecast time-series," Towards Data Science, 2018. [Online]. Available: https://towardsdatascience.com/using-lstms-to-forecast-time-series-4ab688386b1f.

[26] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th Symposium on Operating Systems Design and Implementation 16)*, 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

[27] Stanford, "Optimization: Stochastic gradient descent," in *UFLDL Tutorial.* [Online]. Available: http://deeplearning.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent.

[28] R. Mitra, "What are differences between update rules like adadelta, rmsprop, adagrad, and adam," Quora, 2016. [Online]. Available: https://www.quora.com/What-are-differences-between-update-rules-like-AdaDelta-RMSProp-AdaGrad-and-AdaM.

[29] M. C. Mukkamala and M. Hein, "Variants of rmsprop and adagrad with logarithmic regret bounds," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR.org, 2017, pp. 2545–2553. [Online]. Available: https://arxiv.org/pdf/1706.05507.pdf.

[30] R. Khandelwal, "Overview of different optimizers for neural networks," Medium, 2019. [Online]. Available: https://medium.com/datadriveninvestor/overview-of-different-optimizers-for-neural-networks-e0ed119440c3.

[31] J. L. B. Diederik P. Kingma, "Adam: A method for stochastic optimization," in *arXiv preprint arXiv:1412.6980*, arXiv, 2014. [Online]. Available: https://arxiv.org/pdf/1412.6980.pdf.

[32] I. Rish *et al.*, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, 2001, pp. 41–46. [Online]. Available: https://www.cc.gatech.edu/~isbell/reading/papers/Rish.pdf.

[33] Skymind, "A beginner's guide to bag of words and tf-idf," in *A.I Wiki*, Skymind, 2018. [Online]. Available: https://skymind.ai/wiki/bagofwords-tf-idf.

[34] T. Karmali, "Spam classifier in python from scratch," Towards Data Science, Aug 2, 2017. [Online]. Available: https://towardsdatascience.com/spam-classifier-in-python-from-scratch-27a98ddd8e73.

[35] J. Roesslein, "Tweepy documentation," 2009. [Online]. Available: http://docs.tweepy.org/en/v3.5.0/.

[36] S. Deoras, "Tensorflow vs. theano: What do researchers prefer as an artificial intelligence framework," Analytics India, 2017. [Online]. Available: https://www.analyticsindiamag.com/tensorflow-vs-theano-researchers-prefer-artificial-intelligence-framework.

[37] bitcoincharts, Bitcoin Charts. [Online]. Available: http://api.bitcoincharts.com/v1/csv/.

[38] A. Nolla, "Detecting text language with python and nltk," Alejandro Nolla Blog. [Online]. Available: http://blog.alejandronolla.com/2013/05/15/detecting-text-language-with-python-and-nltk/.

[39] P. Cryptography, "A tutorial on automatic language identification - ngram based," Practical Cryptography. [Online]. Available: http://practicalcryptography.com/miscellaneous/machine-learning/tutorial-automatic-language-identification-ngram-b/.

[40] T. Risueno, "What is the difference between stemming and lemmatization," Bitext, Feb 26, 2018. [Online]. Available: https://blog.bitext.com/what-is-the-difference-between-stemming-and-lemmatization/.

[41] scikit-learn developers, "Naive bayes," Scikit-Learn. [Online]. Available: https://scikit-learn.org/stable/modules/naive_bayes.html.

[42] T. K. .-.-. tejank10, "Spam-or-ham," Github, Aug 2, 2017. [Online]. Available: https://github.com/tejank10/Spam-or-Ham.

15 Appendices

15.1 Appendix A - Project Initiation Document

Displayed on the following pages below.

Individual Project (CS3IP16)

Department of Computer Science
University of Reading

Project Initiation Document

PID Sign-Off

Student No.	**24005432**
Student Name	**Andrew Sotheran**
Email	**andrew.sotheran@student.reading.ac.uk**
Degree programme (BSc CS/BSc IT)	**BSc CS**
Supervisor Name	**Kenneth Boness**
Supervisor Signature	
Date	

SECTION 1 – General Information

Project Identification

1.1	**Project ID** (as in handbook)
	N/A
1.2	**Project Title**
	Cryptocurrency market and value prediction tracking
1.3	**Briefly describe the main purpose of the project in no more than 25 words**
	To provide a means to predict the value of cryptocurrencies that will aid in investor decision making in investment of the market

Student Identification

1.4	**Student Name(s), Course, Email address(s)** e.g. Anne Other, BSc CS, a.other@student.reading.ac.uk
	Andrew William Sotheran BSc CS Andrew.sotheran@student.reading.ac.uk

Supervisor Identification

1.5	**Primary Supervisor Name, Email address** e.g. Prof Anne Other, a.other@reading.ac.uk
1.6	**Secondary Supervisor Name, Email address** Only fill in this section if a secondary supervisor has been assigned to your project

Company Partner (only complete if there is a company involved)

1.7	**Company Name**
	N/A
1.8	**Company Address**
	N/A
1.9	**Name, email and phone number of Company Supervisor or Primary Contact**
	N/A

SECTION 2 – Project Description

2.1 **Summarise the background research for the project in about 400 words. You must include references in this section but don't count them in the word count.**

To create a tool that aims to predict the price of cryptocurrencies that aids in investor decisions. Research will need to be conducted into the following topics that surround data mining, machine learning and artificial neural networks.

This research will consist along the lines of;
Natural Language processing and analysis – To analyse and process fed in data gathered through RSS data feeds and social media feeds, through the underlying tasks of Natural language processing. Content categorisation (search and indexing, duplication detection), Topic discovery and modelling (Obtain meanings and themes within the data and perform analytic techniques), sentiment and semantic analysis (which will identify the mood and opinions within the data), summariser (to summarise a block of text and disregard the rest).

Machine learning algorithms: The three types of machine learning (Supervised, Unsupervised and Reinforced)
The types of common algorithms used, each of these will be researched to identify the most suitable for this project and only one will be used: (Linear Regression, Logistic Regression, Decision Tree, SVM, Naive Bayes, kNN, K-Means, Random Forest, Dimensionality Reduction Algorithms, Gradient Boosting algorithms (GBM, XGBoost, LightGBM, CatBoost).

Artificial Neural Networks: To identify the drawbacks and benefits of using them or other computational models within machine learning. Recurrent Neural networks and 3rd generation Neural Networks.

Data mining: To investigate the different techniques and algorithms used (Same as the ones listed above for machine learning including C4.5, Apriori, EM, PageRanks, AdaBoost and CART) these will be researched and the most appropriate identified.

To investigate techniques: for storing and processing large amount of data, such as Hadoop, Elasticsearch utilities, Graphing and data modelling and visualisation.

To identify appropriate libraries for python or C for each of the topics above to aid in the creation of this project. Libraries such as:
Natural Language Toolkit (NLTK) – python
Pandas - python
Sklearn - python
Numpy – python - scientific computation for working with arrays
Matplotlib - python - data visualisation

Investigate into types of databases. Sql and nosql for a storage medium between receiving data and feeding it into the machine learning algorithm.
Investigate into the use of REST API and other web-service based technologies (GRPC, Elasticsearch)
Investigate into frameworks for the thin client, such as Angular vs React, Nodejs, Leafelt.js, charts.js
Additionally Web scraping may be needed if certain website that don't either have an API or JSON for the data needed.

https://www.sas.com/en_gb/insights/analytics/what-is-natural-language-processing-nlp.html
https://blog.algorithmia.com/introduction-natural-language-processing-nlp/
https://gerardnico.com/data_mining/algorithm
https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/
https://www.kdnuggets.com/2015/05/top-10-data-mining-algorithms-explained.html
https://www.datasciencecentral.com/profiles/blogs/artificial-neural-network-ann-in-machine-learning
http://scikit-learn.org/stable/index.html
https://grpc.io/docs/

2.2	**Summarise the project objectives and outputs in about 400 words.** These objectives and outputs should appear as tasks, milestones and deliverables in your project plan. In general, an objective is something you can do and an output is something you produce – one leads to the other.
	To produce a thin web client that provides a dashboard that provides tangible and useful information to users such as; Their current price (Updated every 5 minutes), exchange rates, network hashrates, historical price data. It will also display statistics about sentiment analysis conducted on social media about the currency, graphical predictions on what the price may be, in a given time, and will also compare this to other currencies for aid in investment. To produce significant research into the topics in and around data mining, machine learning and Artificial Neural network and the underlying tasks and algorithms used, the efficiency, drawbacks and advantages of each to identify the most suitable for the use in this project. To produce a system that analyses a data set obtained through social media feeds and posts on news sites regarding crypto currencies. It should perform sentiment analysis using Natural Language processing and analysis techniques to identify features and identifies the type of sentiment in the data and categorises it for machine learning. To utilise machine learning techniques and algorithms to produce a system that learns from historical data to predict to an extent the possible future price of a given currency. To compare this with the use of an Artificial Neural Network and to analyse the drawbacks of both.
2.3	**Initial project specification - list key features and functions of your finished project.** Remember that a specification should not usually propose the solution. For example, your project may require open source datasets so add that to the specification but don't state how that data-link will be achieved – that comes later.
	The finished project should provide a thin client single page application. This will provide a means to users the ability to view various statistics on crypto currencies on a dashboard that incorporates text analysis through natural language analysis, and will utilise various machine learning and data mining techniques to provide price predictions to the users. The nature and level of this will depend on the research conducted into the areas of data mining, machine learning, natural language processing and artificial neural networks, along with the algorithms used. The data set will be created from scratch for this project as it will require the gathering of data from numerous sources and performing text analysis on them to for the data needed. Data sets for the characteristic and data for the currencies can be obtained from pre-existing data sets such as: https://www.kaggle.com/sudalairajkumar/cryptocurrencypricehistory https://www.kaggle.com/jessevent/all-crypto-currencies Web scraping may be included if certain news/social media websites do not provide an API or RSS feed for the analysis engine to perform text analysis on Additionally, there will be a server between the analysis/prediction engine and the thin client that will maintain a database, either SQL or NoSQL, that will hold statistics about the currencies and data about the price predictions about the currencies. It will not hold any of the data used in the analysis engine, as this database will only hold data available to the end users.

2.4	**Describe the social, legal and ethical issues that apply to your project. Does your project require ethical approval? (If your project requires a questionnaire/interview for conducting research and/or collecting data, you will need to apply for an ethical approval)**
	The project will not be handling any user related data, therefore it does not need ethical approval.
2.5	**Identify and lists the items you expect to need to purchase for your project. Specify the cost (include VAT and shipping if known) of each item as well as the supplier.** e.g. item 1 name, supplier, cost
	None Needed
2.6	**State whether you need access to specific resources within the department or the University e.g. special devices and workshop**
	Possibly a server to host the database and analysis engine on to perform the computation necessary, and a server to host the thin client.

SECTION 3 – Project Plan

3.1	**Project Plan** Split your project work into sections/categories/phases and add tasks for each of these sections. It is likely that the high-level objectives you identified in section 2.2 become sections here. The outputs from section 2.2 should appear in the Outputs column here. Remember to include tasks for your project presentation, project demos, producing your poster, and writing up your report.

Task No.	**Task description**	**Effort (weeks)**	**Outputs**
1	**Background Research**		
1.1	Investigate into RPC frameworks and REST APIs	0.3	To identify the type of API/RPC framework that would be most suitable
1.2	Research into Natural Language processing and analysis techniques	0.5	To get an understanding of how NLP works and how it could be used
1.3	Research into the use of machine learning – types and algorithms	0.5	To grasp how ML paradigms work and how this project will use it
1.4	Research into the application of Neural Networks – drawbacks and advantages of using them	0.3	To identify whether there will be a need for a neural network or ML paradigms can be used instead
1.5	Research techniques for storing and processing large amount of data, such as Hadoop, spark or Elasticsearch utilities.	1	To understand the uses, application and whether the use of these are more viable solution than standard ML practices
1.6	Identify appropriate libraries for data modelling and visualisation, NLP and Machine Learning	1	To identify what libraries will aid in the construction of this project
1.7	Investigate into frameworks for the front-end thin clients	0.3	To identify what frameworks the thin client should be used with, along with drawbacks and advantages
1.8	Research web scraping techniques	0.3	To understand the application of these techniques and learn how to apply them
2	**Analysis and design**		
2.1	Resolve issues discovered by background research	0.2	...
2.2	Identify limitations discovered from research and what is not feasible	0.1	…
2.3	UML Diagrams/ XUML	0.2	
2.4	Wire frames for frontend	0.1	
2.5	Data Flow	0.1	
2.6	User Flow	0.1	
3	**Develop prototype**		
3.1	Develop thin client	2	
3.2	Develop analysis Engine	4	
3.3	Develop Prediction Engine	3	
3.4	Develop Unit tests	2	
4	**Testing, evaluation/validation**		
4.1	Unit testing	1	
4.2	Acceptance Testing	0.8	
4.3	User testing	0.8	
5	**Assessments**		
5.1	write-up project report	2	Project Report
5.2	produce poster	0.5	Poster
5.3	Log book	0.5	

TOTAL	**Sum of total effort in weeks**	**21.9**	

SECTION 4 - Time Plan for the proposed Project work

For each task identified in 3.1, please *shade* the weeks when you'll be working on that task. You should also mark target milestones, outputs and key decision points. To shade a cell in MS Word, move the mouse to the top left of cell until the curser becomes an arrow pointing up, left click to select the cell and then right click and select 'borders and shading'. Under the shading tab pick an appropriate grey colour and click ok.

	START DATE: 10/2018 <enter the project start date here> **Project Weeks**												
Project stage	0-3	3-6	6-9	9-12	12-15	15-18	18-21	21-24	24-27	27-30	30-33	33-36	36-39
1 Background Research													
Investigate into RPC frameworks and REST APIs	🟦												
Research into Natural Language processing	🟦												
Research into the use of machine learning –	🟦												
Research into the application of Neural	🟦												
Research techniques for storing and	🟦												
Identify appropriate libraries for data		🟦											
Investigate into frameworks for the front-		🟦											
Research web scraping techniques		🟦											
2 Analysis/Design													
Resolve issues discovered by background		🟦											
Identify limitations discovered from		🟦											
UML Diagrams/ XUML		🟦											
Wire frames for frontend			🟦										
Data Flow			🟦										
User Flow			🟦										

3 Develop prototype.													
Develop thin client			🟦										
Develop analysis Engine			🟦	🟦	🟦								
Develop Prediction Engine					🟦	🟦							
Develop Unit tests						🟦							
4 Testing, evaluation/validation													
Unit testing						🟦	🟦						
Acceptance Testing							🟦						
User testing							🟦						
5 Assessments													
write-up project report							🟦	🟦					
produce poster								🟦					
Log book								🟦					

RISK ASSESSMENT FORM

Assessment Reference No.		**Area or activity assessed:**	
Assessment date			
Persons who may be affected by the activity (i.e. are at risk)	**Andrew Sotheran**		

SECTION 1: Identify Hazards - *Consider the activity or work area and identify if any of the hazards listed below are significant (tick the boxes that apply).*

1.	Fall of person (from work at height)		**6.**	Lighting levels		**11.**	Use of portable tools / equipment		**16.**	Vehicles / driving at work		**21.**	Hazardous fumes, chemicals, dust		**26.**	Occupational stress	
2.	Fall of objects		**7.**	Heating & ventilation		**12.**	Fixed machinery or lifting equipment		**17.**	Outdoor work / extreme weather		**22.**	Hazardous biological agent		**27.**	Violence to staff / verbal assault	
3.	Slips, Trips & Housekeeping	X	**8.**	Layout , storage, space, obstructions		**13.**	Pressure vessels		**18.**	Fieldtrips / field work		**23.**	Confined space / asphyxiation risk		**28.**	Work with animals	
4.	Manual handling operations		**9.**	Welfare facilities		**14.**	Noise or Vibration		**19.**	Radiation sources		**24.**	Condition of Buildings & glazing		**29.**	Lone working / work out of hours	
5.	Display screen equipment	X	**10.**	Electrical Equipment	X	**15.**	Fire hazards & flammable material		**20.**	Work with lasers		**25.**	Food preparation		**30.**	Other(s) - specify	X

SECTION 2: Risk Controls - *For each hazard identified in Section 1, complete Section 2.*

Hazard No.	Hazard Description	Existing controls to reduce risk	**Risk Level** (tick one)			Further action needed to reduce risks *(provide timescales and initials of person responsible)*
			High	Med	Low	
3	Tripping over wires	Cable management is at a minimum, none are currently properly cable managed and kept out of way			x	Sufficient cable management needed, cables tied together and moved out of way of feet
5	Eye strain from looking at a monitor	Current screen contrast and brightness is acceptable		x		To have periodic breaks from the screen
Name of Assessor(s)			**SIGNED**			
Review date						

Health and Safety Risk Assessments – continuation sheet

Assessment Reference No	
Continuation sheet number:	

SECTION 2 continued: Risk Controls

Hazard No.	Hazard Description	Existing controls to reduce risk	**Risk Level** (tick one)			Further action needed to reduce risks
			High	Med	Low	*(provide timescales and initials of person responsible for action)*

Name of Assessor(s)		**SIGNED**
Review date		

15.2 Appendix B - Log book

The log book for this project is a physical book and was handed to the School of Computer Science. Due to being a physical book, it cannot be inserted here.