April 2016

Data Mining in Financial Domain

Essam R. Al-Mansouri
Worcester Polytechnic Institute

Sean J. Amos
Worcester Polytechnic Institute

Using Artificial Neural Networks and Sentiment Analysis to Predict Upward Movements in Stock Price

A Major Qualifying Project Submitted to the faculty of

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree in Bachelor of Science in Computer Science

Submitted by:

Essam Al-Mansouri

Sean Amos

Date:

April 28th, 2016

Report Submitted to:

Professor Carolina Ruiz

Professor Hossein Hakim

Professor Michael Radzicki

ABSTRACT

For this project, we explored the use of text mining, clustering, and machine learning models to develop a system that combines technical and sentiment analysis to determine the movement of a stock. The final result of our project is a system comprised of a novel sentiment analysis used as input for the larger recurrent neural networks, each trained on a cluster of stocks from the S&P 100. Experimental results show that our system can predict upward movements in stock price over a 65-minute period with up to 77% accuracy for a specific cluster compared to 52% of randomly guessing for the same cluster.

ACKNOWLEDGEMENTS

The following individual and institutions, in no particular order, have been tremendously supportive throughout the duration of the project. We simply cannot thank them enough.

- **Professor Carolina Ruiz** of Worcester Polytechnic Institute for her tireless support and feedback throughout the entire experience.
- **Professors Hossein Hakim and Michael Radzicki** of Worcester Polytechnic Institute for lending us their time and knowledge of the financial market.
- **Nicholas Bradford** of Worcester Polytechnic Institute for his help with investigating clustering techniques.
- **Worcester Polytechnic Institute** for the opportunity to work on a Major Qualifying Project that has real world applications.

TABLE OF CONTENTS

Abstract i

Acknowledgements ii

Table of Contents iii

Table of Figures v

Table of Tables vi

Executive Summary vii

Background vii

Methodology vii

Findings & Conclusion viii

1. Introduction 1

2. Background 3

2.1. Financial Analysis 3

2.1.1. Sentiment Analysis 3

2.1.2. Technical Analysis 3

2.2. Text Mining 4

2.3. Machine Learning 5

2.4. Clustering & Risk Minimization 8

3. Methodology 9

3.1. Sentiment Analysis 9

3.1.1. Data Collection 9

3.1.2. Preprocessing 11

3.1.3. Predicting and Training 12

3.1.4. Post processing 15

3.1.5. Example Sentiment Analysis Machine 16

3.1.6. Evaluation 22

3.2. Clustering & Risk Minimization 23

3.3. Neural Network 23

3.3.1. Features & Outputs 24
3.3.2. Regularization 25
3.3.3. Activation function 25
3.3.4. Training & Hyperoptimization 26

4. Results 28

4.1. Sentiment Analysis Results 28

4.2. Clustering 31

4.3. Neural Network Results 32

5. Conclusion & Future Work 35

5.1. Future Work 35

6. Bibliography 37

7. Appendices 42

Appendix A Source Code 42

Appendix B List of Third Party Libraries Used 43

Appendix C Example RSS Feed Data 44

Appendix D Google Historic Price Data Format 45

Appendix E Yahoo Historic Price Data Format 46

Appendix F S&P 100 by Cluster 47

TABLE OF FIGURES

Figure 1 Accuracy by clusters viii

Figure 2 Japanese Candlestick Chart [36] 3

Figure 3 A Candlestick [37] 4

Figure 4: A simple neural network with five inputs 5

Figure 5 Example Long Short Term Memory Neural Network 6

Figure 6 Dropout applied to connections between feed forward connections [12] 7

Figure 7 Example DTW [35] 8

Figure 8 First example article prediction vector 17

Figure 9 First example article price vector 17

Figure 10 Second example prediction vector 18

Figure 11 Second example price vector 19

Figure 12 Third example prediction vector 20

Figure 13 Third example price vector 20

Figure 14 Fourth example article prediction vector 21

Figure 15 Fifth example article prediction vector 21

Figure 16 Example graph of a weighted slice 22

Figure 17: Dow Jones Index on 297 consecutive days (Left). Daily change of Dow Jones Index on 297 consecutive days (Right) 24

Figure 18 Accuracy by clusters 32

Figure 19 Trading strategies over 5 year time 33

TABLE OF TABLES

Table 1 Neural network layers 26

Table 2 Article distribution over days 28

Table 3 0 threshold accuracy 29

Table 4 0 threshold average profit % per trade 29

Table 5 Optimized threshold values 30

Table 6 Optimized accuracy 30

Table 7 Optimized average profit % per trade 31

Table 8 Neural network based prediction accuracy 32

Table 9 Financial metric comparison 34

EXECUTIVE SUMMARY

BACKGROUND

In today's computerized world, everyone can quickly gain access to the financial market and trade stocks or currencies from the comfort of their office or home through online brokers. The internet has provided a level of accessibility to the average person that had been previously confined to professional traders and investors. These traders, much like any trader, are constantly analyzing the market and trying to predict the future value of a stock. All traders must consider different types of the market information when they think about a trade. The first is sentimental analysis or trying to understand how investors, consumers, or the world feel about a company and use that to predict how a company's stock will behave. On the other side we have technical analysis which makes predictions based on historical price data or defined company information. Unlike sentiment analysis predictions can be made based solely on the formulaic understanding of historic price data and trade volume.

METHODOLOGY

The goal of our project was to design and implement a machine learning system that would accurately predict whether a stock's price would be higher 65 minutes into the future. To achieve this our system utilized clustering over the stocks of the S&P 100, sentiment analysis, and for each cluster a neural network that took as input date information, historic price data, and a sentiment value from the sentiment analysis. This system was implemented in Python, utilizing over external libraries focusing in machine learning, and natural language processing.

The sentiment analysis that we used for the project is a machine learning technique that utilizes stemmed bag-of-words models and weighted performance averages of stemmed words from past news articles to predict the movement of a stock for the next 2.5 trading days. The news articles and stock price data were collected from Google and Yahoo RSS feeds. The end product was able to take a stock, and arbitrary date and time, and a timebar to produce a scalar value indicating whether it thinks the stock will go up or down.

We decided to cluster the stocks of the S&P 100 since it allows for risk minimizing investments. However, early experiments showed that training and testing neural networks on

clusters of stocks performed better than if they were trained on all the stocks, adding an additionally reason to cluster. Specifically, we used hierarchical agglomerative clustering algorithm with Dynamic Time Warping (DTW) and weighted distance to cluster the stocks

For the neural network we decided to use a recurrent neural network variant called Long Short Term Memory (LSTM), which can handle problems with hundreds of time steps between important events. This neural network serves as the main prediction system and takes as input 100 consecutive 65-minute stock price data points (date and time, open price, min price, max price, close price, and volume) and the sentiment value. The actual price data is detrended, so that it takes value lost or gained from each time step.

Findings & Conclusion

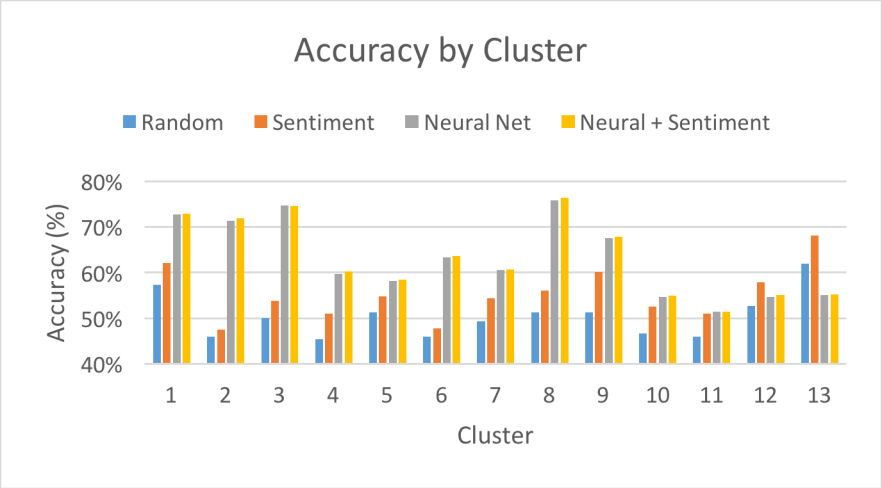

Figure 1 Accuracy by clusters

We ran back testing of unseen data on the individual machines, the combination of the two machines, and a strategy where it randomly decides to buy. Figure 1 shows that, in most cases, across the 13 clusters our system and its components are able to outperform the random strategy by a significant amount. The neural network with the sentiment value is able to achieve accuracy up to 77% compared to the random strategy which only was able to get up to 62% accuracy on a cluster. The results also show that the neural network with the sentiment value does perform better than either of the individual components; even if just marginally better than

the neural network by itself. Therefore, we can say that our system achieved its goal of accurately predicting upward stock movements. We remain confident in our system's predictions and optimistic about its potential use in future real world applications.

1. INTRODUCTION

In today's computerized world, everyone can quickly gain access to the financial market and trade stocks or currencies from the comfort of their office or home through online brokers. The internet has provided a level of accessibility to the average person that had been previously confined to professional traders and investors. The increase in non-professional traders and investors has greatly impacted advancements in trading software. Popular electronic trading platforms such as TradeStation allow traders to write and run custom programs that can automatically enter or exit traders based on any specific programmed conditions [1].

Unfortunately, the financial market is always changing and evolving and as a consequence, is extremely difficult to predict. Any fixed trading strategy is guaranteed to face unfavorable market conditions and potentially suffer major losses. One study suggests that four out of five of all day traders actually quit within the first two years and that only one out of every 100 traders consistently profit from trading [2].

Luckily, most popular trading platforms allow traders to write custom trading applications that can be executed in a simulated trading environment. This allowed us to apply statistical clustering and machine learning tools on decades of historical data to ultimately create a system that can predict if a stock's price will be higher approximately an hour into the future.

Most day traders trade based on technical analysis are based on analyzing a stock's price chart, looking for meaningful repeating patterns in the data that can be used to forecast market conditions. However, not only do many of those patterns not actually have predictive properties, many of them are so vaguely defined that it is practically impossible for a trader to consistently recognize, and then the decision to buy or sell falls largely on intuition instead of data and statistical analysis. A significant advantage to using statistical machine learning models that can learn to trade is that we avoid make decisions based on unreliable human emotion and cognitive bias.

For our project we wanted to use the advantages brought by statistical machine learning models. More precisely, the goal of this project was to design and implement a machine learning system that would accurately predict whether a stock's price would be higher 65 minutes into the future. To achieve this our system utilized clustering over the stocks of the S&P 100, sentiment

analysis, and for each cluster a neural network that took as input date information, historic price data, and a sentiment value from the sentiment analysis.

Experimental results of this system on five years of unseen 65 minute S&P100 stock data show that our Long Short Term Memory (LSTM) neural networks can predict if a stock's price will be higher one hour later with up to 77% accuracy on several different stocks, an approximately 25% improvement over baseline performance.

2. BACKGROUND

2.1. FINANCIAL ANALYSIS

Traders are constantly analyzing the market and trying to predict the future value of a stock. There are different types of the market information that most traders consider before making a trade. In this project, we will utilize technical and sentiment analysis to predict future value of the top 100 leading stocks in the U.S (referred to as the S&P 100).

2.1.1. Sentiment Analysis

Many traders base their trades strictly on sentiment analysis. Sentiment analysis is the process of trying to understand the sentiment of consumers, investors or traders and predicting how that would affect the value of a stock. Company announcements, news articles, and even rumors can have a profound effect on a company's reputation and perceived value. Unfortunately, information used in sentiment analysis can be very difficult to quantify.

2.1.2. Technical Analysis

Technical analysis is the process of analyzing historical stock data when predicting future stock value. Unlike sentiment analysis, technical analysis uses strictly quantitative information, such as past stock prices and volume, to make predictions about the stock's price in the future.

Stock traders commonly use Japanese Candlestick charts, Figure 2 ,when analyzing historical stock data.

Figure 2 Japanese Candlestick Chart [36]

A candlestick chart presents price over time as a series of red and green 'candlesticks' each presenting a specific period of time. In an hourly candlestick chart, each candlestick is constructed or formed over a 1 hour period.
A candlestick, Figure 3, shows information about the stock price during the candlestick's formation. The top and the bottom of the candlestick's wicks represents the highest and lowest price during the bar's formation. The real body of the bar represents the opening price (price as soon as candlestick began forming) and the closing price (price at the end of the 1 hour period).

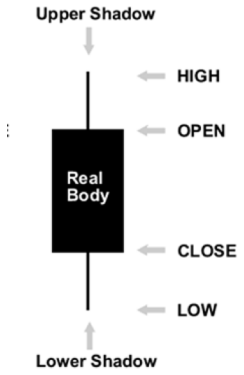

Figure 3 A Candlestick [37]

2.2. Text Mining

Text mining is a complex task in which computer algorithms are used to process text and to derive meaning or patterns from the text. In the process some form of natural language processing is usually utilized to create a more appropriate understanding of the text. This is combined with machine learning and statistical analysis to automatically discover patterns in the data [3].

The use of text mining has been widely studied in the field of financial markets, but the problem still remains very difficult. This problem has been approached in a number of ways. The main one and the one used by this project, is the belief that articles, blogs, and tweets encompass a sort of market sentiment towards a company. If there is a sudden surge of negative tweets or articles about a company then the market will react accordingly and the value of the company's stock will decrease [4].

One research project looked at the arduous task of creating a lexicon, a list of words that make up the language, based on a number of statistical functions. The researchers showed that they were able to create such a lexicon and by classifying each word in a message as bullish or bearish they were able to market trend up to 80% accuracy just using tweeter feeds as the input [5]. Another researcher looked into experimenting with a number of linguistic representations of news articles. They explored the use of Bag of Words (considering the multiset representation of the text), Noun Phrases, and Named Entities as article representations that are given to a Support Vector Machine (SVM), a learning algorithm. Using this system, they were able to get

directional accuracy to be around 55%. The Noun Phrases performed the best for the textual representation however the bag of words model faired only slight worse [6].

2.3. Machine Learning

It is near impossible to write a program that follows a set of hardcoded rules that can adapt to the constantly changing market conditions. If we wanted to predict future stock prices with hard coded rules, we might first try to find specific indicators or markers that foreshadow a specific movement in the market in the past and then code software that makes predictions based on those markers. However, even if our software performed well, our work would quickly be rendered obsolete due to a changing market environment. The markers that foreshadow a specific price movement can be very different in different market environments. This quickly becomes an unsurmountable project when we consider that we would need to constantly search for new useful markers to learn and for old obsoleted markers to unlearn and to update our software accordingly.

Machine learning algorithms give computers the ability to learn, without being explicitly hard coded, to solve a problem. Recent progress in training deep neural networks and recurrent neural networks have made them excel at learning patterns and correlations in data even in extremely noisy domains.

Artificial neural networks are machine learning models comprised of connected layers of computational units, referred to as neurons, Figure 4. Each neuron's incoming connections have an assigned weight that is learned through training on example dataset. Training a network to behave in a certain way usually consists of giving the network a training example input and comparing its output to our actual desired output, and then slightly updating the network weights such that the network's output is more accurate in the future. Variants of the neural network model were able to

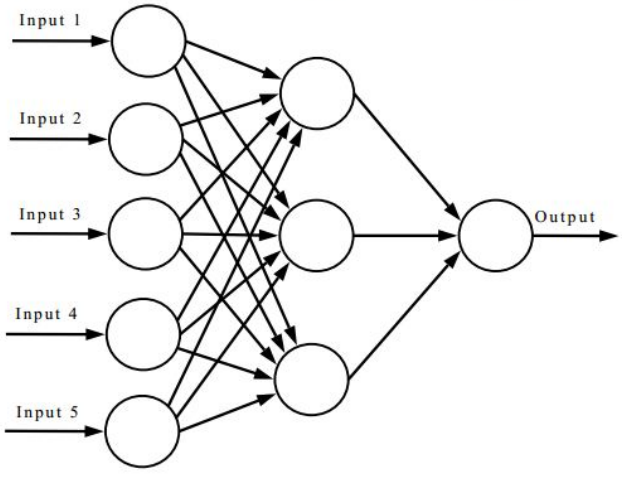

Figure 4: A simple neural network with five inputs

outperform humans in playing games, reading handwritten text [7], recognizing faces [8], driving cars [9], flying helicopters [10], and even diagnosing ill patients [11].

Recurrent neural networks refer to a type neural networks whose connections form a directed cycle. This allows neurons to store an internal state or memory in a previous time step that influences the network's output at timestep **t**.

In this project we use a variant of a recurrent neural network called Long Short Term Memory (LSTM). LSTMs are well suited to learn from experience when there are very long time lags of unknown size between important events, which makes them especially attractive for applications in the financial market.

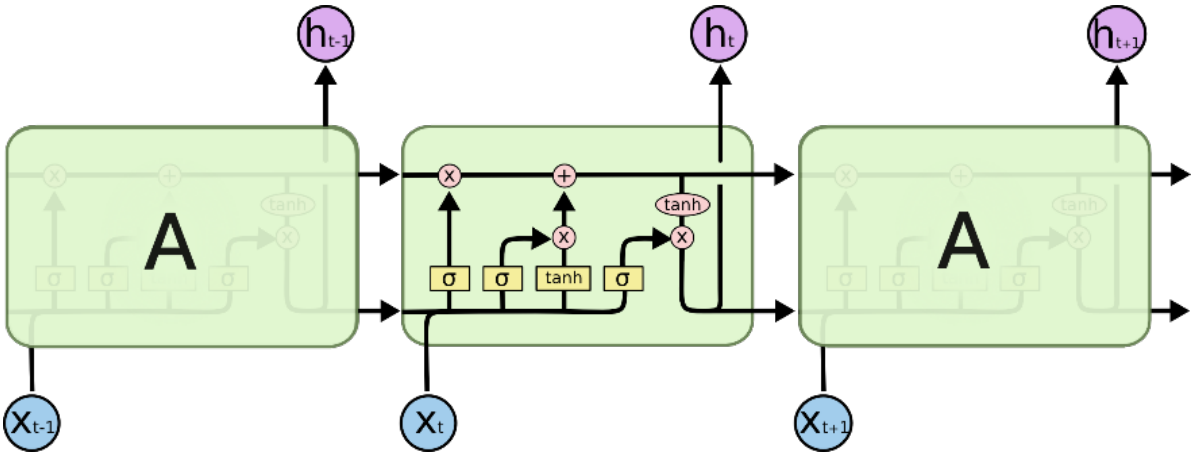

Figure 5 Example Long Short Term Memory Neural Network

An LSTM unit has a forget gate **f**, input gate **i**, output gate **i** and a memory cell **C**. The final output of the neuron **h** is the cell's output modulated by the output gate **o**. These gates allows a network to learn what values to store, forget or remember.

$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Unfortunately, despite recent improvements in neural networks and their proven performance in multiple applications, training neural networks remains to be a difficult problem. A major difficulty in training neural networks is the vanishing gradient problem. The vanishing

gradient problem is a difficulty found in training that causes updates to the network's weights to be disproportionately small which can slow down training and reduce performance. The exploding gradient problem is a similar difficulty which causes weight updates to sometimes be disproportionately large which can also slow down training and reduce performance. Another major difficulty in training is the problem of over fitting. An over fit network performs well on examples it has been trained on, but performs poorly on new unseen data. To help minimize the effect of the over fitting problem, we use different methods referred to as regularization methods.

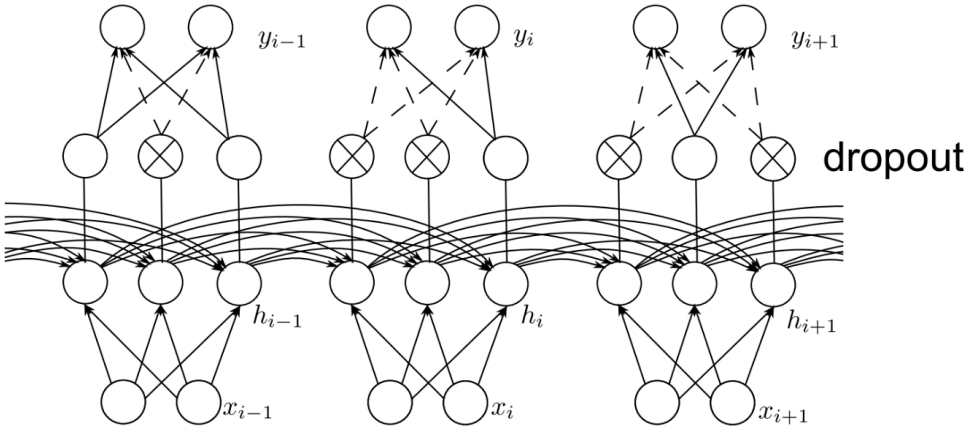

Figure 6 Dropout applied to connections between feed forward connections [12]

Dropout is probably one of the most common regularization methods used in deep networks today. Essentially, a percentage of neurons in a hidden layer are dropped (their output changed to 0) on every training example. This helps the neural network learn to avoid over fitting by forcing the network to learn redundant representations of the data. Each neuron is forced to learn a useful representation that isn't dependent on only one or two other neurons. The expected result is that the network generalizes better over the data because if only a handful of the neurons misbehave, redundancy alleviates their effects. *Regularizing RNNs by Stabilizing Activations* was the only regularization method that was specifically designed for recurrent neural networks. It penalizes neurons for changing their values too wildly between one time step and the next unless it improves performance. The intuition is that it reduces activation variance and results in a more stable network that generalizes better.

2.4. Clustering & Risk Minimization

In finance, an investment portfolio is an investor's collection of investments. Portfolio management is the decision making process in the selection of assets to invest in and the allocation an investor's resources to these investments. Portfolio management consists largely of minimizing an investment portfolio's risk to reward ratio. In this project we develop a methodical and mathematical approach to select a diverse set of stocks by applying clustering algorithms to financial data.

Clustering is the process of grouping a set of objects into groups or clusters based on their similarity to each other, such that objects are most similar to the objects in their own cluster than to objects in other clusters. One type of clustering is hierarchical agglomerative clustering which is an approach that initially places each object in its own cluster then iteratively combines the most similar clusters until all objects are in a single cluster.

Similarity between objects can be measured a number of ways. One possibility is through the use of Dynamic Time Warping, which is a method used in speech recognition to measure similarity between two sequences to group audio of similar words together, even if the speakers are talking at different speeds.

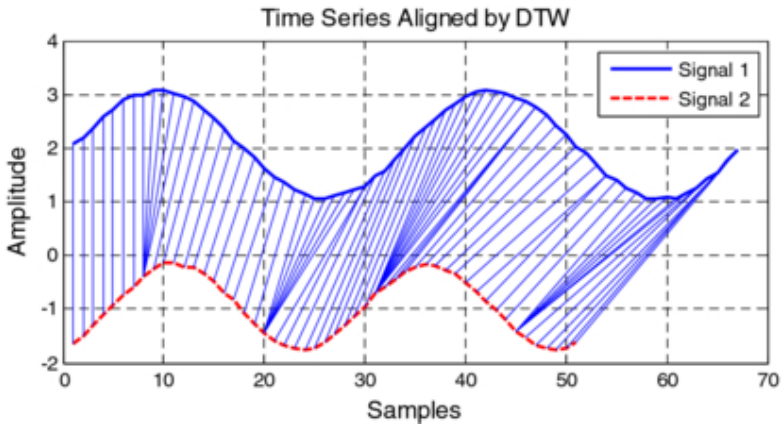

Figure 7 Example DTW [35]

3. METHODOLOGY

The goal of this project was to design and implement a machine learning system that would accurately predict whether a stock's price would be higher 65 minutes into the future. To achieve this our system utilized clustering over the stocks of the S&P 100, sentiment analysis, and for each cluster a neural network that took as input date information, historic price data, and a sentiment value from the sentiment analysis.

For our implementation language we chose to use Python. The readability and ease of use of the language allowed for quick development time and multiple iterations throughout the project. In addition, Python has an extensive suite of third party libraries for neural networks, clustering, and natural language processing. We utilized these libraries to significantly lessen the effort necessary for the complex tasks needed by this project. Our source code and third party libraries for the implementation can be found in Appendix A and Appendix B, respectively.

3.1. Sentiment Analysis

The sentiment analysis that we used for a project is a machine learning technique that utilizes stemmed bag-of-words models and weighted performance averages of stemmed words from past news articles to predict the movement of a stock for the next 2.5 trading days. The end product is able to take a stock, and arbitrary date and time, and a timebar to produce a scalar value. This scalar value represents the direction and confidence that the given stock will go up. The actual confidence scale will depend on the post processing technique and potentially is unbounded. A value of +.5 would indicate a high upward movement confidence while -.02 would indicate a low downward confidence. As a general note, times in this work are considered continuous (e.g. 3:55 PM on a Friday + 5 minutes would then become 9:00 AM on a Monday).

3.1.1. Data Collection

Our sentiment analysis machine requires articles about a specific company and the stock price in order to make predictions and train from past data. In both cases Google and Yahoo provide optimal APIs for collecting news articles by company and historic stock prices. To ensure enough articles for large scale training we pull articles and prices for for each company on the S&P 500. To collect articles, we made a call to both search engine's Rich Site Summary

(RSS) feed, a continuously updated list of relevant articles of format found in Appendix A. This prevents the need for us to implement a web crawler to find not only the articles but also determine the relevant company in the article. The calls we used are as follows:

http://finance.yahoo.com/rss/headline?s=[Stock_Ticker]

https://www.google.com/finance/company_news?q=[Exchange]:[Stock_Ticker]&output=rss

These RSS feeds, however, only contain a short description of the article and the link to the actual web page. So once we have collected all RSS feed information, we can then make an HTTP request to all previously unseen article pages. The raw HTML from these requests is then stripped of unnecessary style and script tags using the BeautifulSoup module for Python. This is done simply to decrease the size necessary to store this information, decreasing disk usage ~60%. This stripped down HTML, the web page URL, stock ticker, API type, RSS ID, and publication date are stored in a relational database, in our case SQLite. This ensures that we do not needlessly duplicate previously seen data and it allowed storage of base data over iterations of the sentiment analysis machine.

The collection of stock prices for companies was done in a similar manner. An HTTP request was sent for each of the companies. The calls used are as follows:

http://chartapi.finance.yahoo.com/instrument/1.0/[Stock_Ticker]/chartdata;type=quote;range=15d/csv

https://www.google.com/finance/getprices?q=[Exchange]&x=[Stock_Ticker]&i=60&p=15d&f=d,c,h,l,o,v

Each call returned stock prices for the past 15 days in a format encompassing time, close, high, low, open, and volume. Where the time is the start of a specific duration during the trading day, open is the price of the stock at the start, close is the price of the stock at the end of the duration, high is the highest price of the stock during that period, low is the lowest price during the period and volume is the total number of shares bought and sold during the duration. However, the exact format of these returns differ, as seen in Appendix B and Appendix C. The returns were converted to the same data structure and then saved to a SQLite database, where the columns are the stock ticker, the source, and the six features from before. To function properly, each time, source, and ticker set has to be unique. If a row would violate this constrain, it would not be added to the database. Also in the case of a stock split, or reverse stock split, the resulting prices would be rebased around the original price to maintain consistent pricing.

We used only the Google stock pricing because it produced full minute bar data versus Yahoo's seemingly arbitrary 5-minute (approx.) timebar. Additionally, these data collections were made between 10/12/2015 and 3/4/2016.

3.1.2. Preprocessing

In order to use each of the previously collected datasets we had to preprocess such that the sentiment analysis machine can interpret the data. In the case of articles, we had to first extract the article content from the stripped down HTML. In order to accomplish this, we used a Python module called Newspaper. We constrained the size of the article to between 40 and 5000 words. If the article did not fit that criterion we rejected it. Additionally, we rejected the article if it did not contain the associated company's ticker or name.

Next, we tokenized each continuous alpha string in the article and filtered out those that were not considered real words and those that are considered stop words (i.e. words that appear too frequently in the English language to make meaningful difference between articles). Then for each remaining word we created a bag of words model from the stem of that word. Stemming the word means to create a possibly artificial root for similarly rooted words. This was accomplished using the Snowball English stemmer found in the Natural Language Tool Kit for Python. This stemmed bag of words is the representation of the article that is used by the sentiment machine. The use of a bag of words model has been successfully deployed for sentiment analysis by Schumaker and Chen [6]. To reduce the dimensionality, the stemming was employed.

Much like the articles, we could not directly use the stock pricing data in the machine since each stock has a price that is not directly comparable to another stock price. Since we predicted over an entire period of time, we said we only care about a range starting at some arbitrary time and extending *k* minutes of trading into the future. Thus the closing stock price for a specific company in this range could be represented by the following vector $(c_a, c_{a+1}, \dots, c_{a+k-1}, c_{a+k})$. We can also write a vector as the percentage gained or lost since the initial point in time, $\left(\frac{c_a - c_a}{c_a} = 0, \frac{c_{a+1} - c_a}{c_a}, \dots, \frac{c_{a+k-1} - c_a}{c_a}, \frac{c_{a+k} - c_a}{c_a}\right)$. This puts the vector of different stocks in a similar scale. We may also get a close approximation of this vector by instead taking the partial surrounding average of every n points. So we end up with a vector that resembles the following:

$$\left(0, \frac{\frac{c_{a+n-1}+c_{a+n}+c_{a+n+1}}{3}-c_a}{c_a}, \ldots, \frac{\frac{c_{a+(m-1)n-1}+c_{a+(m-1)n}+c_{a+(m-1)n+1}}{3}-c_a}{c_a}, \frac{\frac{c_{a+mn-1}+c_{a+mn}+c_{a+mn+1}}{3}-c_a}{c_a}\right),$$

where $k = mn$. This vector will be known as the *price vector*, which is the vector that was used when the stock price for a given range was requested. In this machine, *a* is the publication date or that date brought into the first active trading period, *m* is set to 100 intervals (the price vector is rather 101 dimensions), *n* is set 10 minutes, so each dimension in the vector is separated by 10 minutes, and *k* is 1000 or approximately 2.5 days worth of trading data. If for some reason there isn't enough data to fill the *price vector*, then anything that uses it must wait long enough for the vector to be completely filled. Thus we can be assured that all *price vectors* are completely filled.

3.1.3. Predicting and Training

Since we have both data and the ability to parse that data into a usable form, we can now use the machine to make predictions on an arbitrary article and train from that same article. First we will define the mathematical representation and then describe the implementation of the processes.

Let us say that in the initial state of the machine the following is true:

$\forall \text{ word} : w_{\text{word}} = 0^{\left(\frac{m}{n}+1\right)} = 0^{(101)}$, this means for all words the value w_{word} starts as the 0 vector in 101 dimensions. Used as storage for what is known as the *word vector*.

$\forall \text{ word} : c_{\text{word}} = 0$, for all words the value c_{word} starts as 0. Count of times a document has word in it

$\forall \text{ word} : t_{\text{word}} = 0$, for all words the value t_{word} starts as 0. Count of all times *word* has appeared among articles trained from so far.

$C = 0$, the total count of documents trained from so far.

Let us also define the components of an article in a stemmed bag of words model.

$N =$ the total number of words in a document.

$S =$ the set of all words in the article .

$n_{word} =$ the number of times a *word* appears in the article.

$P = \mathbb{R}^{101}$ the *prediction vector* calculated prior to training.

$A = \mathbb{R}^{101}$ the closing *price vector* of the article

We have fully defined all variables that are needed to describe the machine. However, we need to define a function, or rather the possible composing functions, necessary for predictions.

This function is known as term-frequency inverse-document-frequency (TF-IDF). This class of function is used heavily in text mining to give higher weights to words that appear less frequently in some body of articles and lower weights to words that appear in a particular article [5]. Formally we define the functions as:

$$TFIDF(\text{word}) = TF(\text{word}) * IDF(\text{word}), \qquad TF(\text{word}) = \{TF_1(\text{word}), TF_2(\text{word})\}$$

$$TF_1(\text{word}) = \frac{n_{word}}{N}$$

$$TF_2(\text{word}) = \frac{n_{word}^2}{\sum_{k \in S} n_k^2}$$

$$IDF(\text{word}) = \log\left(1 + \frac{C}{1 + c_{\text{word}}}\right)$$

$TF(\text{word})$ can either be $TF_1(\text{word})$ or $TF_2(\text{word})$. We create a machine for both cases to determine the optimal function.

With all this in mind we may finally define how to create a prediction for a given article. Let us say that we are given some arbitrary article then the following calculates P, the *prediction vector* for that article:

```
tfidf(BoWA, word): //Given a bag of words article and a word
      idf = log(1 + C / (1+c_word))
      tf1 = BoWA.n_word / BoWA.N // option 1 term frequency
      //option 2 term frequency
      totalweight = 0
      For wword in BoWA.S:
            totalweight += n_wword^2
      tf2 = n_wword^2 / totalweight
      return tf1*idf
      // or return tf2*idf

makePrediction(BoWA): //Given a bag of words article
      P = (0, 0, …, 0, 0) // start prediction vector as a 0 vector
      For word in BoWA.S:
            P += w_word * tfidf(BoWA, word)
      return P
```

This is equivalent to the weighted average of *word vectors* for all words in the given article where the weight is given by the TFIDF for that word.

Now let us formally define the steps necessary for training from an article. Note that much like the TF function there are two possible training methods, one based on whether an article contains a word and the other based on the total count of words in the article.

```
trainModel(BoWA): //Given a bag of words article
      // BoWA.A waits until A is filled
      For word in BoWA.S:
            //update the word vectors
            w_word = (w_word * c_word + BoWA.A) / (c_word + 1) //Option 1
            w_word = (w_word * t_word + BoWA.A * BoWA.n_word) / (t_word + BoWA.n_word) //Option 2
            //update the word in article count
            c_word = c_word + 1
            //update the total count
            t_word = t_word + BoWA.n_word
      //update the total article count
      C = C + 1
```

The only difference between the two training methods is how they calculate the *word vector*. The first case is normal mean and the second is weighted mean based on the amount of times a word appears in an article. Since there are two possible TF functions and two possible training methods, we have four possible ways of choosing both. Thus we will have four different machines that will be run.

Keeping that in mind we constructed a system that allowed any of the four machines to be run. A relational database, yet again SQLite, was created with a main table where each row is contains an ID, stock ticker, publication date, stemmed bag of words, *price vector*, and the date at which the *price vector* ends. In order to fill this table, we looped through all articles in the article database creating a unique ID for each article based on its source URL and RSS ID, preprocessing the stripped html in the first database, creating the *price vector* for the current article, and then calculating where it would end.

The database also contains a table made of stateful information about all models being run against the current database. Each model is also given access to its own table in the database to store information necessary to create, or fetch, *prediction vectors*. For our scheme it stores all w_{word}, c_{word}, t_{word}, and C in the stateful information as well as the time and id of last article predicted or trained from. This is used to continue training in case it stops for any reason. In the

additionally table each model stores the stock ticker, ID, publication date, the date at which the *price vector* ends, and the *prediction vector* for the article with the corresponding ID.

In order to actually make these predictions, we created two min heaps each composed of all rows. Though, these two heaps really merge into a single min heap. The first heap is dedicated to calculating the *prediction vector* and then writing them to the database. The second heap is meant to handle training the machine. The heap value used by the first heap is the publication date and the heap value for the latter is the end time of the *price vector*. We iterate over the min of both of those and begin to predict or train. This ensures that at no time the machine can cheat (i.e. predict using data that should not have been seen yet).

3.1.4. Post processing

Having created a prediction for each article for a machine, we needed to map these predictions to a scalar usable by both humans and the neural nets. In order to accomplish this, we must first discuss how we may add *prediction vectors* starting at different times. If we have two *prediction vectors* for any stocks, they must both start at some article publication date. Since the *prediction vectors* are essentially time series we can plot the vectors starting at the publication dates and extending for 1000 minutes. Suppose that these two interfere for some amount of time. If we take the portions that overlap and then linearly shift both such that the beginning of each port remains at 0 we can add the two *prediction vectors* to create a *sliced prediction vector*. This *sliced prediction vector* is guaranteed to have between 0 and 101 dimensions. However, if we wanted to restrict it to an exact range we would create a *sliced prediction vector* by adding a *prediction vector* to a fake *prediction vector* with all 0 over the specified range. Thus producing a *sliced prediction vector* of the specified range and values. Let us also define an active article for a time and company. An article is active if for a specific range there is a *sliced prediction vector* that can be created and that article is about the specific company / stock.

With this in hand let us define four techniques that will convert from the raw *prediction vectors* to a scalar. All these techniques are based on a linear regression's slope for a regular or scaled *sliced prediction vector*. Having the scalar based on the slope helps to mitigate potential noise by taking into account the entire performance over 65 minutes. Additionally, all the linear regressions go through the origin, which matches the 0 for the first dimension and ensures that the higher dimensions have more weight in corresponding line.

- Simple mean: The first technique and perhaps the most simplistic is to create an average of all *prediction vectors* for active articles at the point at which the sentiment analysis wants to be made for some time bar. That is add each *prediction vector* and divide by the total count of active articles. Find the slope of this article with a fixed point at (0,0), using linear regression. This slope will be the sentiment value.
- Weighted mean: The second technique is similar to the first, but using weighted means instead. Where the weight is determined by the inverse of the time distance between the prediction point and the publication of the article. This is to give higher weight to articles that came out closer to the prediction point.
- Regression: The third technique is the mean of all slopes of all *sliced prediction vectors* between the specified time. Again all lines must go through the origin.
- Classification: We classify an active article as good if the slope of the *sliced prediction vectors* between the specified times is greater than a given threshold. Likewise, it is bad if it is below a threshold and if it is in between thresholds it is considered neutral. The scalar is produced by the count of good minus the bad divided by the addition of both. This only works if there exists at least 1 good or bad article.

If for the specified range there are no active article to work with, or in the fourth's case no good or bad articles, then the system will merely return 0 stating it has no confidence either way.

3.1.5. Example Sentiment Analysis Machine

Let us take a couple simplistic articles, describing different companies, to go through the process of the machine. First let the machine be in the initial state as described where the vector lengths will be 11 and intervals set at 10 minutes. Starting with the article given by "An up stock is moving upward". If we were to create the stemmed bag of words representation of this article it would be {"up":2, "stock":1 "move":1}; "up" and "upward" share the same stem and the other two words are removed since they are stop words. Now we can try to make a prediction on that article.

$$P = \sum_{\text{word} \in S} w_{word} * \frac{n_{\text{word}}}{N} * \log\left(1 + \frac{C}{1 + c_{\text{word}}}\right)$$

$$\text{word} = \text{"up"}: \ 0^{11} * \frac{2}{4} * \log\left(1 + \frac{0}{1+0}\right) = 0^{11}$$

$$\text{word} = \text{"stock"}: \ 0^{11} * \frac{1}{4} * \log\left(1 + \frac{0}{1+0}\right) = 0^{11}$$

$$\text{word} = \text{"move"}: \ 0^{11} * \frac{1}{4} * \log\left(1 + \frac{0}{1+0}\right) = 0^{11}$$

$$P = 0^{11}$$

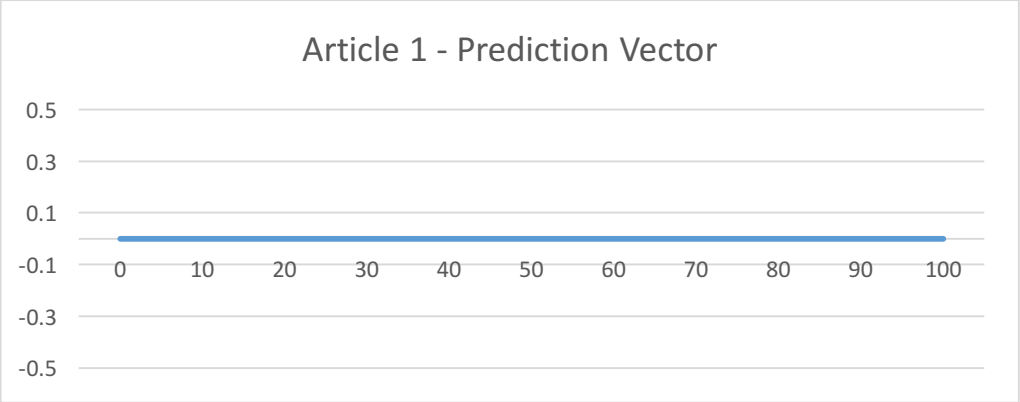

Figure 8 First example article prediction vector

As we can see the prediction for the first article is the zero vector which is to be expected since all *word vectors* are still at their initial state. Now let us say that 100 minutes have passed so we can obtain a full *price vector*.

$$A = (0, 0.5, 1.0, 1.5, 2.0, 2.0, 2.5, 3.0, 3.0, 3.0, 3.0)$$

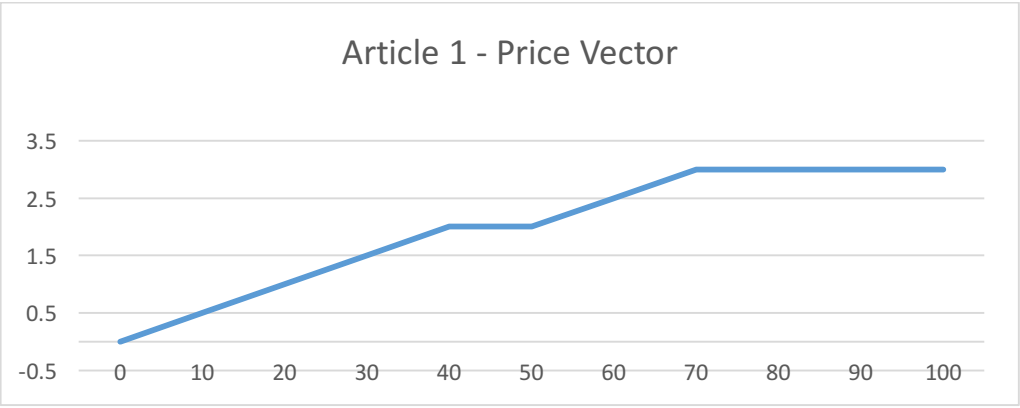

Figure 9 First example article price vector

$$w_{up} = \frac{w_{up} * c_{up} + A}{c_{up} + 1} = \frac{0^{11} * 0 + A}{0 + 1} = (0, 0.5, 1.0, 1.5, 2.0, 2.0, 2.5, 3.0, 3.0, 3.0, 3.0)$$

$$w_{stock} = \frac{w_{stock} * c_{stock} + A}{c_{stock} + 1} = \frac{0^{11} * 0 + A}{0 + 1} = (0, 0.5, 1.0, 1.5, 2.0, 2.0, 2.5, 3.0, 3.0, 3.0, 3.0)$$

$$w_{move} = \frac{w_{move} * c_{move} + A}{c_{move} + 1} = \frac{0^{11} * 0 + A}{0 + 1} = (0, 0.5, 1.0, 1.5, 2.0, 2.0, 2.5, 3.0, 3.0, 3.0, 3.0)$$

$$c_{up} = 1, c_{stock} = 1, c_{move} = 1, t_{up} = 2, t_{stock} = 1, t_{move} = 1\ , C = 1$$

Now let the machine take the article "The company's stock is upward bound" after the training on article 2. The stemmed bag of words representation will be {"compani":1, "stock":1, "up":1, "bind":1}. Which we will use to calculate a prediction for the article.

$$w_{compani} * \frac{n_{\text{compani}}}{N} * \log\left(1 + \frac{C}{1 + c_{\text{compani}}}\right) = 0^{11} * \frac{1}{4} * \log\left(1 + \frac{1}{1 + 0}\right) = 0^{11}$$

$$w_{stock} * \frac{n_{\text{stock}}}{N} * \log\left(1 + \frac{C}{1 + c_{\text{stock}}}\right) = w_{stock} * \frac{1}{4} * \log\left(1 + \frac{1}{1 + 1}\right) = w_{stock} * 0.1014$$

$$w_{up} * \frac{n_{\text{up}}}{N} * \log\left(1 + \frac{C}{1 + c_{\text{up}}}\right) = w_{up} * \frac{1}{4} * \log\left(1 + \frac{1}{1 + 1}\right) = w_{up} * 0.1014$$

$$w_{bind} * \frac{n_{\text{bind}}}{N} * \log\left(1 + \frac{C}{1 + c_{\text{bind}}}\right) = 0^{11} * \frac{1}{4} * \log\left(1 + \frac{1}{1 + 0}\right) = 0^{11}$$

$$P = (0.0, 0.1014, 0.2027, 0.3041, 0.4055, 0.4055, 0.5068, 0.6082, 0.6082, 0.6082, 0.6082)$$

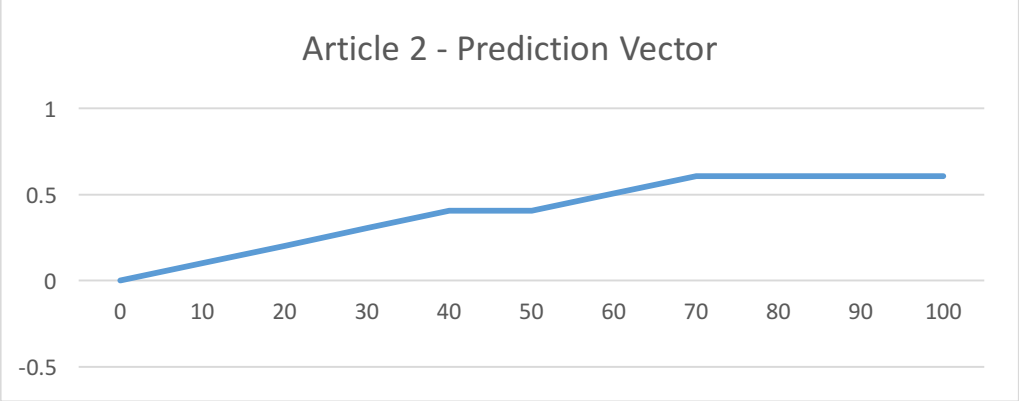

Figure 10 Second example prediction vector

As we can see since we have only trained on a single article the prediction is a simple scaling of the *price vector* from the first article. Over time as individual words change and the machine trains the scaling will become less apparent. Again let us say that the 100 minutes necessary for a valid *price vector* have passed thus we can train the machine on the data.

$$A = (0.0, 1.0, 2.0, 3.0, 3.0, 2.0, 1.0, 0.0, -1.0, -2.0, -3.0)$$

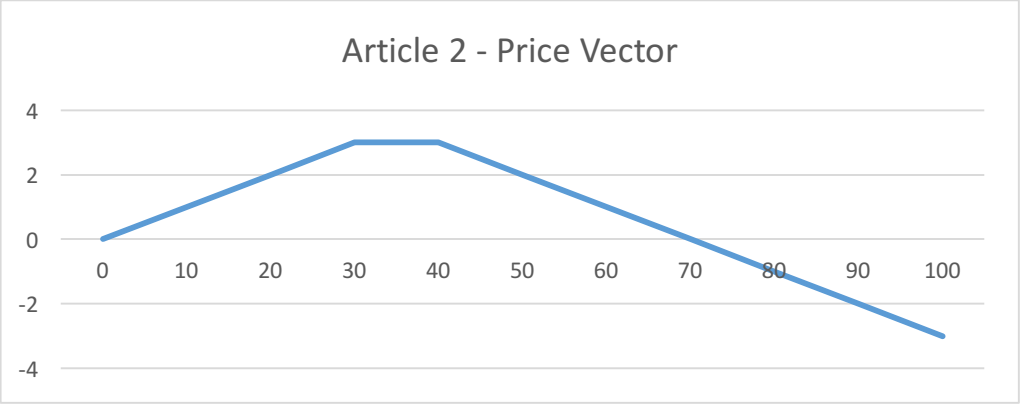

Figure 11 Second example price vector

$$w_{compani} = \frac{0^{11} * 0 + A}{0 + 1} = (0.0, 1.0, 2.0, 3.0, 3.0, 2.0, 1.0, 0.0, -1.0, -2.0, -3.0)$$

$$w_{bind} = \frac{0^{11} * 0 + A}{0 + 1} = (0.0, 1.0, 2.0, 3.0, 3.0, 2.0, 1.0, 0.0, -1.0, -2.0, -3.0)$$

$$w_{stock} = \frac{w_{stock} * 1 + A}{1 + 1} = (0.0, 0.8, 1.5, 2.3, 2.5, 2.0, 1.8, 1.5, 1.0, 0.5, 0.0)$$

$$w_{up} = \frac{w_{up} * 1 + A}{1 + 1} = (0.0, 0.8, 1.5, 2.3, 2.5, 2.0, 1.8, 1.5, 1.0, 0.5, 0.0)$$

$$c_{compani} = 1, c_{bind} = 1, c_{stock} = 2, c_{up} = 2, t_{compani} = 1, t_{bind} = 1, t_{stock} = 2, t_{up} = 3$$

$$C = 2$$

Now let us consider a final article "The company is bound by upward moves ." The article will have a stemmed bag of words {"compani":1, "bind":1, "up":1, "move":1}. And will have the following prediction.

$$w_{compani} * \frac{n_{compani}}{N} * \log\left(1 + \frac{C}{1 + c_{compani}}\right) = w_{compani} * \frac{1}{4} * \log\left(1 + \frac{2}{1 + 1}\right)$$

$$= w_{compani} * 0.173$$

$$w_{bind} * \frac{n_{bind}}{N} * \log\left(1 + \frac{C}{1 + c_{bind}}\right) = w_{bind} * \frac{1}{4} * \log\left(1 + \frac{2}{1 + 1}\right) = w_{bind} * 0.173$$

$$w_{up} * \frac{n_{\text{up}}}{N} * \log\left(1 + \frac{C}{1 + c_{\text{up}}}\right) = w_{up} * \frac{1}{4} * \log\left(1 + \frac{2}{1 + 2}\right) = w_{up} * 0.127$$

$$w_{move} * \frac{n_{\text{move}}}{N} * \log\left(1 + \frac{C}{1 + c_{\text{move}}}\right) = w_{move} * \frac{1}{4} * \log\left(1 + \frac{2}{1 + 1}\right) = w_{move} * 0.173$$

$$P \approx (0.0, 0.5, 1.1, 1.6, 1.7, 1.3, 1.0, 0.7, 0.3, -0.1, -0.5)$$

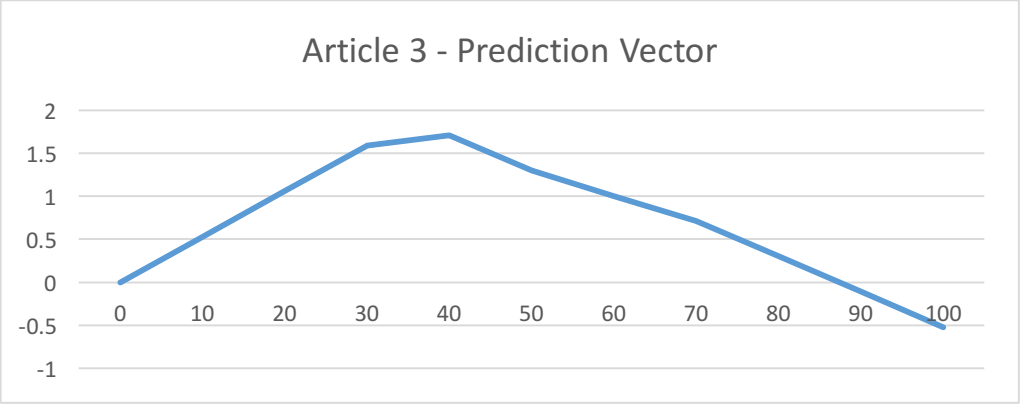

Figure 12 Third example prediction vector

As we can see this new prediction vector takes into acount the effect of the two prior articles. To finish the machine off let us again say that the 100 minutes necessary for a valid *price vector* have passed thus we can train the machine on that data.

$$A = (0.0, 0.0, -0.5, -0.5, -1.0, -1.0, -1.5, -1.5, -2.0, -2.0, -2.0)$$

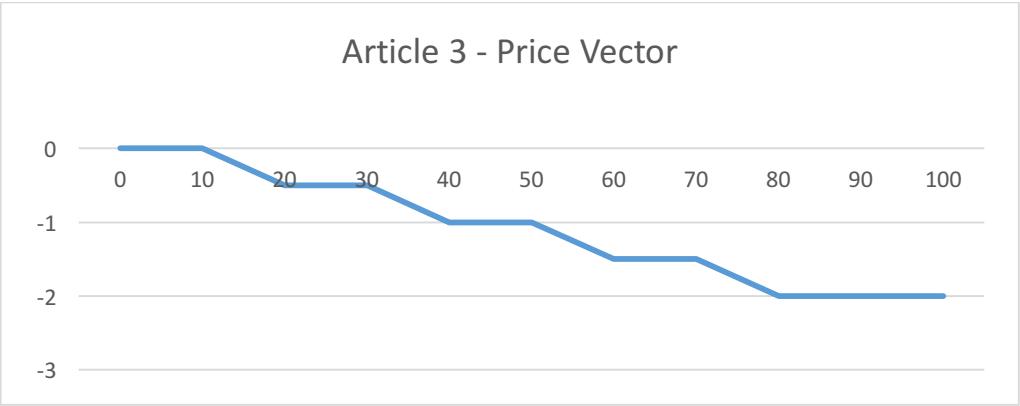

Figure 13 Third example price vector

$$w_{compani} = \frac{w_{compani} * 1 + A}{1 + 1} = (0.0, 0.5, 0.8, 1.3, 1.0, 0.5, -0.3, -0.8, -1.5, -2.0, -2.5)$$

$$w_{bind} = \frac{w_{bind} * 1 + A}{1 + 1} = (0.0, 0.5, 0.8, 1.3, 1.0, 0.5, -0.3, -0.8, -1.5, -2.0, -2.5)$$

$$w_{up} = \frac{w_{up} * 2 + A}{2 + 1} = (0.0, 0.7, 1.2, 1.8, 1.7, 1.0, 0.2, -0.5, -1.3, -2.0, -2.7)$$

$$w_{move} = \frac{w_{move} * 1 + A}{1 + 1} = (0.0, 0.3, 0.3, 0.5, 0.5, 0.5, 0.5, 0.8, 0.5, 0.5, 0.5)$$

$$c_{compani} = 2, c_{bind} = 2, c_{move} = 2, c_{up} = 3, t_{compani} = 2, t_{bind} = 2, t_{move} = 2, t_{up} = 4$$

$$C = 3$$

In order to produce values that may used more readily we must now employ the post processing techniques. We will showcase the generally more complex weighted mean scheme. We are given two arbitrary articles about the same company with the following *prediction vectors* and publication information:

Released: Oct 12th 2015, 8:30 AM

$$P_4 = (0.0, -0.5, -1.0, -1.0, 0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0)$$

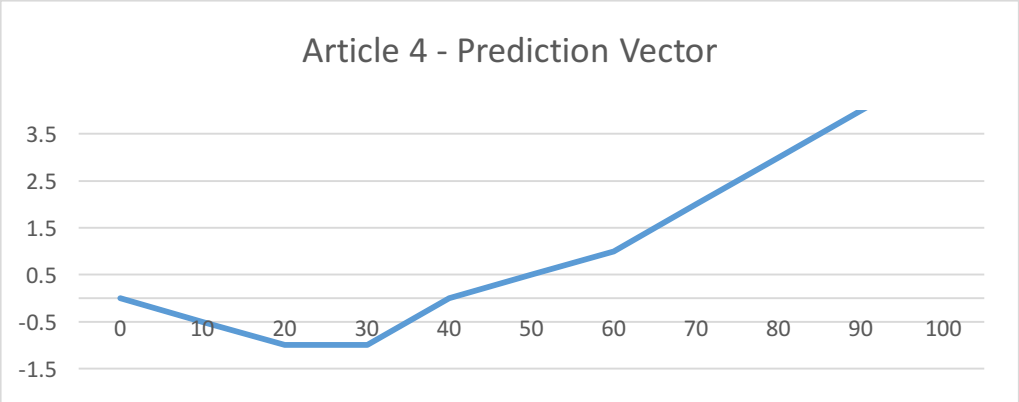

Figure 14 Fourth example article prediction vector

Released: Oct 12th 2015, 9:10 AM

$$P_5 = (0.0, 0.5, 0.5, 0.0, 0.5, 1.0, 1.5, 1.5, 1.0, 1.0, 1.0)$$

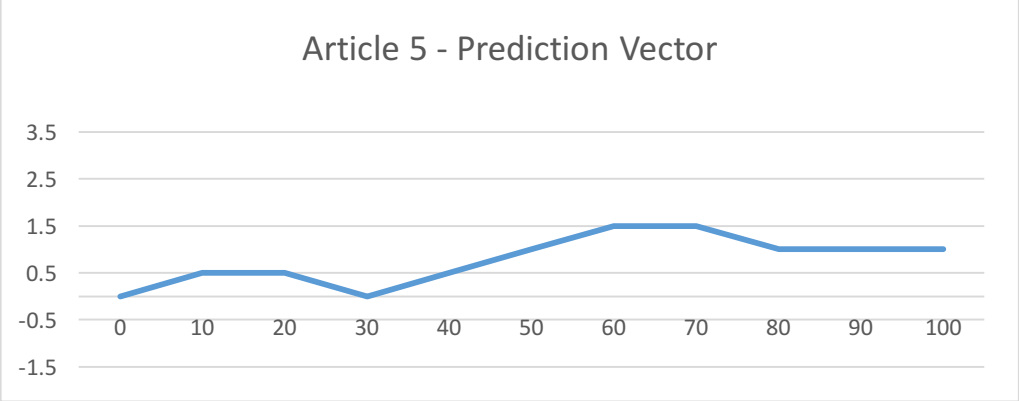

Figure 15 Fifth example article prediction vector

As we can tell from the publication dates there are 60 minutes in which the two intersect, so if we wanted to use the prediction vectors of both we would have to constrain our values within that 60 minutes. So let us say we want to create a prediction value on Oct 12th 2015, 9:30 AM for 30 minutes into the future. We have to create the *sliced prediction vector* for that specific timeframe. As described we take the same real time values from each prediction vector and then rebase them.

$$SP_4 = (1.0 - 1.0, 2.0 - 1.0, 3.0 - 1.0, 4.0 - 1.0) = (0.0, 1.0, 2.0, 3.0)$$
$$SP_5 = (0.5 - 0.5, 0.0 - 0.5, 0.5 - 0.5, 1.0 - 0.5\ ) = (0.0, -0.5, 0.0, 0.5)$$

Now to produce the sentiment value we need to take the weighed average of the two *sliced prediction vector* with weight based on their time since publication.

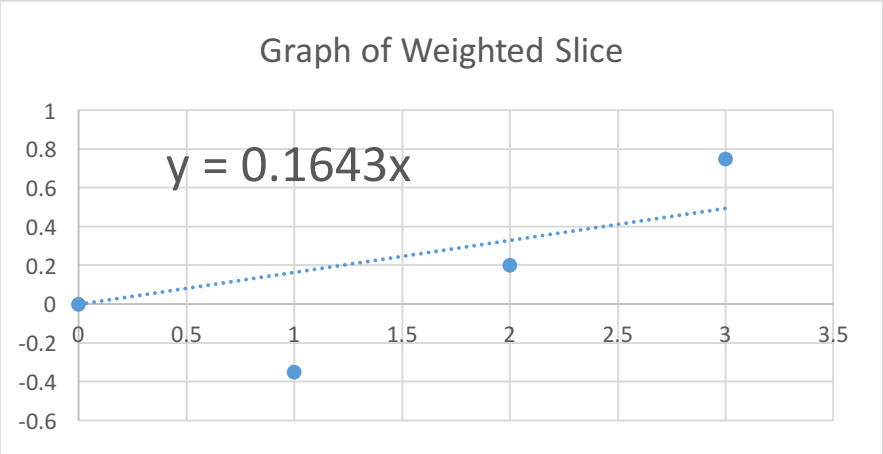

Figure 16 Example graph of a weighted slice

$$\frac{\sum \frac{1}{\Delta t^2} * SP}{\sum \frac{1}{\Delta t^2}} = \frac{\frac{1}{(60*60)^2} * SP_4 + \frac{1}{(20*60)^2} * SP_5}{\frac{1}{(60*60)^2} + \frac{1}{(20*60)^2}} = (0., -0.35, 0.2, 0.75)$$

Now we must run a linear regression on this considering. So the final prediction value for these specfic paramaters would be 0.1643 which would indicate that the machine thinks that the company's stock value will go up.

3.1.6. Evaluation

In order to test the sentiment analysis machine, a sample of 1000 date times were chosen from 10/12/2015 – 3/4/2016. Each time a random ticker from the S&P 100 was also chosen. With a time bar of 65 minutes each of the 4 possible combinations of the model will be run with the following 5 post processing techniques: mean, weighted mean, regression, classification for

good > 0 and bad < 0, and classification for good > .001 and bad < -.001. The best performing scheme amongst the 20 possible will then be used as the input for the neural network.

3.2. Clustering & Risk Minimization

A common technique used in risk minimization is investment diversification, which is the investment in a diverse selection of assets that behave differently such that if a handful of the investments lose some of their value, the return from other investments would compensate for the losses.

In our search to find a diverse selection of stocks, we first used K-Means clustering, a clustering algorithm which unfortunately does not work well with time series data (such as our financial data). In our experience, K-Means with random initialization resulted in extremely different clusters on every run with no consistent pattern. Instead, we used hierarchical agglomerative clustering algorithm with Dynamic Time Warping (DTW) and weighted distance to cluster stocks based on how similarly their prices have behaved in the past.

To determine the number of clusters in K-Means clustering, we used the Elbow method. However, when performing hierarchical agglomerative clustering, we selected a number of clusters such that each cluster had at least 2 stocks and the largest cluster had a maximum of 15 of the S&P 100 stocks to avoid training a single network on an overly large cluster due to hardware limitations constraining our network's size.

Although diversification was the initial motivation behind clustering, we found that our neural networks performed better when trained on a cluster of stocks instead of only on a single stock or on all S&P 100 stocks. Previous research also found that training a neural network on a cluster improved performance [13]. We believe that training a neural network on the combined data of a cluster of stocks helped our network in learning patterns that are consistently useful when making predictions and helped avoid over fitting.

3.3. Neural Network

We decided to use a recurrent neural network variant called Long Short Term Memory (LSTM) for our main prediction system. LSTM networks have been shown to be able to solve

problems that have hundreds of time steps between important events. This was obviously a huge deciding factor when compared to other neural networks which can have trouble learning dependencies between events only twenty time steps apart.

3.3.1. Features & Outputs

Examples or instances were constructed as unique series of 100 consecutive candlesticks from a stock's 65 minute candlestick chart. Initially, each data point or time step in a series consists of a candlestick's high, low, open, and closing price, volume and date. To achieve statistical stationarity, we difference the prices in the input features. A differenced time series sometimes referred to as detrended time series) is computed as the differences between consecutive observations.

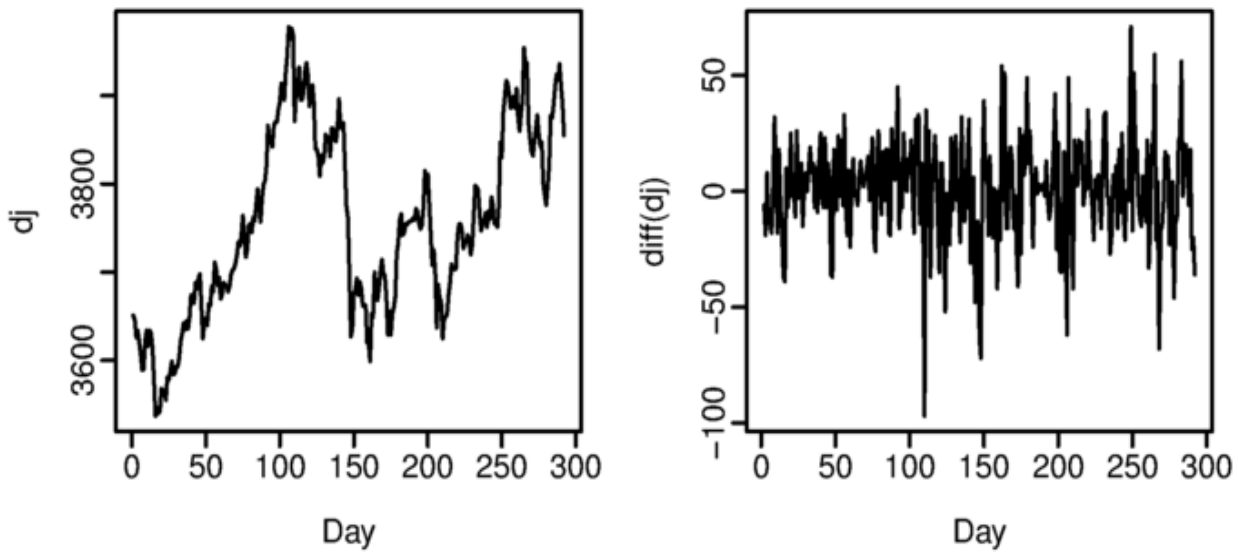

Figure 17: Dow Jones Index on 297 consecutive days (Left). Daily change of Dow Jones Index on 297 consecutive days (Right)

As shown in Figure 17, differencing stabilizes the mean and variance of a time series, which allows a neural network to learn a set of weights that are shared across time steps. Differencing a time series is a technique that is used in training the state of the art neural network in handwritten recognition and even proved useful in previous research that also attempted to forecast stock prices [14].

At each time step, we also use binary features to represent the day of the week (3 inputs), day of the month (5 inputs), month (4 inputs) and time of day (3 inputs) to represent the date. For example, on the 6th of December, 6 (day of the month) is represented as 0b00110 and 12

(month) is represented as 0b1100. We used binarized features for the discrete valued inputs because binarized features are shown to be more robust to noise [15].

We train our system to predict if the closing price of a candlestick on a 65 minute time chart will be higher than its opening price. The system makes a prediction after a new candlestick starts to form. Our system has only a single sigmoid output. An output closer to 1 means that the system predicts that the closing price of the current candlestick is higher than the opening price. The output can be interpreted as a confidence value, where a number really close to 1 or 0 means that the system is very confident in its prediction.

3.3.2. Regularization

Initially, we regularized the network by using Dropout [16], by stabilizing activations [17] and by decorrelating representations [18]. Unfortunately, unlike Dropout, regularization by stabilizing activations and regularization by decorrelating representations had parameters that were too costly for our hardware to optimize and slowed down training more than our timeline could allow.

We also considered RNNDropout [19], a special variant of the Dropout regularization method, which was designed specifically for LSTM. However, our system performed better with traditional Dropout. Ultimately, we decided to use Dropout as our only regularization method.

3.3.3. Activation function

For this project we were initially using Rectified Linear Units [20]for all layers in the neural network. A major problem with training deeper neural networks using back propagation is vanishing or exploding error gradient. Rectified Linear Units (ReLU) do not suffer as greatly from vanishing gradients as sigmoid or hyperbolic tangent sigmoid. However, the network would sometimes start producing NaN (Not A Number) errors when using ReLU due to the network producing incredibly large numbers. We had similar problems using Exponential Linear Units [21]. Ultimately, we decided to use a standard hyperbolic tangent activation function.

3.3.4. Training & Hyperoptimization

	Size	Activation
Layer 1	400	tanh
Layer 2	400	tanh
Output	1	sigmoid

Table 1 Neural network layers

We used 3 LSTM layers. The network consumes a time series of 100 time steps, each consisting of 22 inputs to produce a single sigmoid (value between 0 and 1) prediction.

Due to hardware constraints, we limited the number of parameters of our network to approximately 2 million parameters. Each cluster's dataset consisted of 80,000 to 600,000 examples for training, 3,200 to 24,000 for validation, and 3,200 to 24,000 for testing. We use mini batch gradient descent with nesterov momentum to train a new network on each cluster of stocks. Nesterov momentum has been shown to significantly improve performance on multiple tasks [22].

When training using batch gradient descent, a network is first evaluated on the entire set of training examples before a single update is made to the network's weights. However, when training using mini-batch gradient descent, the network's weights are updated after the network is evaluated on only a small batch of training examples instead of the entire dataset. This training method makes an assumption that the small batch of training examples are a good enough representation of the entire dataset, such that the gradient of mini batch gradient descent and batch gradient descent generally move the network in the same direction. Due to limited GPU memory, the maximum batch size we could select was around 200. We ran one epoch with a mini batch size of 200 and summed up the error gradients, then repeated it with batch sizes [1, 5, 10 .. 195] and compared the gradients, in an attempt to find a small batch size that has gradients similar to the 'true gradient' (the gradient when using batch gradient descent). Ultimately, we selected batch size 40 as a happy medium between frequent weight updates, time per epoch and noise. Additionally, the gradient norm was scaled down if it was larger than 5 to mitigate the effect of exploding gradient [23].

We initialized our learning rate at 0.1 and after every training epoch we increased the learning rate by 10% if performance on the validation set improves or reduced it by 50% otherwise, down to a minimum of $1 * 10^{-8}$. This is a commonly used inexpensive method to find a good learning rate. As for momentum, we arbitrarily selected the initial value 0.9.

In an attempt to fight overfitting, if the network fails to improve its performance on the validation set 10 times in a row, we restore the network's weights to its previously best performing weights and raise our batch size in increments of 20 up to our maximum batch size. The intuition is that a larger batch size would reduce noise and help the network settle at a local minima.

4. RESULTS

4.1. Sentiment Analysis Results

During the period between 10/12/2015 and 3/4/2016 we were able to collect 294500 unique html articles from Google's and Yahoo's RSS feeds. Our preprocessing could successfully extract the article content from 179400 of those html articles. On average each stock was written about 2.37 times per day, including weekends, and 20.57 times per week. The exact distribution of articles by day can be found in Table 2. Overall, these figures show that there are substantial news articles for the sentiment machine to not only learn but also make predictions at the daily rate.

Article Percentage by Day

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
18.37%	19.91%	19.68%	18.92%	15.11%	3.72%	4.29%

Table 2 Article distribution over days

Additionally, we collected the historic stock price data during the same period, 10/12/2015 and 3/4/2016, by using Google's finance API. During this interval the API provided an average of 44300 points of minute bar data (time, opening price, minimum price, maximum price, closing price, and volume) for each stock. In total, there were 22.15 million points of minute bar data gathered during the collection period.

To evaluate the accuracy of the sentiment value we created full predictions for the four specified models and the following five post processing techniques: weighted mean, simple mean, regression, classification with both thresholds set at 0.0, and classification with thresholds set at 0.01 and -0.01 for good and bad articles, respectively. However, these post processing methods return a scalar, so we us a trading strategy such that if a post process method's return value is above a threshold then we "should buy" the specified stock at the time.

"Should Buy" Accuracy (0.0 Threshold)

	Term Frequency 1		Term Frequency 2	
	Train 1	Train 2	Train 1	Train 2
Weighted Mean	54.81%	54.15%	53.35%	53.78%
Simple Mean	55.41%	55.63%	54.65%	55.20%
Regression	54.97%	55.29%	54.52%	55.50%
0.0 Classification	55.00%	53.42%	55.10%	54.98%
0.01 Classification	54.82%	54.68%	55.28%	55.50%

Table 3 0 threshold accuracy

Table 3 shows the accuracy of a "should buy" command from the trading strategy for all the different models with the different post processing functions. The threshold used in the trading strategy is set to 0.0, stating that any post process return value above 0 should be bought. Each scheme is better than a random trading strategy, i.e. a strategy where it randomly predicts whether to buy. For the same times and stocks used in the evaluation of the sentiment value, the random strategy was only accurate 51.5% of the time. Since we are dealing with finances, we also wanted to look at the profitable of this strategy. Table 4 shows the average percentage profit made for every "should buy" command, again with a 0.0 threshold. The random strategy meanwhile had a profit of 0.01%. We can see that all all of the sentiment models perform better than following a purely random strategy in some cases meagerly better and others significantly better.

"Should Buy" Profit % (0.0 Threshold)

	Term Frequency 1		Term Frequency 2	
	Train 1	Train 2	Train 1	Train 2
Weighted Mean	0.06%	0.03%	0.05%	0.06%
Simple Mean	0.12%	0.11%	0.12%	0.11%
Regression	0.10%	0.10%	0.10%	0.12%
0.0 Classification	0.10%	0.10%	0.11%	0.11%
0.01 Classification	0.09%	0.10%	0.11%	0.11%

Table 4 0 threshold average profit % per trade

These results have shown the base case of the system, where if it might go up we trade on it. However, we could improve the prediction results by selecting optimal thresholds (likely non 0.0) based on the first half, chronologically, of the evaluation data and then testing on the latter

half of the data. More specifically, we had to maximize accuracy while ensuring that there were enough "should buy" orders, i.e. the "should buy" command should be issued no less than approximately 20% of the time. The data used for optimization and the data used for testing was pulled from times between 10/12/2015 and 12/12/2015 and between 12/12/2015 and 3/4/2016, respectively. (Note all threshold values are scaled up by a factor of 100). The optimizations were primarily done by hand, allowing human feel to choose optimal thresholds based on experimental results.

"Should Buy" Thresholds

	Term Frequency 1		Term Frequency 2	
	Train 1	Train 2	Train 1	Train 2
Weighted Mean	0.0020	0.0035	0.0024	0.0026
Simple Mean	0.0050	0.0052	0.0040	0.0046
Regression	0.0044	0.0049	0.0049	0.0056
0.0 Classification	0.61	0.72	0.71	0.72
0.01 Classification	0.60	0.71	0.71	0.72

Table 5 Optimized threshold values

"Should Buy" Accuracy (Optimized Threshold)

	Term Frequency 1		Term Frequency 2	
	Train 1	Train 2	Train 1	Train 2
Weighted Mean	59.63%	57.75%	61.48%	59.84%
Simple Mean	58.21%	57.81%	59.14%	58.62%
Regression	58.33%	56.06%	60.00%	55.22%
0.0 Classification	55.00%	60.42%	59.42%	53.70%
0.01 Classification	55.42%	62.50%	59.21%	53.52%

Table 6 Optimized accuracy

"Should Buy" Profit % (Optimized Threshold)

	Term Frequency 1		Term Frequency 2	
	Train 1	Train 2	Train 1	Train 2
Weighted Mean	0.07%	0.13%	0.16%	0.11%
Simple Mean	0.31%	0.30%	0.26%	0.26%
Regression	0.28%	0.27%	0.30%	0.27%
0.0 Classification	0.19%	0.22%	0.22%	0.23%
0.01 Classification	0.16%	0.19%	0.20%	0.17%

Table 7 Optimized average profit % per trade

Table 5 shows the optimal thresholds that we found. There is a distinct scale difference between classification and the other post process methods, since classification is based on a -1 to 1 scale while the others are based on slope. Also, by looking at the optimal thresholds we can see the regardless of the model chosen the optimal threshold for a post process method will be about the same. The corresponding "should buy" accuracy and profit can be found in Table 6 and Table 7.

As can be seen, these optimizations dramatically increase both the accuracy and the profit of any combination of model and post process method. The optimized weighted mean performs at the slightly best rate of 59%. Additionally, it can be argued that the weighted mean with the first model, term frequency 1 and training method 1, would produce the best results since it has a higher number of "should buy" commands. As such the weighted mean with term frequency 1 and training method 1 were used as input into the neural network, since the primary focus is prediction accuracy.

However, the weighted mean has the lowest profitability, though it still beats a random strategy. Therefore, it may not be the best candidate if used by itself. The simple mean and regression techniques both perform at roughly 0.28% profit and 58% accuracy. So in general it can be argued that these two methods would be preferred over the weighted mean.

4.2. Clustering

By running the hierarchical agglomerative clustering algorithm with Dynamic Time Warping and weighted distance on 5 years of word of time series data we were able to cluster the S&P 100 into the 13 clusters found in Appendix D. As expected a number of clusters contain

companies in the same sector. More surprising, however is the unequal size of the clusters. A number of clusters only have two stocks while a large portion of stocks are in clusters with 10+ stocks. This would seem to indicate that large portions of the S&P 100 follow similar patterns.

4.3. Neural Network Results

We ran back tests on unseen data, spanning back up to 5 years, for the LSTM neural networks by themselves and with using sentiment values. Table 8 shows the results of those tests. It is clear to see that the sentiment value increases the accuracy of the neural network, though marginally. Though there is one cluster, #3, where the sentiment value actually decreases the accuracy. However, the advantage gained by using the sentiment value far outweighs that small drawback.

"Should Buy" Accuracy

Cluster #	Neural Network	Neural Network & Sentiment Analysis
1	72.80%	72.90%
2	71.40%	71.90%
3	74.70%	74.60%
4	59.70%	60.20%
5	58.20%	58.50%
6	63.40%	63.70%
7	60.50%	60.70%
8	75.80%	76.40%
9	67.50%	67.90%
10	54.70%	55.00%
11	51.40%	51.40%
12	54.60%	55.10%
13	55.10%	55.20%

Table 8 Neural network based prediction accuracy

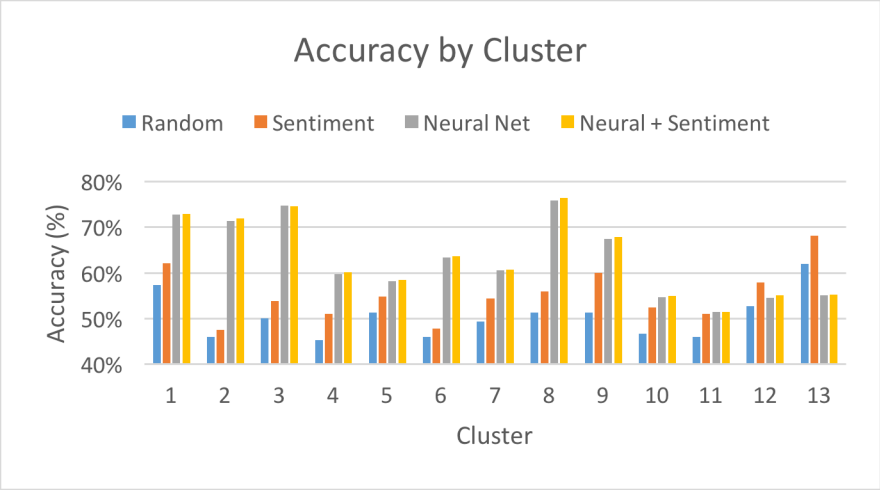

Figure 18 Accuracy by clusters

Figure 18 shows a side by side comparison between the neural network, optimized sentiment, the combination of the two, and the random strategy as defined as randomly deciding whether to buy. With this comparison, it can be seen that the sentiment analysis should not be used by itself since in comparison to the neural network based predictions it barely performs better than the random strategy. We can also see that the neural network outperforms the random strategy by an extremely large gap. The neural network with sentiment is accurate up to 77% in comparison to the random strategy which is only accurate up to 62%

While accuracy is important in investing, the key objective is to to make money. To that end, we wanted to test the profitability of the system. The test uses a simple trading strategy that buys short and sells long based solely on the prediction value of the system. Figure 19 shows the result of that test over a five year period with unseen data with a $7.50 trading fee and $100000 principle account balance. In order to compare this to normal market conditions we looked at two other indicators. The first is the "S&P 500" which is the average weighted stock price for stocks on the S&P 500 during the same period. The second is the "Buy & Hold" strategy where instead of buying and selling every 65 minutes it buys once at the beginning and the holds onto it for five year period.

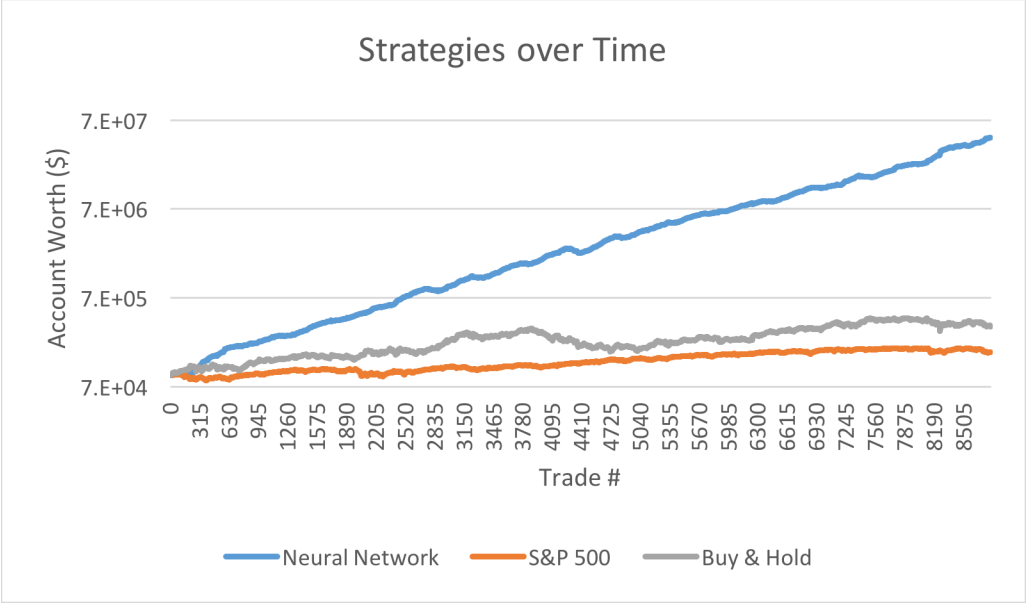

Figure 19 Trading strategies over 5 year time

At first glance this seems to performs extremely well, being an order of magnitude higher than the "Buy & Hold" or "S&P 500." However, to confirm this we must look at a financial metrics for each to determine their actual performance. The first metric we should consider is the net profit of each of the schemes. The second metric is the Sharpe Ratio which measures a risk-adjusted return. More precisely, it is the the mean return in excess of some risk free base over the volatility of the returns. For our purposes we used 5% annually as the risk free base. The next metric is the max drawdown, which is defined as the highest percentage of money lost from one point to another in the future. The final metric we considered was the value at risk, which is a measurement of the financial risk for a system. Table 9 shows the values that we calculated for each of the metrics. Looking at the net profit it clear that our system made an extremely large amount in comparison to either of the two.

	Net Profit	Sharpe Ratio	Max Drawdown	Value at Risk
Neural Network	47463.39%	0.245	-11.51%	0.38%
Buy & Hold	255.32%	0.019	-44.92%	1.15%
S&P 500	82.00%	0.022	-18.64%	0.27%

Table 9 Financial metric comparison

5. CONCLUSION & FUTURE WORK

Experimental results show our LSTM-based system performed incredibly well, predicting whether stock prices will be higher 65 minutes into the future with up to ~77% accuracy on entire clusters of stocks. In comparison, randomly predicting whether a stock would be higher only achieves up to 62% accuracy. Results also show that the use of the sentiment value as an input to the network marginally increases the accuracy of the neural network.

As an experimental application of the system in the stock market, we developed a simple trading strategy that longs or shorts a stock based on the system's predictions. The strategy had a Sharpe Ratio of 0.245, which indicates a higher expected return to risk ratio than both the "buy & hold" and the "S&P 500." Although a Sharpe ratio below 1 is still considered too small for most traders [24], our simple trading strategy still significantly outperformed other strategies. We remain confident in our system's predictions and optimistic about its potential use in future real world applications.

5.1. FUTURE WORK

The bag of words model utilized by our sentiment analysis, is simplistic for the complex task of text mining. A large amount of information, context mainly, is lost in the transformation from article to bag of words. A more suitable representation that maintains the context of the textual information could drastically increase the accuracy of the sentiment analysis and the system as a whole. One possibility is a neural network based on fixed length feature representations, which has been shown to outperform several textual representations [25]. Additionally, our sentiment analysis approach could be improved by adding a decay function, such that gradually over time the system forgets the effects articles, making the *word vectors* a weighted average with higher weight for articles recently released. This would be critical to a long term system since words could change connotation depending on the market conditions.

Hardware limitations were a huge factor in multiple aspects of this project. We did not possess the computational power necessary to hyperoptimize the network's input features, hidden layers size, training parameters and regularization methods. Along with that, the chosen timebar, 65 minutes, was used because it easily divided a trading day into equal section of time.

In the future, an optimization should be made on the length of the timebar. Finally, we also would have liked to experiment with boosting techniques for the neural network (e.g. [26]).

Another key area where this system could be improved on is the use of the output value. At this point, the system only produces a value between 0 and 1 representing how confident it is that a stock will go up in the next 65 minutes. A trading system needs to use that value to make decisions. From the results based on the optimal cluster, we made a substantial profit with even an arguably simple trading system. Moving forward, a smarter trading system could easily outperform the one we made.

We also wished to test the system in a realistic setting. One way to test the real profitable of the system could be to run it as a walk forward or paper trading system. Paper trading is a test state where the system sells and buys but does not exchange money [27]. This minimum risk test method has the benefit of being able to be run in real time, as if it was a real trading system. Using this it would be possible to determine the true accuracy and profitability as a fully forward looking system.

6. BIBLIOGRAPHY

[1] TradeStation, "TradeStation Platform," 2016. [Online]. Available: http://www.tradestation.com/trading-technology/tradestation-platform. [Accessed 20 Apr 2016].

[2] B. M. Barber, Y.-t. Lee, Y.-j. Liu and T. Odean, "Do Day Traders Rationally Learn About Their Ability," 2010.

[3] B. Yoon and Y. Park, "A text-mining-based patent network: Analytical tool for high-technology trend," *High Technology Management Research,* vol. 15, no. 1, pp. 37-50, 2004.

[4] V. Kalyanaraman, S. Kazi, R. Tondulkar and S. Oswal, "Sentiment Analysis on News Articles for Stocks," in *Asia Modelling Symposium*, 2014.

[5] N. Oliveira, P. Cortez and N. Areal, "Automatic Creation of Stock Market Lexicons for Sentiment Analysis Using StockTwits Data," in *IDEAS*, New York City, 2014.

[6] R. P. Schumaker and H. Chen, "Textual Analysis of Stock Market Prediction Using Breaking Financial News: The AZFinText System," *ACM Trans. Inf. Syst.,* vol. 27, no. 12, pp. 1-19, Feb 2009.

[7] A. Graves and J. Schmidhuber, "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks," in *Neural Information Processing Systems*, 545-552, 2008.

[8] C. Lu and X. Tang, "Surpassing Human-Level Face Verification Performance on LFW with GaussianFace," 20 Dec 2014. [Online]. Available: http://arxiv.org/abs/1404.3840. [Accessed 12 Feb 2016].

[9] N. Lomas, "Google Says Its Self-Driving Cars Drive Better Than You," 12 May 2015. [Online]. Available: http://techcrunch.com/2015/05/12/google-says-its-self-driving-cars-drive-better-than-you/. [Accessed 12 Feb 2016].

[10] A. Ng, "Stanford University Autonomout Helicopter," [Online]. Available: http://heli.stanford.edu. [Accessed 12 Feb 2016].

[11] J. Dorrier, "Exponential Medicine: Deep Learning AI Better Than Your Doctor at Finding Cancer," 11 Nov 2015. [Online]. Available: http://singularityhub.com/2015/11/11/exponential-medicine-deep-learning-ai-better-than-your-doctor-at-finding-cancer/. [Accessed 15 Feb 2016].

[12] V. Pham, T. Bluche, C. Kermorvant and J. Louradour, "Dropout improves Recurrent Neural Networks for Handwriting Recognition," 10 Mar 2014. [Online]. Available: http://arxiv.org/abs/1312.4569. [Accessed 20 Feb 2016].

[13] D. Enkea, M. Grauerb and N. Mehdiyevb, "Stock Market Prediction with Multiple Regression, Fuzzy Type-2 Clustering and Neural Networks," in *Complex Adaptive Systems*, Chicago, 2011.

[14] A. M. Rathera, A. Agarwala and V. Sastryb, "Recurrent neural network and a hybrid model for prediction of stock returns," 11 Dec 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417414007684. [Accessed 21 Feb 2016].

[15] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv and Y. Bengio, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," 17 Mar 2016. [Online]. Available: http://arxiv.org/abs/1602.02830. [Accessed 16 Feb 2016].

[16] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 3 July 2012. [Online]. Available: http://arxiv.org/abs/1511.07289. [Accessed 5 October 2015].

[17] D. Krueger and R. Memisevic, "REGULARIZING RNNS BY STABILIZING ACTIVATIONS," in *International Conference on Learning Representations*, Caribe Hilton, San Juan, 2016.

[18] M. Cogswell, F. Ahmed, R. Girshick, L. Zitnick and D. Batra, "Reducing Overfitting in Deep Networks by Decorrelating Representations," in *International Conference on Learning Representations*, Caribe Hilton, San Juan, 2016.

[19] T. Moon, H. Choi, H. Lee and I. Song, "RNNDROP: A NOVEL DROPOUT FOR RNNS IN ASR," 2015. [Online]. Available: http://www.stat.berkeley.edu/~tsmoon/files/Conference/asru2015.pdf. [Accessed 2 October 2015].

[20] E. E. o. R. e. A. i. C. Network, "Xu, Bing ;Wang, Naiyan ; Chen,Tianqi; Li, Mu;," 27 Nov 2015. [Online]. Available: https://arxiv.org/abs/1505.00853. [Accessed 12 Apr 2016].

[21] D.-A. Clevert, T. Unterthiner and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," 22 Febuarary 2016. [Online]. Available: http://arxiv.org/abs/1511.07289. [Accessed 23 Febuarary 2016].

[22] Y. Bengio, N. Boulanger-Lewandowski and R. Pascanu, "Advances in Optimizing Recurrent Networks," 14 Dec 2012. [Online]. Available: http://arxiv.org/abs/1212.0901. [Accessed 18 Feb 2016].

[23] R. Pascanu, T. Mikolov and Y. Bengio, "On the difficulty of training Recurrent Neural Networks," 16 Feb 2016. [Online]. Available: http://arxiv.org/abs/1211.5063. [Accessed 23 Feb 2016].

[24] "Sharpe Ratio," [Online]. Available: http://www.investopedia.com/terms/s/sharperatio.asp?o=40186&l=dir&qsrc=999&qo=investopediaSiteSearch. [Accessed 12 April 2016].

[25] Q. Le and T. Mikolov, "Distributed Representations of Sentences and Documents," in *International Conference on Machine Learning*, Bejing, 2014..

[26] R. E. Schapire, "Explaining AdaBoost," in *Empirical Inference*, Berlin, Springer Berlin Heidelberg, 2013, pp. 37-52.

[27] "Paper Trade," [Online]. Available: http://www.investopedia.com/terms/p/papertrade.asp. [Accessed 20 April 2016].

[28] S. Bird, E. Loper and E. Klein, *Natural Language Processing with Python,* O'Reilly Media Inc., 2009.

[29] S. Van der Walt, S. C. Colbert and G. Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation," *Computing in Science & Engineering,* vol. 13, no. 2, pp. 22 - 30, March 2011.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, V, J. erplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *JMLR,* vol. 12, pp. 2825-2830, 2011.

[31] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley and Y. Bengio, *Theano: new features and speed improvements,* D. L. a. U. F. L. N. 2. Workshop, Ed., NIPS, 2012.

[32] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley and Y. Bengio, "Theano: a CPU and GPU Math Expression Compiler," in *Proceedings of the Python for Scientific Computing Conference*, Austin, 2010.

[33] P. Uhr, J. Zenkert and M. Fathi, "Sentiment Analysis in Financial Markets," in *IEEE International Conference on Systems, Man, and Cybernetics*, San Diego, 2014.

[34] M. Makrehchi, S. Shah and W. Liao, "Stock Prediction Using Event-based Sentiment Analysis," in *IEEE/WIC/ACM International Conferences on Web Intelligence (WI) and Intelligent Agent Technology (IAT)*, Atlanta, 2013.

[35] "Dynamic time warping," American Pink, [Online]. Available: http://america.pink/dynamic-time-warping_1339527.html. [Accessed 12 Nov 2015].

[36] "Tutorial," [Online]. Available: http://www.tradersedgesystems.com/tutorials/. [Accessed 20 April 2016].

[37] "What is a Japanese Candlestick," [Online]. Available: http://www.babypips.com/school/elementary/japanese-candle-sticks/what-is-a-japanese-candlestick.html. [Accessed 20 April 2016].

7. APPENDICES

Appendix A Source Code

The complete source code can be found at the following url:

https://github.com/sjamos/Financial-MQP-2015

However, the historic data we used cannot be included with the source due to copyright. We could also not include any of the sentiment analysis data since the data sets were to large to fit with the hosting site.

Appendix B List of Third Party Libraries Used

- Python
    - BeautifulSoup
    - DateUtil
    - FeedParser
    - Lasagne
    - Newspaper
    - NLTK [28]
    - NumPy [29]
    - pyTZ
    - Requests
    - Sklearn [30]
    - Theano [31] [32]

Appendix C EXAMPLE RSS FEED DATA


```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">
  <channel>
    <title>News for Apple Inc. - Google Finance</title>
    <description>News for Apple Inc. - Google Finance</description>
    <link>http://www.google.com/finance?q=NASDAQ:AAPL&amp;amp;client=news-
rss&amp;amp;ei=VZghV4jQHIeB2Aab46iIAg</link>
    <image>
      <title>News for Apple Inc. - Google Finance</title>
      <url>/finance/s/M4XFgDr56u4/images/logo_us.gif</url>
      <link>http://www.google.com/finance?q=NASDAQ:AAPL&amp;amp;client=news-
rss&amp;amp;ei=VZghV4jQHIeB2Aab46iIAg</link>
    </image>
    <item>
      <title>Apple Inc. (AAPL) Shares Tumble, Wiping Almost $50B Off Value Following Weak</title>
      <link>http://www.ibtimes.com/apple-inc-aapl-shares-tumble-wiping-almost-50b-value-following-
weak-first-quarter-2360439</link>
      <guid isPermaLink="false">tag:finance.google.com,cluster:52779096541869</guid>
      <pubDate>Wed, 27 Apr 2016 14:07:54 GMT</pubDate>
      <description>Investors jumped on an Apple stock-buying opportunity Wednesday, pulling the price
back from a more than 8 percent plunge since the company reported Tuesday its first quarterly sales drop
in 13 years and missed Wall Street</description>
    </item>
    …
    <item>
      <title>QQQ: Profit From Apple Inc.&amp;amp;#39;s Failure (AAPL)</title>
      <link>http://investorplace.com/2016/04/qqq-profit-from-apple-aapl-failure/</link>
      <guid isPermaLink="false">tag:finance.google.com,cluster:52779095972947</guid>
      <pubDate>Wed, 27 Apr 2016 15:11:15 GMT</pubDate>
      <description>Tech lovers beware. The generals are being taken out and shot one by one. And the
Nasdaq is suffering under the onslaught. First it was Netflix, Inc. (NFLX), then Microsoft Corporation
(MSFT) and Alphabet Inc (GOOG, GOOGL) bit the dust. And today Apple</description>
    </item>
  </channel>
</rss>
```

Appendix D GOOGLE HISTORIC PRICE DATA FORMAT

```
 EXCHANGE%3DNASDAQ
MARKET_OPEN_MINUTE=570
MARKET_CLOSE_MINUTE=960
INTERVAL=60
COLUMNS=DATE,CLOSE,HIGH,LOW,OPEN,VOLUME
DATA=
TIMEZONE_OFFSET=-240
a1460035800,110.05,110.13,109.94,109.95,281703
1,109.8725,110.14,109.85,110.11,206135
2,109.96,110,109.87,109.87,140909
3,110.0096,110.17,109.92,109.96,178861
4,109.792,110,109.6,110,270163
5,109.83,109.98,109.76,109.792,203361
6,110.06,110.1,109.78,109.82,204226
7,110.245,110.27,110.03,110.05,225070
8,109.91,110.25,109.91,110.25,206933
9,109.73,109.94,109.7,109.91,291477
10,109.82,109.85,109.73,109.74,173793
11,109.88,109.89,109.79,109.813,126870
12,109.9,109.96,109.84,109.88,135006
13,109.8963,109.925,109.88,109.9,113565
14,109.9601,109.99,109.88,109.895,96487
15,109.895,109.96,109.88,109.96,124241
16,109.82,109.92,109.79,109.89,126239
17,109.66,109.8201,109.65,109.81,142667
18,109.64,109.75,109.64,109.67,148275
19,109.69,109.78,109.55,109.64,168302
...
365,104.36,104.36,104.33,104.35,129551
366,104.3767,104.41,104.35,104.3507,141661
367,104.32,104.42,104.32,104.38,134364
368,104.3024,104.32,104.28,104.32,101674
369,104.31,104.35,104.3,104.3099,139387
370,104.255,104.31,104.23,104.305,162050
371,104.24,104.29,104.23,104.255,203829
372,104.22,104.27,104.2,104.23,210616
373,104.1847,104.26,104.17,104.22,105913
374,104.1801,104.21,104.14,104.18,159327
375,104.211,104.23,104.16,104.19,126741
376,104.2001,104.24,104.16,104.21,197864
377,104.25,104.26,104.19,104.204,193759
378,104.265,104.27,104.21,104.25,152899
379,104.1889,104.27,104.18,104.26,181424
380,104.2101,104.28,104.1638,104.18,277427
381,104.1838,104.2562,104.1838,104.21,246361
382,104.22,104.24,104.18,104.18,207740
```

Appendix E YAHOO HISTORIC PRICE DATA FORMAT

```
 uri:/instrument/1.0/AAPL/chartdata;type=quote;range=15d/csv
ticker:aapl
Company-Name:Apple Inc.
Exchange-Name:NMS
unit:MIN
timezone:EDT
currency:USD
gmtoffset:-14400
previous_close:104.3500
range:20160407,1460035800,1460059200
...
range:20160427,1461763800,1461787200
Timestamp:1460035800,1461787200
values:Timestamp,close,high,low,open,volume
close:96.6000,112.3000
high:96.7900,112.3900
low:95.7000,112.2500
open:96.0000,112.2950
volume:0,17167400
1460036099,109.8900,110.1700,109.6000,109.9500,1748800
1460036340,109.8080,110.2700,109.7000,109.8900,1166600
1460036640,109.9100,109.9900,109.7900,109.8200,632600
1460036999,109.6600,109.9200,109.5500,109.8900,723300
1460037240,109.7250,109.7300,109.5800,109.6700,407100
1460037541,109.9200,109.9800,109.7201,109.7300,473800
1460037841,110.0500,110.1161,109.7801,109.9300,480500
1460038140,110.0450,110.1700,109.9800,110.0500,356500
1460038499,109.7500,110.0499,109.7500,110.0400,352700
1460038740,109.7800,109.8000,109.6450,109.7500,394200
...
1461786299,97.6159,97.7100,97.4400,97.4600,923800
1461786540,97.6900,97.8000,97.6100,97.6200,812600
1461786840,97.6250,97.7700,97.6200,97.6900,1058400
1461787199,97.8100,97.8200,97.6100,97.6300,2251300
1461787200,97.8200,97.8200,97.8200,97.8200,31700
```

Appendix F S&P 100 BY CLUSTER

S&P 100 by Cluster			
Company	**Symbol**	**Sector**	**Cluster**
Capital One Financial	COF	Financials	1
General Motors	GM	Consumer Discretionary	1
Johnson & Johnson	JNJ	Health Care	1
The Coca Cola Company	KO	Consumer Staples	1
Lilly (Eli) & Co.	LLY	Health Care	1
MetLife Inc.	MET	Financials	1
Nike	NKE	Consumer Discretionary	1
PepsiCo Inc.	PEP	Consumer Staples	1
Philip Morris International	PM	Consumer Staples	1
Schlumberger Ltd.	SLB	Energy	1
Simon Property Group Inc	SPG	Financials	1
AT&T Inc	T	Telecommunications Services	1
Time Warner Inc.	TWX	Consumer Discretionary	1
United Parcel Service	UPS	Industrials	1
Wells Fargo	WFC	Financials	1
Bank of America Corp	BAC	Financials	2
Celgene Corp.	CELG	Health Care	2
American International Group, Inc.	AIG	Financials	3
The Bank of New York Mellon Corp.	BK	Financials	3
Berkshire Hathaway	BRK-B	Financials	3
Cisco Systems	CSCO	Information Technology	3
Mastercard Inc.	MA	Information Technology	3
McDonald's Corp.	MCD	Consumer Discretionary	3
Mondelez International	MDLZ	Consumer Staples	3
Medtronic plc	MDT	Health Care	3
Morgan Stanley	MS	Financials	3
QUALCOMM Inc.	QCOM	Information Technology	3
Raytheon Co.	RTN	Industrials	3
Verizon Communications	VZ	Telecommunications Services	3
Citigroup Inc.	C	Financials	4
Exelon Corp.	EXC	Utilities	4

Facebook	FB	Information Technology	4
General Dynamics	GD	Industrials	4
Alphabet Inc Class C	GOOG	Information Technology	4
Goldman Sachs Group	GS	Financials	4
Texas Instruments	TXN	Information Technology	4
General Electric	GE	Industrials	5
Monsanto Co.	MON	Materials	5
Norfolk Southern Corp.	NSC	Industrials	5
PayPal	PYPL	Information Technology	5
AbbVie	ABBV	Health Care	6
American Express Co	AXP	Financials	6
Caterpillar Inc.	CAT	Industrials	6
Comcast A Corp	CMCSA	Consumer Discretionary	6
CVS Caremark Corp.	CVS	Consumer Staples	6
Intel Corp.	INTC	Information Technology	6
Lowe's Cos.	LOW	Consumer Discretionary	6
Occidental Petroleum	OXY	Energy	6
Pfizer Inc.	PFE	Health Care	6
Procter & Gamble	PG	Consumer Staples	6
Southern Co.	SO	Utilities	6
United Health Group Inc.	UNH	Health Care	6
Visa Inc.	V	Information Technology	6
Allergan plc	AGN	Health Care	7
Boeing Company	BA	Industrials	7
Costco Co.	COST	Consumer Staples	7
EMC Corp.	EMC	Information Technology	7
Ford Motor	F	Consumer Discretionary	7
FedEx Corporation	FDX	Industrials	7
Twenty-First Century Fox Class B	FOX	Consumer Discretionary	7
Twenty-First Century Fox Class A	FOXA	Consumer Discretionary	7
Gilead Sciences	GILD	Health Care	7
Alphabet Inc Class A	GOOGL	Information Technology	7
3M Company	MMM	Industrials	7
Target Corp.	TGT	Consumer Discretionary	7
United Technologies	UTX	Industrials	7
Home Depot	HD	Consumer Discretionary	8
Kinder Morgan	KMI	Energy	8
Priceline.com Inc	PCLN	Consumer Discretionary	8

Apple Inc.	AAPL	Information Technology	9
Accenture plc	ACN	Information Technology	9
Allstate Corp	ALL	Financials	9
Anadarko Petroleum Corp	APC	Energy	9
BIOGEN IDEC Inc.	BIIB	Health Care	9
Bristol-Myers Squibb	BMY	Health Care	9
Chevron Corp.	CVX	Energy	9
Dow Chemical	DOW	Materials	9
Emerson Electric Company	EMR	Industrials	9
Honeywell Int'l Inc.	HON	Industrials	9
International Bus. Machines	IBM	Information Technology	9
Merck & Co.	MRK	Health Care	9
Amazon.com Inc	AMZN	Consumer Discretionary	10
BlackRock	BLK	Financials	10
Colgate-Palmolive	CL	Consumer Staples	10
ConocoPhillips	COP	Energy	10
The Walt Disney Company	DIS	Consumer Discretionary	10
Devon Energy Corp.	DVN	Energy	10
Oracle Corp.	ORCL	Information Technology	10
Union Pacific	UNP	Industrials	10
Walgreens Boots Alliance	WBA	Consumer Staples	10
Abbott Laboratories	ABT	Health Care	11
Amgen Inc	AMGN	Health Care	11
JPMorgan Chase & Co.	JPM	Financials	11
Microsoft Corp.	MSFT	Information Technology	11
U.S. Bancorp	USB	Financials	11
Du Pont (E.I.)	DD	Materials	12
Halliburton Co.	HAL	Energy	12
Starbucks Corp.	SBUX	Consumer Discretionary	12
Lockheed Martin Corp.	LMT	Industrials	13
Altria Group Inc	MO	Consumer Staples	13