Stock Jump Prediction and Feature Set Comparison

FEATURE VECTOR COMPARISON OF NEURAL NETWORK STOCK JUMP PREDICTION MODELS

By John Heisey, B.Sc.

A Thesis Submitted to the School of Graduate Studies in the Partial Fulfillment of the Requirements for the Degree Master of Science

McMaster University

Master of Science (2019)

Hamilton, Ontario (Department of Computational Science and Engineering)

TITLE: FEATURE VECTOR COMPARISON OF NEURAL NETWORK STOCK JUMP PREDICTION MODELS

AUTHOR: John Heisey (McMaster University)

SUPERVISOR: Dr. Tom Hurd

NUMBER OF PAGES: xi, 70

Lay Abstract

This project compares three unique data sets originating from the NASDAQ electronic stock market. These data sets are used as input data to create separate predictive models. These models are compared based on their ability to predict stock price jumps. The results indicate that each of the three data sets have very similar capabilities. Because of the similar capabilities, it is concluded that the data set that is the easiest to develop should be considered for future experiments.

Abstract

This project assesses the viability of three feature vector sets derived from NASDAQ limit order book data in predicting the occurrences of stock price jumps in five separate high market cap stocks. Two of the three feature sets, which are novel and incorporate sentiment and statistical moment features, are compared to the third feature set established by Kercheval and Zhang 2015. The feature vectors were derived from two intraday limit order book data sets for all five stocks processed to 30 second and 5 minute time intervals. A midprice jump classification technique was applied to each stock data set in each time interval, to classify whether or not a jump occurred in every time step. A long short-term memory (LSTM) neural network model was then trained on these classified data sets, for which the three feature vectors were used as input, and the predictive results compared. The three feature sets produced the similar predictive capabilities achieving precision scores between 0.71 and 0.18, and F1 scores between 0.65 and 0.16. These findings suggest that the feature set with the lowest data and processing requirements can be used to the same effect as the other two, reducing processing requirements and improving efficiency.

Acknowledgements

First and foremost I would like to thank my project supervisor Dr. Tom Hurd, who was always helpful and provided insightful and useful feedback throughout the project. I would also like to thank my fellow McMaster graduate student colleagues, particularly Hassan Chehaitli and Pritpal Matharu, who were very helpful when discussing the computational challenges of the project. Lastly I would like to thank my family and friends for their support and advice throughout my career in academia.

Contents

List of Figures

List of Tables

Dedicated to my parents

Chapter 1

Introduction

The primary objective of this project is to classify intraday stock midprice jumps of five high market cap stocks in the NASDAQ market using high-frequency limit order book data. Upon completion of jump classification, neural network models are trained using three unique feature vectors to predict these midprice jumps, and their resulting performance metrics are compared.

This project seeks to continue research conducted by Mäkinen 2017 regarding jump detection and prediction by utilizing a recurrent neural network layer schematic proposed by Tsantekidis et al. 2017, and introduces two novel feature vector inputs for comparison with the feature vector input proposed by Kercheval and Zhang 2015 for time series prediction in a limit order book setting. The capabilities of the two proposed feature vectors, one which incorporates daily sentiment metrics and one which is a reduced version possessing limit order book moments, are compared to the feature vector of Kercheval and Zhang 2015, which possesses data extracted from both the raw order book data and order message data.

The questions this project seeks to address are:

1. How effectively do the feature sets proposed in this project compare to that of Kercheval and Zhang 2015 for prediction results?

2. Is there benefit to utilizing the statistical moments of the order book instead of the raw order book data?

3. How do the 30 second and 5 minute time interval models compare?

The outline of this project is as follows: Chapter two provides theoretical background of the various methods and concepts incorporated in this project, while providing a review of the most recent academic literature pertaining to each subject. These subjects begin with a description of an electronic market limit order book and the mechanics involved. This is then followed by a background of neural network predictive models, as well as their applications in financial time series prediction. Next, a background of the standard models used is examined to describe stock price dynamics, and the incorporation and classification of price 'jumps' in these models. Lastly, the concept of sentiment analysis and its applications in a financial framework are presented, alongside the proposed use of agglomerating the limit order book data into statistical moment features for feature vector use.

Chapter three presents a detailed account of the extensive data acquisition and processing methods. Included in these methods are important steps such as jump classification for different time scales, conversion of event data into a uniform time series, and extraction of feature vector inputs. Chapter four describes the neural network model construction process, as well as the unique cross validation techniques necessary for classification problems with a time dependence. The results of the project are provided in chapter five, where comparisons are made between the three separate feature sets, and the two different time scales of time series data used. The results of the jump detection process are also presented in this chapter. The final chapter summarizes the results and proposes potential future directions to continue research on the subject.

Chapter 2

Background and Review of Current Literature

2.1 The Limit Order Book

The limit order book is a longstanding and important tool utilized in stock markets. Its core purpose is to record and maintain a list of limit orders, whose fulfillment priority depends on the limit order's price. For multiple orders with an identical price, the fulfillment priority goes to the order placed first in the queue. Traditionally, limit order books were manually updated, which placed a limit on the efficiency at which orders are queued and executed. In the past decade, this process at most major trading venues has been almost entirely automated. This has increased the efficiency at which external events can be incorporated into the current price, but has also given rise to trading strategies which take advantage of the new automated efficiency.

Currently, limit order books for large electronic exchanges are updated on a nanosecond-order timescale and consist of two main types of orders - market orders and limit orders. A market order is an order to buy or sell shares immediately at the best available price, while a limit order defines a lowest price to sell at (ask) or a highest price to buy at (bid), which may or may not be executed depending on whether or not the market reaches that pre-specified price (Cartea et al. 2015). At any point in time, the order book structure consists of n levels of the best bid and best ask prices for a given stock, along with their respective order volumes (Figure 2.1).

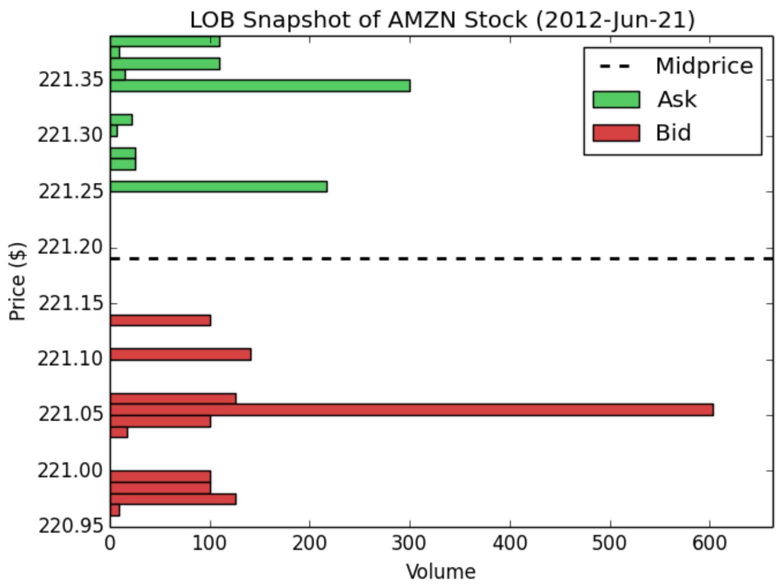
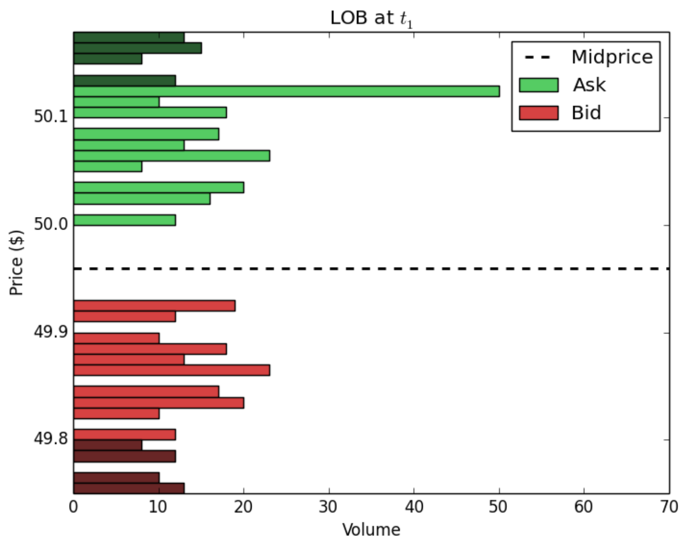


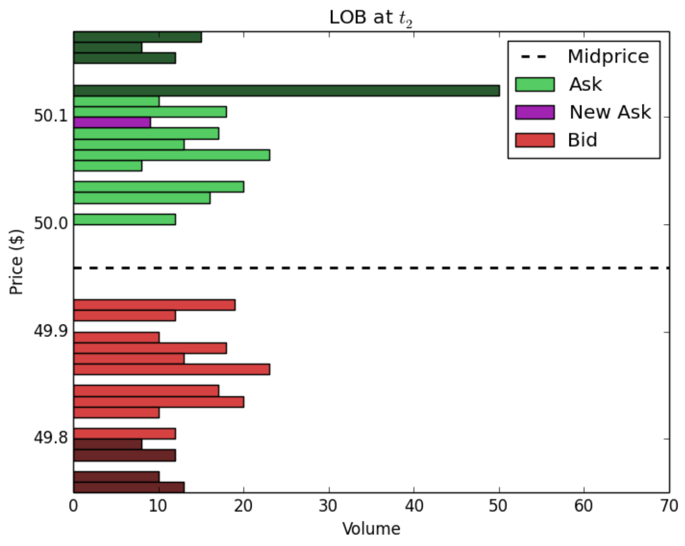
FIGURE 2.1: An example of the first 10 best bid and ask levels and their respective volumes at some instant t for AMZN stock.

When a market order to buy or sell a stock is placed, the order is filled beginning with the best available bid or ask orders ($n = 1$). If the volume of the market order exceeds the available volume of the best bid or ask limit order, the next best available price is then used. This process is then repeated until all the available limit orders are executed - this is known as “walking the book” (Cartea et al. 2015). If the stock being purchased is highly liquid, or the market order is relatively small in volume, the order can be executed with minimal price drift. However if the stock is illiquid or the market order is large in size, considerable price drift from the current mid price is possible (Biais et al. 1995). For an order book with n levels where n is large, there is a decreasing probability that a limit order l will be executed as $l \to n$. Additionally, when dealing with large time spans of multilevel high frequency intraday order data, the issue of data size and training times has to be addressed when building a predictive model. For these reasons, the best 10 bid and ask prices and their respective volumes ($n = 10$) for all points in time will be the focus for the modelling and feature detection/prediction in this project.

While choosing $n = 10$ helps mitigate classic concerns with training large swathes of time series data, it introduces additional considerations when calculating useful LOB metrics. Taking Figure 2.2 as an example, which displays two snapshots of a hypothetical limit order book for a stock at times t_1 and t_2. At some time t where $t_1 < t < t_2$ a small volume limit order is placed which shifts the orders with less desirable prices away from the best ask. If one wishes to calculate the volume weighted mean of the 10 best ask prices for this stock, there is a large difference between the volume weighted mean between t_1 and t_2, while there is little probability (depending on the liquidity and volatility of the stock) that this new order placed at time t will affect the price movement of the stock in question. While this is a potential concern regarding predictive potential, it can be mitigated by other features present in the neural network inputs explained in Chapter 3.



(A) LOB at t_1



(B) LOB at t_2

FIGURE 2.2: An example of how a 'shift' in the 10 best bid/ask prices (light red/light green) can affect metrics when limiting scope to $n = 10$. Between t_1 & t_2, a new limit order is placed (purple) which shifts a high volume ask order to orders outside of the scope of observation (dark green/dark red) causing a large change in calculated moments from t_1 to t_2.

2.2 Neural Networks

Neural networks in their most general form are a computational learning technique implemented in classification/regression, clustering, and prediction problems. The most basic setup for classification of K-classes begins with a vector X containing the original inputs (otherwise known as a "feature" vector). This vector can contain anything from literal numerical quantities of the subject in question (e.g. security midprices and limit order volumes like in Sirignano 2018 and Mäkinen 2017), to more abstract numerical interpretations of the classification task at hand (e.g. word vectorization of text as in Zhou et al. 2016). These numerical values are commonly normalized or scaled to unit length. The network ends with K target measurements Y_k, where K is specified beforehand. Existing between the feature vector and target measurements is at least one "hidden layer" (Figure 2.3), which is a layer of M neurons[1] Z_m derived from linear combinations of the feature vector X:

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X),\ m = 1, \ldots, M$$

where $\sigma(\nu)$ is the activation function (Friedman et al. 2001). The activation function is commonly chosen to be the sigmoid function, hyperbolic tangent function, or the rectified linear unit (ReLU):

$$\text{Sigmoid :} \quad \sigma(\nu) = \frac{1}{1 + e^{-\nu}}$$

$$\text{Hyperbolic tangent :} \quad \sigma(\nu) = \frac{e^{2\nu} - 1}{e^{2\nu} + 1}$$

[1]Also commonly known as "nodes" or "units"

$$\text{Rectified linear unit :} \quad \sigma(\nu) = max(0, \nu)$$

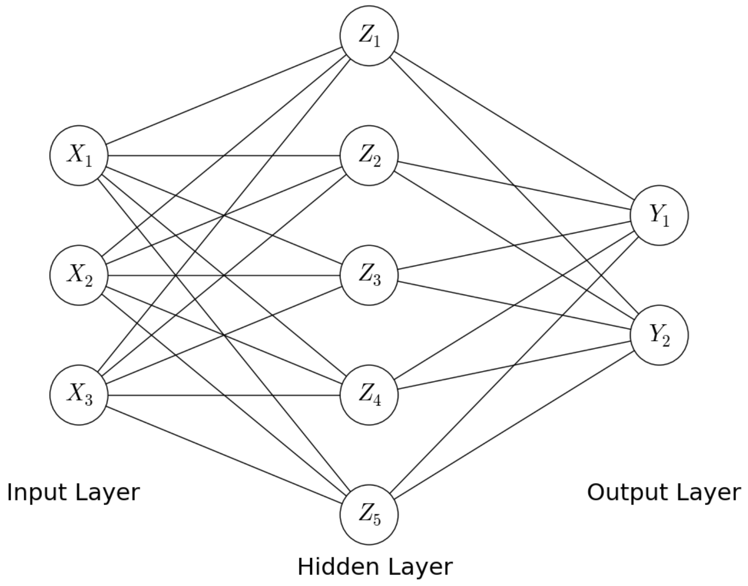


FIGURE 2.3: An example node schematic of a single hidden layer, feed-forward, neural network.

These activation functions are situated within the hidden layer neurons (Figure 2.4). Each function takes a combination of numerical inputs $X_1 \cdot \alpha_1, \ldots, X_N \cdot \alpha_N$, which are the previous layers' output multiplied by their respective weights α_i. The final input is an optional bias component α_0 which provides a trainable constant value which can aid in the learning process. The role of the activation function is to introduce non-linearity to the neuron's output Y, and to the model as a whole.

A model conducts the learning process, forward propagation of data, and backward propagation of errors. All weights α_N are initially randomly assigned. As the input passes through the model, the output is observed and its classifications are compared to the correct classifications to obtain measurements of error from the output nodes. These errors are then propagated *backwards* through the network

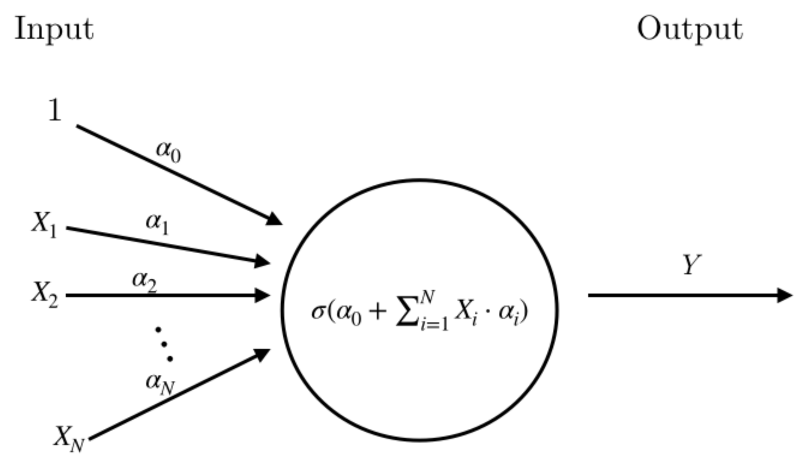


FIGURE 2.4: A diagram of the inputs and outputs of a single artificial neuron with activation function $\sigma(\nu)$.

to calculate the gradients, its at this step an optimization method is chosen to minimize the value of a loss function. Choice of both optimizer and loss function tend to be dependent on the learning problem and data used, but all follow the same general concepts.

The loss function is a function which measures the inconsistency between a predicted value $\hat{Y}$, and the actual value Y. As the model trains, a minimization of the loss function value indicates an increase in the robustness and predictive ability of the model. The loss function can be represented as:

$$\mathcal{L}(\alpha) = \frac{1}{n}\sum_{i=1}^{n} L(y^i, f(x^i, \alpha))$$

where y^i are the target values, and $f(x^i, \alpha)$ is the activation function with arguments x^i and α pertaining to the sample input and model parameters, respectively (Vapnik 1999). There are a multitude of loss functions that have been shown to perform better or worse for a variety of machine learning problems (Janocha and

Czarnecki 2017), but a particular loss function that has been shown to effectively work for a binary classification problem is binary cross entropy. Cross entropy, defined as:

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}[y^i \log(\hat{y}^i) + (1 - y^i)\log(1 - \hat{y}^i)] \tag{2.1}$$

measures the divergence between two probability distributions. Minimizing the cross entropy increases the similarities between the two distributions. The use of cross entropy as a loss function is commonly paired with the use of a sigmoid activation function when dealing with binary classification. Substituting a sigmoid activation function into (2.1) grants:

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}[y^i \log(\sigma(\alpha \cdot \mathbf{x})) + (1 - y^i)\log(1 - \sigma(\alpha \cdot \mathbf{x}))] \tag{2.2}$$

Considering one sample, the partial derivative of (2.2) with respect to α is:

$$\frac{\partial \mathcal{L}}{\partial \alpha} = (y - \sigma(\alpha x)) \cdot \mathbf{x} \tag{2.3}$$

where the $\sigma'(\alpha x)$ term is eliminated. Due to this cancellation, the learning rate of the model depends on the difference between predicted value and actual value, so a large difference between predicted and actual produces a fast convergence, while a small difference produces a slower convergence (Murphy 2012).

While a good choice of loss function is important for properly training a model, the right choice of optimizer can substantially increase the rate of model learning.

The most basic optimizer is standard gradient descent, which updates the model parameters through the algorithm:

$$\alpha = \alpha - \eta \cdot \nabla \mathcal{L} \tag{2.4}$$

where η is the pre-specified rate of learning, and $\nabla \mathcal{L}$ is the gradient of the loss function. While this method is entirely functional, there is a need to specific an efficient learning rate. If the learning rate is too large, local and global minima can be "overshot", leading to slow convergence, and if the learning rate is too small, it can take a long time to converge to an acceptable minimum.

To improve this system, many adaptations have been made to reduce dependency on the learning rate and increase overall training efficiency. Duchi et al. 2011 produced the "Adagrad" algorithm, which allows the learning rate η to adapt in a parameter-specific manner. It updates the parameters by making larger changes for infrequent parameters, and smaller changes for more frequent parameters. Tieleman and Hinton 2012 introduced "RMSProp" which improved upon Adagrad's tendency to radically reduce learning rates in late-stage training, by dividing the gradient by a running decaying average of its recent magnitudes. Kingma and Ba 2014 further improved on the optimization techniques used by RMSProp with their "Adaptive Moment Estimation (Adam)" algorithm. In addition to storing a decaying average of (now bias-corrected) past gradient magnitudes, Adam also stores a decaying average of bias-corrected past gradients. The authors show empirical evidence that Adam works well in comparison to the aforementioned methods (Figure 2.5), and is the current default optimizer used for the well known machine learning API Keras (Chollet et al. 2015).

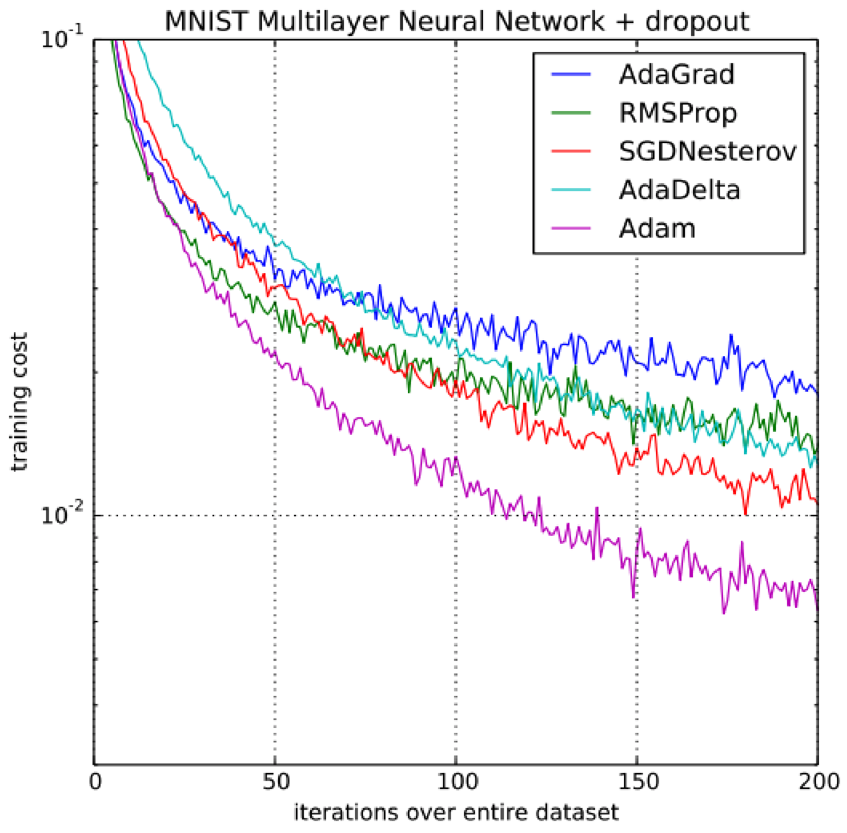


FIGURE 2.5: Comparison of the Adam optimization algorithm to other leading algorithms by training cost for the MNIST 'Handwritten digits' dataset (from Kingma and Ba 2014).

While optimization algorithms can greatly increase the training efficiency, there still exists the potential issue of a model "overfitting". This occurs when the model starts to incorporate residual variation in the training data to its parameters, reducing its ability to predict future observations reliably (Burnham and Anderson 2003). This issue can be mitigated in a neural network setting by a technique known as "dropout" (Srivastava et al. 2014). For each forward/backward propagation iteration, the dropout technique will randomly remove neurons along with their associated weights. This forces the model to spread out the weights amongst the other neurons, and prevents the model from focusing too much on a specific feature in the feature set. Due to its simplicity and efficacy, dropout layers are commonly used in a variety of neural network models.

Aside from the options for activation functions and optimization techniques which are necessary for any neural network model, there are various ways to advance the classic neural network setup itself, such as the convolutional neural network and the deep neural network. For time series data such as limit order book data, and chronologically dependent data like text and audio data, a recurrent neural network setup can prove beneficial.

2.2.1 Recurrent Neural Networks and Long Short-term Memory

A recurrent neural network (RNN) in its most general form is a model where the connections between neurons are cyclical. This creates a feedback loop where each node at some time step takes input from previous nodes (Figure 2.6). This method of sequential connection between nodes has shown to be very effective in machine learning tasks possessing sequential data, due to the ability for the network to update weights based on previous states of the system.

A RNN in its most elementary form suffers from phenomena known as the “vanishing” and “exploding” gradients. This occurs when backpropagating errors across many time steps, and the error signals from prior time steps either exploding

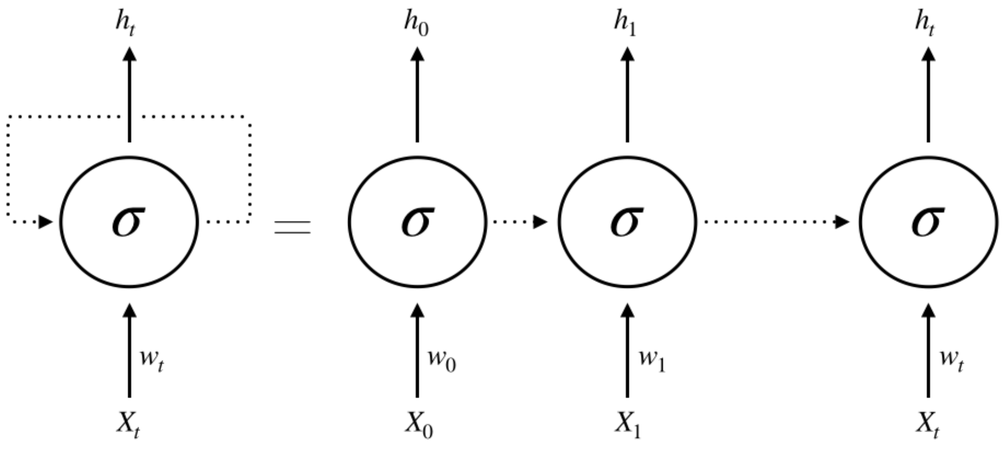


FIGURE 2.6: The cyclical structure of a recurrent neuron with input X, weight w, activation function σ, and output h.

or vanishing[2] due to the exponential dependence on the weight size (Hochreiter et al. 2001).

Hochreiter and Schmidhuber 1997 proposed a method to remove this problem called “Long Short-term Memory” (LSTM). The method maintains the same general recurrent structure and computational complexity, but adds additional operations to the repeating module which enable the network to effectively remember long-term dependencies. These additional operations consist of a “forget gate” operation, an “output gate” operation, and an “update gate” operation, governed by the following equations:

$$f_t = \sigma(w_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.5}$$

[2]The error will ‘explode’ if the weight size is greater than 1, and will ‘vanish’ if less than 1.

$$\begin{aligned} i_t &= \sigma(w_i \cdot [h_{t-1}, x_t] + b_i), \\ \tilde{C}_t &= \tanh(w_C \cdot [h_{t-1}, x_t] + b_C), \\ C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \end{aligned} \tag{2.6}$$

$$\begin{aligned} o_t &= \sigma(w_o \cdot [h_{t-1}, x_t] + b_o), \\ h_t &= o_t * \tanh(C_t) \end{aligned} \tag{2.7}$$

The first step in the order of cell operations is the forget gate (2.5), this operation takes the current time step input x_t as well as the previous time step output h_{t-1} along with their respective weights and optional bias through a sigmoid activation function to produce a value $f_t \in [0, 1]$. An element-wise multiplication is then done between f_t and the previous cell state C_{t-1}, if the value obtained is 0, it is removed from C_{t-1}, whereas if the value is 1 it is completely let through (Gers et al. 1999). The second step is the update gate (2.6), where the output i_t decides which values will be updated, and $\tilde{C}_t$ consists of a vector of new candidate values with potential to be added to the cell state C_t. The previous cell state C_{t-1} is now updated to the new cell state C_t by summing the products of the forget gate output f_t with the previous state C_{t_1}, and the input gate output i_t with the candidate cell state $\tilde{C}_t$. The final output gate (2.7) utilizes a sigmoid activation layer to decide what parts of the cell state to output (this is the closest component to the standard recurrent node), then produces the final output h_t by multiplying the output layer o_t with the output of a hyperbolic tangent function[3] applied to the updated cell state C_t.

This sequence of operations allows an LSTM network to achieve complicated sequential tasks such as handwriting recognition (Graves and Schmidhuber 2009), music generation (Boulanger-Lewandowski et al. 2012), and financial forecasting (Tsantekidis et al. 2017).

2.3 Stock Price Modeling and Prediction

Alongside the prerequisite of understanding the technical details of the limit order book and predictive modelling methods, there is a need to understand how stock prices and markets are modelled in order to better predict specific events. Malkiel and Fama 1970 have presented the hypothesis that existing markets can be classified into three potential levels of market efficiency: weak, semi-strong, and strong. A weak market efficiency states that the market's rates of return are independent, meaning there is no dependence of future rates on past rates. This implies that technical analysis of past patterns utilized by traders is irrelevant in determining future returns. A semi-strong market efficiency dictates that all new publicly available information is absorbed quickly into the price of stocks, and that participants in the market make trading decisions after this information is released - implying that participants cannot capitalize on this new information. A semi-strong classification possesses the weak classification properties as well. A strong market efficiency builds off of the conditions for semi-strong efficiency, stating that

[3]$\tanh(C_t)$ is used instead of just C_t so that $h_t \in [-1, 1]$

private information as well as public information pertaining to a stock is quickly absorbed into the market price, effectively preventing any investor from profiting off of the majority of participants even with the possession of new, private, information. A strong market efficiency encompasses the weak and semi strong classifications, possessing their consequences as well.

In all of the above classifications some degree of efficient market is assumed. This means that even in the weakest form of market efficiency, strategies built upon past information should be rendered entirely ineffective. There has been much research published arguing the contrary, in particular the strong market hypothesis. Lin and Howe 1990 show that insider trading in the OTC/NASDAQ market can be profitable, albeit difficult due to high bid-ask spreads. Furthermore, Jaffe 1974 tracks 200 large firms over several months of market activity, and concludes that use of insider information was indeed profitable - further arguing against the strong-market form of the efficient market hypothesis.

There is still ongoing debate regarding the validity of semi-strong and weak market efficiency. Irrespective of efficiency level, both suggest that it is not possible to develop predictive strategies utilizing past information. Regardless, attempts are consistently made to develop models with predictive capabilities using historical data, including utilizing recent developments in machine learning models. Kercheval and Zhang 2015 proposed an effective support vector machine using a feature vector derived from equity market limit order book and message data to predict future limit order book dynamics in a high frequency setting. Prior to that, Fletcher and Shawe-Taylor 2013 achieved profitable predictive accuracy in foreign exchange markets using a multi-kernel support vector machine approach.

Stemming from the efficient market hypothesis is the assumption that the price of a stock S follows geometric Brownian motion, from which the equation for the expected value of a stock price S_t at time T can be presented:

$$S_T = S_0 \exp\left[(\mu - \frac{\sigma^2}{2})T + \sigma\epsilon\sqrt{T}\right] \tag{2.8}$$

where μ and σ respectively correspond to the stock's expected rate of return and volatility, and $\epsilon \sim N(0,1)$. This equation has been accepted as an accurate way to model stock price dynamics (Hull 2003), however the model neglects an important observed phenomena - the tendency for stock prices to occasionally "jump" quickly in one particular direction in a short period of time.

2.4 Jump Component Introduction and Detection

One of the first applications of a "jump diffusion" model, meaning a mixture model of a jump process and a diffusion process, was in the field of physics to explain the phenomenon of atomic particles jumping between oscillatory and directed motion (Singwi and Sjölander 1960). Merton 1976 was the first to apply the jump diffusion model in a financial context to explain sudden discontinuities in the random variation of stock prices. This model consisted of the standard model shown in

(2.8) with an additional jump component that follows a Poisson process:

$$d \log S(t) = \mu(t)dt + \sigma(t)dW(t) + Y(t)dJ(t) \tag{2.9}$$

where the additional term $J(t)$ can be defined as a non-homogeneous Poisson-type jump process with jump size $Y(t)$ whose mean and standard deviation are $\mu_y(t)$ and $\sigma_y(t)$ (Lee and Mykland 2007). The non-homogeneity of the Poisson process is introduced to incorporate deterministic events in the real markets which can affect the rate and intensity of jumps, such as company earnings reports for returns, and macroeconomic reports for index returns.

With the jump component developed and incorporated into the model, various methods have been proposed to detect when the jump component is present in financial time series. Two non parametric methods were considered for this project, that of Huang and Tauchen 2005 and Lee and Mykland 2007. The primary difference between the two methods being that Huang and Tauchen 2005 utilized the realized variance of the stock price as an estimator for instantaneous volatility to identify jumps, and Lee and Mykland 2007 utilized the realized bipower variation. The advantages of the use of realized variance is that it is considered a more efficient estimator, although it is less robust to the presence of a previous jump in its calculation. Additionally, the method proposed by Huang and Tauchen 2005 seeks only to determine whether or not a jump was present in a day, and not the intraday quantity and time of jumps (Davis 2008). Due to its ability to determine intraday jump metrics, as well as its resistance to the existence previous jumps, the bipower variation method of Lee and Mykland 2007 was chosen for the classification of jumps.

The method proposed by Lee and Mykland 2007 involves the construction of a test statistic $\mathcal{L}(i)$, initially proposed by Barndorff-Nielsen and Shephard 2004, which is expressed as:

$$\mathcal{L}_i = \frac{\log S(t_i)/S(t_{i-1})}{\widehat{\sigma(t_i)}} \tag{2.10}$$

This tests for the existence of a jump in a stock mid price S between time t_i and t_{i-1} by dividing the logarithmic return by the square root of the realized bipower variation $\widehat{\sigma(t_i)}^2$. The realized bipower variation of $S(t_i)$ can be calculated via:

$$\widehat{\sigma(t_i)}^2 = \frac{1}{K-2} \sum_{j=i-K+2}^{i-1} \left| \log \frac{S(t_j)}{S(t_{j-1})} \right| \cdot \left| \log \frac{S(t_{j-1})}{S(t_{j-2})} \right| \tag{2.11}$$

where K is a predefined window (Figure 2.7) which is chosen within the bounds:

$$\sqrt{252 \cdot n_{obs}} \leq K \leq 252 \cdot n_{obs} \tag{2.12}$$

where K exists between the product of the average number of trading days in a year, and the number of intraday observations, and the square root of the aforementioned product. The number of intraday observations is determined through the length of time step chosen[4].

[4]For example: if a timestep length of 5 minutes was chosen, $n_{obs} = 78$, since 78 intervals of 5 minutes exist within a 6.5 hour (390 minute) trading day

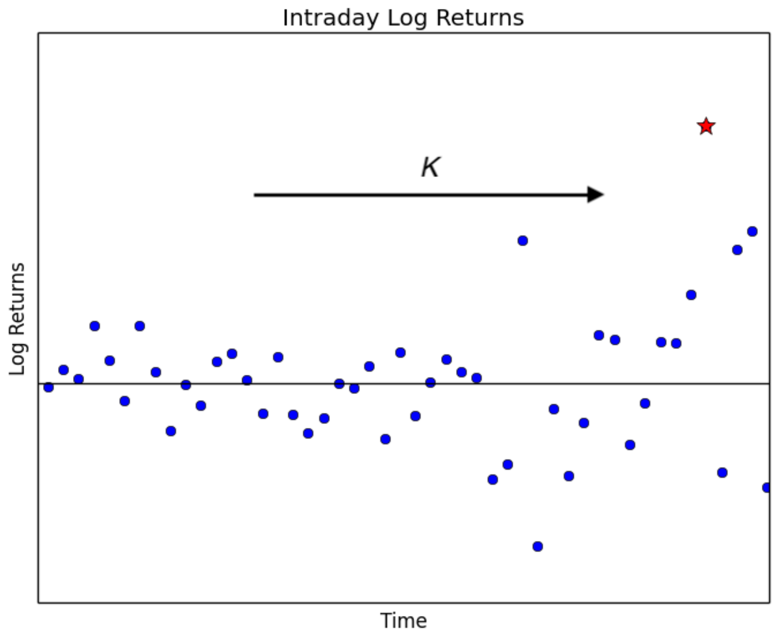


FIGURE 2.7: As the window K progresses through time, the bipower variation is consistently updated and the newest point is determined to exceed the jump statistic or not. The red star is a log return classified as a jump.

To begin classifying jumps with the above model, a threshold for the test statistic $\mathcal{L}$ must be made so that exceedance of this threshold results in the classification of a jump. The assumption made for the case where no jumps exist is that the test statistic follows an approximately normal distribution, where the magnitude of the limits of this distribution as $\Delta t \to 0$ are:

$$\frac{\max|\mathcal{L}(i)| - C_n}{S_n} \to \xi \tag{2.13}$$

where ξ has a cumulative distribution function $P(\xi \leq x) = \exp(-e^{-x})$, and C_n and S_n are defined by (Lee and Mykland 2007):

$$C_n = \frac{\pi(2\log n_{obs})^{1/2}}{\sqrt{2}} - \frac{\log\pi + \log(\log n_{obs})}{\frac{2^{3/2}}{\pi}(2\log n_{obs})^{1/2}} \tag{2.14}$$

$$S_n = \frac{1}{\frac{\sqrt{2}}{\pi}(2\log n_{obs})^{1/2}} \tag{2.15}$$

Equation (2.13) can now be used by testing the likelihood of an observed $\mathcal{L}(i)$ existing within the usual region of maximums. The testing setup is complete when a significance level α is chosen to create a test threshold β^* such that:

$$P(\xi \leq \beta^*) = \exp(-e^{-\beta^*}) = 1 - \alpha \tag{2.16}$$

Therefore, if a test statistic $\mathcal{L}(i)$ is presented such that:

$$\frac{|\mathcal{L}(i)| - C_n}{S_n} > \beta^*$$

then $\mathcal{L}(i)$ is assumed to not be from the continuous part of the jump diffusion model, and is classified as a jump.

The choice of α will determine the range of classification certainty, and can be suitably adjusted depending on the precision requirements of the presented problem.

2.5 Sentiment Analysis

Sentiment analysis, like the limit order book, is a tool that has been used in the financial sphere for a long time to measure market sentiment, but has recently evolved new methods due to technological advances. While market sentiment was traditionally measured through surveys or technical analysis of financial market metrics, a near constant stream of market sentiment is now able to be extracted directly from online sources of text-based data. These data sources include social media platforms, online news articles, blogs, and any other sources with relevance to markets and market products.

When a source has been obtained, there are three standard methods used to extract sentiment: the lexicon based approach, the machine learning approach, and a combination of both (Vu and Le 2017). The lexicon-based approach involves referring to a pre-compiled lexicon with sentiment values for specific words. The sentiment values are then aggregated to produce an overall sentiment value. There are limits to this approach, since words in general possess a degree of contextual sentiment, which may or may not be reflected in the lexicon, and there is no attention to chronological order of the words. The machine learning approach utilizes learning algorithms to train a model which can determine sentiment. This approach can be more versatile with respect to context, and can incorporate a temporal dimension to better classify sentences (for instance, there is an important difference between the word order of "Apple surpassed Microsoft in sales" and "Microsoft surpassed Apple in sales" when one is attempting to predict market trends) (Pang, Lee, et al. 2008).

A plethora of research has been conducted regarding sentiment analysis, Tetlock et al. 2008 used a lexicon-based approach to classify *Wall Street Journal* and *Dow Jones News Service* articles, and found that negative words used in their articles tended to forecast low firm earnings. Sun et al. 2016 text-mined heavy volumes of finance-specific social media from the website *StockTwits* data to predict market movements with success.

Alongside sentiment analysis, work has been done to better utilize publicly available text data in different ways as well. Mamaysky, Glasserman, et al. 2016 compiled more than 360,000 articles over 18 years, referencing 50 large financial companies, and determined a correlation between market volatility and an article's use of negative words not commonly associated with a company (described as the "unusualness" metric). Garcia 2013 discovered that the correlation between news article sentiment and stock returns is strengthened during times of recession.

2.6 LOB Statistical Moments

The viability of statistical quantities and methods for use in finance has been studied heavily, particularly the utilization of normal distributions in financial derivative pricing, and risk management through value-at-risk and expected shortfall models Hull 2003. In the realm of high frequency finance, there has been focus on the application of moving statistical moments when analyzing time series (Gençay et al. 2001). Moving k^{th} moments of the best 10 limit order book bid/ask prices and volumes will be used to reduce the stochastic dynamic nature of the limit order book information from timestep to timestep, attempting to reveal underlying price and volume trends.

For each limit order book state used, each k^{th} central moment m_k - namely the variance, skewness, and kurtosis - of all orders in the best 10 levels is calculated via:

$$m_k = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^k$$

where n is the total number of standing order at all 10 levels and $\bar{x}$ is the mean order price. Knowledge of these moments could help with model forecasting of jump probabilities, for instance, if the bid and ask volumes are found to both be skewed heavily toward the midprice (Figure 2.8), there would exist smaller limit order volumes closer to the midprice. Given this state, a single market order with a large enough volume could move the midprice far enough in a particular direction to be classified as a jump.

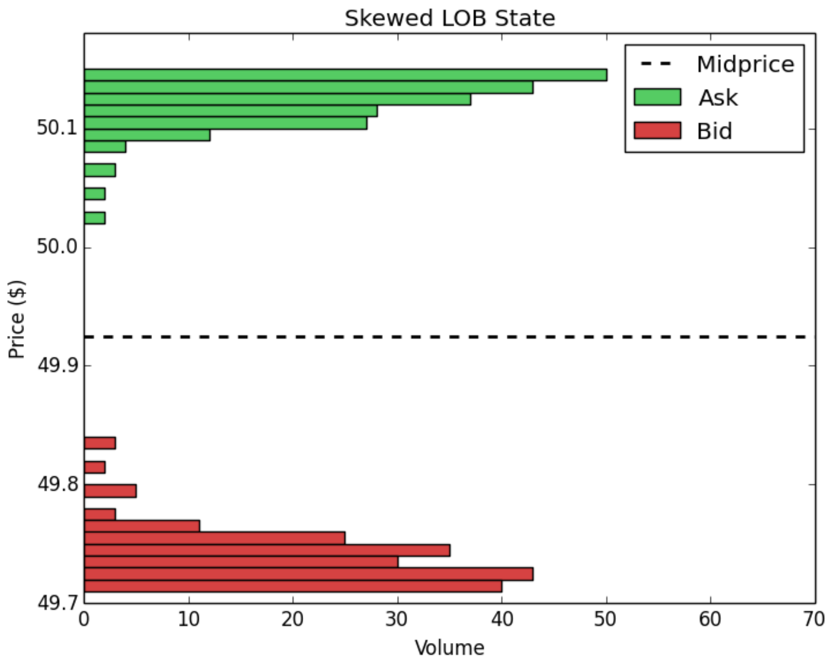


FIGURE 2.8: An example of an order book state where the bid and ask distributions are skewed towards the midprice.

Chapter 3

Data Acquisition and Methods Applied

3.1 Overview of Acquisition Process

The complete acquisition process involved obtaining raw data directly from NASDAQ, and undergoing several steps of processing and classification in order to produce a data set refined enough for a machine learning model (Figure 3.1). The raw data in question was obtained directly from NASDAQ, and was converted to human readable flow of messages for the entire 2013 trading year (1)[1]. Using this message data, the limit order book prices and volumes were reconstructed and the net order imbalance indicator event times were obtained (2). The data was then truncated to only include events taking place during standard trading hours (9:30AM-4:00PM) using NOII event times (3). The data was effectively converted from sporadic, nanosecond-scale event data to consistent 1 second intervals (4).

[1]All bracketed numbers in this section are referencing the steps of Figure 3.1

With the second intervals for each trading day calculated, any desired interval can be chosen for feature vector generation (5). In this case 30 second and 300 second intervals (0.5 and 5 minutes, respectively) were chosen.

With desired intervals chosen, the feature vectors inputs for the neural networks can begin to be constructed by extracting limit order book, message, and sentiment data specific to each time interval (6,7). Now that the specific intervals are chosen, stock midprice data can be extracted from the limit order book in these intervals (8), and the classification of midprice jumps can take place within their respective time intervals (9). With the feature sets extracted and their respective jumps classified, the final step is to convert the data into z-scores using moving windows, where the window sizes are dependent on the features in question (10).

3.2 NASDAQ ITCH Message Processing

The data used in its entirely unprocessed form was ITCH message data obtained directly from NASDAQ through academic licensing. ITCH is a direct data-feed protocol used to track book orders from the moment they are submitted to when they are either executed or canceled, while keeping track of any changes to the order in the process (Nasdaq 2014). This is NASDAQ's framework to collect and disseminate high-frequency market microstructure data. Due to the extremely high volume of orders in a given day, along with the need for information to be chronologically ordered, ITCH keeps track of the time that orders are placed with a nanosecond timescale precision.

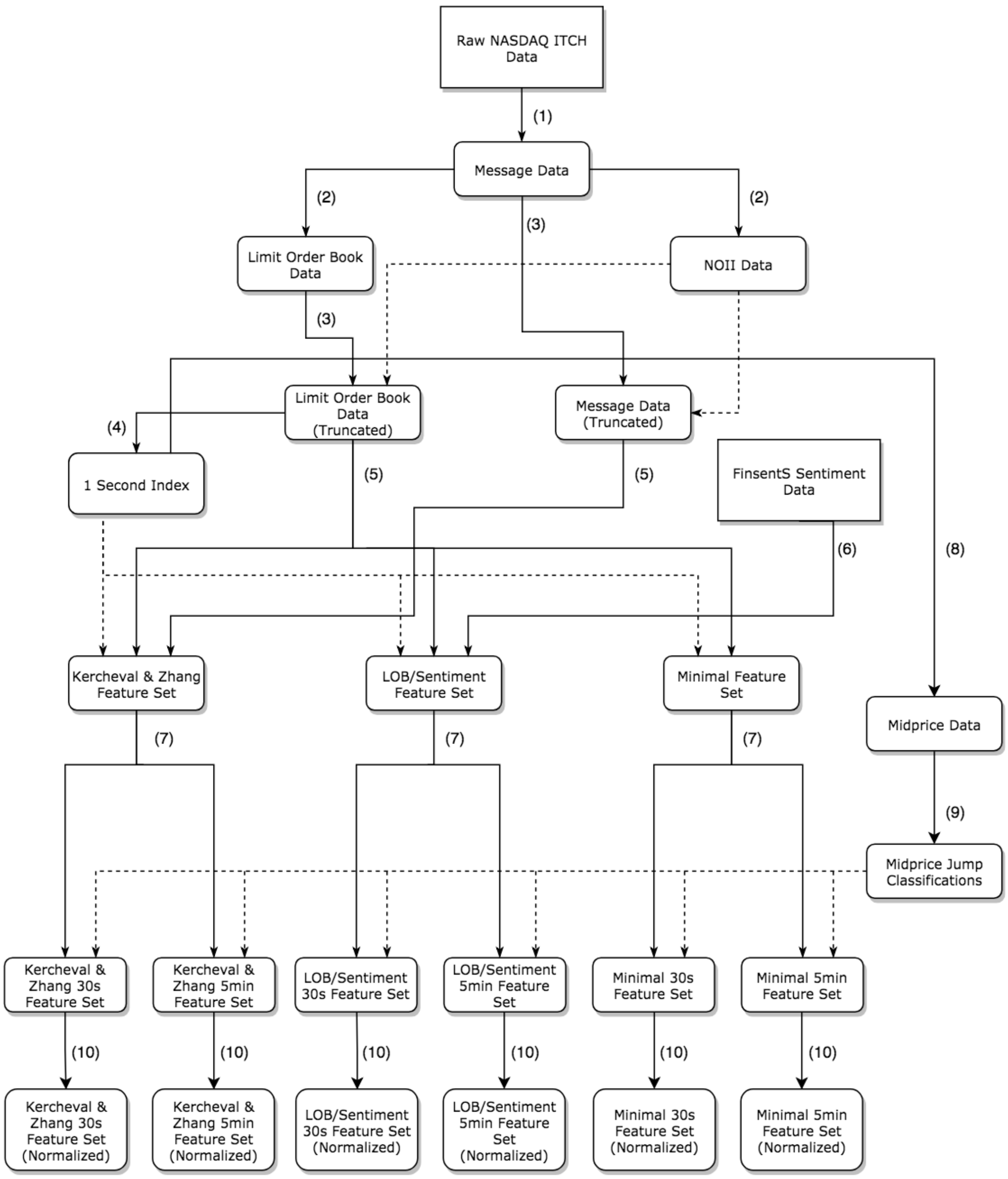
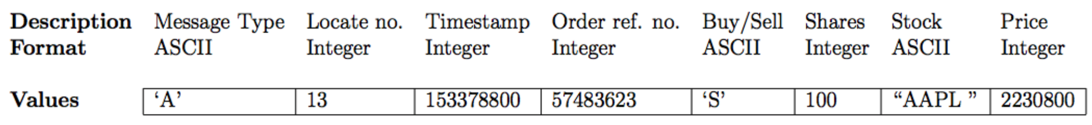

Raw NASDAQ ITCH Data
(1)
Message Data
(2)
(3)
Limit Order Book Data
NOII Data
Limit Order Book Data (Truncated)
Message Data (Truncated)
(4)
1 Second Index
(5)
FinsentS Sentiment Data
(6)
(8)
Kercheval & Zhang Feature Set
LOB/Sentiment Feature Set
Minimal Feature Set
Midprice Data
(7)
(9)
Midprice Jump Classifications
Kercheval & Zhang 30s Feature Set
Kercheval & Zhang 5min Feature Set
LOB/Sentiment 30s Feature Set
LOB/Sentiment 5min Feature Set
Minimal 30s Feature Set
Minimal 5min Feature Set
(10)
Kercheval & Zhang 30s Feature Set (Normalized)
Kercheval & Zhang 5min Feature Set (Normalized)
LOB/Sentiment 30s Feature Set (Normalized)
LOB/Sentiment 5min Feature Set (Normalized)
Minimal 30s Feature Set (Normalized)
Minimal 5min Feature Set (Normalized)

FIGURE 3.1: Flowchart for complete feature vector acquisition process for each stock. Solid lines represent the processing of the parent node to a more refined form, and dotted lines represent the use of information from the parent node to produce the child node.

Due to the sheer volume of orders in a day, storing this information requires ample memory space. NASDAQ mitigates storage concerns by storing limit order book data organized by day in a memory efficient binary format. Each message entry is stored and read as follows:

- A 2-byte integer stating how many bytes the message is comprised of
- A single byte character indicating what kind of message it is (system event message, limit order, timestamp, etc.)
- The message itself, which can now be unpacked into a human-readable format given the previous two pieces of information



Description	Message Type	Locate no.	Timestamp	Order ref. no.	Buy/Sell	Shares	Stock	Price
Format	ASCII	Integer	Integer	Integer	ASCII	Integer	ASCII	Integer
Values	'A'	13	153378800	57483623	'S'	100	"AAPL "	2230800

Figure 3.2: NASDAQ Message Example

Taking Figure 3.2 for example, the data fields, values, and types are specific only to the "add limit order" message. For comparison, a "timestamp" system event message possesses only two data fields (message type in ASCII format, and number of seconds since midnight in integer format). The specific order and format for each message is important as it ensures correct binary unpacking of the data for proper readability and analysis. The python toolkit "Prickle" (Swaney 2018) was utilized to streamline the processing, allowing mass conversion of raw binary data to CSV format for feature vector construction.

3.3 Truncation of Opening/Closing Cross Data

At the beginning and end of a standard NASDAQ trading day, there exists a process known as an opening and closing cross (Nasdaq 2018). The opening cross process is a virtually instantaneous process where orders placed before standard market hours are rapidly executed before normal trading for the day begins (9:30 AM). This can lead to rapid fluctuations in security prices at the beginning of standard trading hours, with a similar process occurring at close of standard trading hours (4 PM). Since this is a unique process that does not follow regular intraday market dynamics, the data for this process is not believed to be useful for machine learning purposes at this time. Because of this, the first and last seconds are truncated from each day of standard trading hours (data used exists from 9:00:01 AM - 3:59:59 PM) to isolate the data following the dynamics of interest[2].

3.4 Order Book Reconstruction

The LOB data used consists of the historical message and order book data 5 of the highest market cap stocks available for public trade on the NASDAQ electronic exchange in 2013 (Table 3.1). The raw data exists on a nanosecond timescale, and spans one full trading year ($\sim$ 252 trading days). The LOB is updated whenever a new message is received by the system, where the possible messages received include:

[2]During holiday hours the opening and closing cross times may be later/earlier than usual, but the large majority intraday trading hours exist with this range

- **Limit order submission**: The standard message where a bid or ask price along with a volume is submitted. The bulk of total messages received in a day fall into this category.
- **Cancellation**: A message sent to partially delete a standing limit order (i.e. reduce volume).
- **Deletion**: A message sent to completely delete a standing limit order.
- **Visible/hidden order execution**: A message stating that a standing visible or hidden limit order has been filled, the standing order is then removed from the book.

It is important to note that a large portion of limit orders do not actually get filled. Looking at Table 3.2 giving sample quantities of of LOB messages received in an average day, it can be seen that for every 100 limit order submissions[3] approximately 9 of those are actually executed. This is important because executions of orders are what will actually change the current price of the asset.

TABLE 3.1: Overview of Selected NASDAQ Stocks (Dec. 2013)

Company	Ticker symbol	Global market value ($m)
Apple	AAPL	504,770.8
Microsoft	MSFT	312,297.3
Google	GOOG	310,079.1
Intel Corp.	INTC	129,022.2
Facebook	FB	105,889.3

[3]Based on the data acquisition process, this is order submissions that affect the 10 best bid/best ask levels, the actual submission:execution ratio is likely much higher.

TABLE 3.2: Summary Statistics for AMZN NAZDAQ LOB 21/06/2013 (10 Best Bid/Ask Levels)

Total Number of Limit Orders	131954
Total Number of Visible Executions	8974
Total Number of Hidden Executions	2445
Total Number of Partial Order Deletions	2917
Total Number of Full Order Deletions	123458
Average Volume of All Limit Orders	98.72
Average Volume of All Buy Orders	105.64
Average Volume of All Sell Orders	92.33
Average Volume of Partial Deletions	118.45
Average Volume of Full Deletions	96.77

3.5 Event Data Conversion

As it stands, the data at the current state of processing exists in an event-based format. This means there is no strict universal time step between each data event - whenever a market participant decides to place an order, their time of placement is recorded to a nanosecond precision, maintaining chronological message order as best as possible. While the electronic order book benefits greatly from an event based system, it poses additional challenges when extracting data for use in a model that relies on uniform time steps.

The entire dataset can be represented by chronologically ordered data vectors $d_1, d_2, \ldots, d_i, \ldots, d_n$, where each vector d_i corresponds to the state of the order book at a unique time. Every time an order is placed in the market by a participant, the order book state is changed and a new d_i is produced.

Ideally, to ensure a smoother feature vector extraction for a time dependent

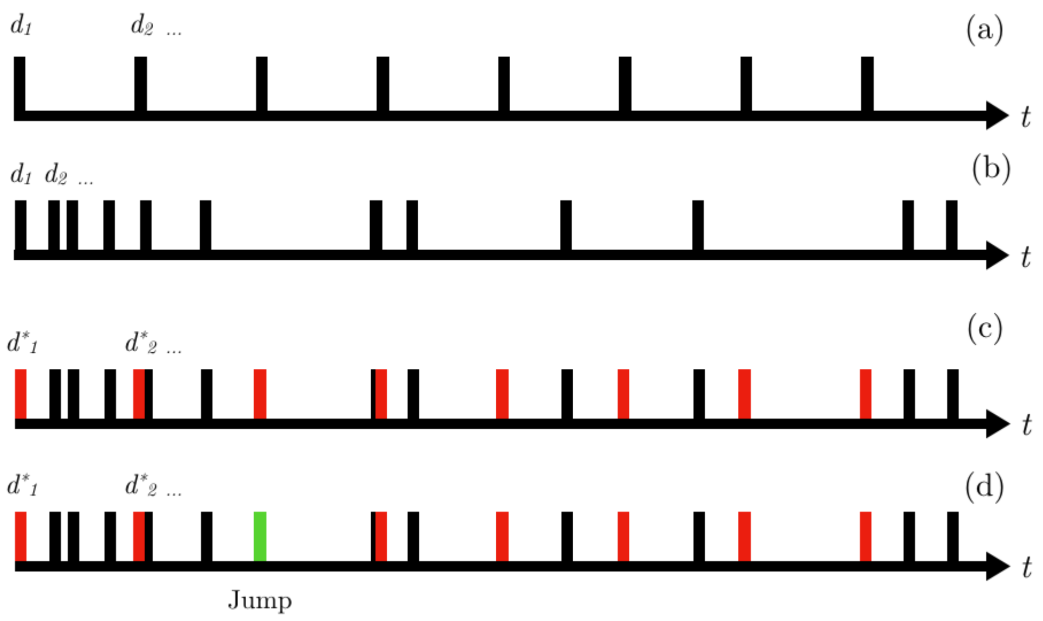


FIGURE 3.3: The different stages of converting event data $(d_1, d_2, \ldots)$ to a uniform time series. For (c) & (d), the red and green ticks represent uniform artificial time steps $(d_1^*, d_2^*, \ldots)$ and jump classified artificial time steps, respectively.

machine learning model, these d_i would exist equally separated by a predefined time step (Figure 3.3(a)). However these d_i actually exist with no predefined time step between each vector (Figure 3.3(b)), and generally follow a denser distribution towards the start and end of a trading day (when order volumes are higher), with potentially large spans of no additional d_i's created between high volume periods.

To counteract this situation, a conversion to uniform discrete time steps was made. For every second in the 23400 seconds of a standard trading day, an artificial set of time steps $d_1^*, d_2^*, \ldots, d_i^*, \ldots, d_{23400}^*$ was created (Figure 3.3(c)). For each d_i^*, the data vector from the most chronologically immediate prior time step was used. This ensures no future data is unfairly incorporated when training the models, and since the system had not been updated by a order between the artificial time step and the prior order, the information used is actually the exact information of the state of the order book at that particular second.

With these artificial time steps generated, larger time intervals can now be chosen if desired, and midprice information contained within these data vectors can be used to classify midprice jumps (Figure 3.3(d)). The time intervals selected for use in this project were of 30 seconds and 300 seconds (5 minutes). Additionally, if oversampling is needed to reduce the classification imbalance of jump/no-jump samples, the aforementioned method can be used to acquire additional, unique, and unused d_i's which are also chronologically closest to the d_i^* in question.

3.6 Stock Midprice Jump Detection

The quantities we are interested in predicting are constructed and classified through a process consisting of a moving window of size K, where K is chosen to be $K_{30s} = 1560$ intervals of 30 seconds for the 30 second data, and $K_{5min} = 156$ intervals of 5 minutes for the 5 minute data (both equivalent to two full 6.5 hour trading days). This window is chosen in accordance with Barndorff-Nielsen and Shephard 2004 where the window K should exist as:

$$\sqrt{d_t \cdot n_{obs}} < K < d_t \cdot n_{obs}$$

where d_t is the standard number of trading days (252) in a given year and n_{obs} is the number of observations in a day (780 for 30 second data, 78 for 5 minute data). These boundaries show K_{30s} and K_{5min} to exist closer to their respective lower bounds, where it has been shown in Lee and Mykland 2007 that it is better to chose a value for K closer to the lower bound since higher choices have diminishing returns while increasing the computational cost unnecessarily.

The significance level α was initially set at 0.05 for both time intervals, but led to too few jumps being classified, which would exacerbate issues regarding the large imbalance of positive/negative time step classifications for the predictive models during training. α was then set to 0.1 which was still within appropriate significance levels for jump classification, and produced more suitable classification results for the subsequent model training. The results of the jump detection algorithm are shown in Chapter 5.

3.7 Sentiment Data Acquisition

In addition to the LOB data obtained from NASDAQ, historical stock-specific sentiment data sourced from the financial engineering/news analytics company *InfoTrie* for use in the proposed new feature vector (InfoTrie 2016). The data is an amalgamation of market sentiment derived from text based data from 6000 business news publications, websites, and blogs, scanned continuously in 5 minute intervals. A relevance score is assigned to each source regarding its relevance towards a particular stock. Sources which surpass the relevance threshold are then analyzed using natural language processing and machine learning to obtain a sentiment score, which is then weighted based on factors like the source's popularity, credentials, etc. All weighted sentiment sources are then grouped and processed at 11PM every day to produce five stock specific market sentiment values for that day (Figure 3.4).

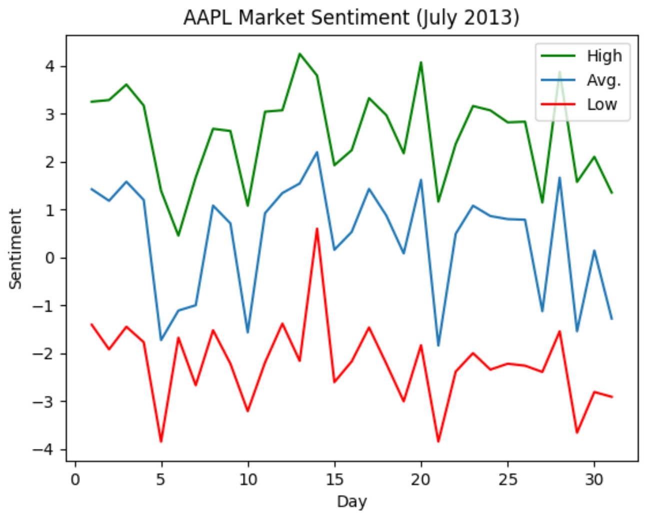


FIGURE 3.4: Calculated high, low, and average sentiment for Apple specific sources (sentiment score $S \in [-5, 5]$)

3.8 Construction of Feature Sets

The feature sets constructed for comparison consist of the feature vector proposed by Kercheval and Zhang 2015, along with two new proposed feature vectors which utilize different combinations of features used in their feature set, as well as some novel additions. The original Kercheval and Zhang 2015 feature set includes a combination of basic features, time-insensitive features, and time-sensitive features (Table 3.5). They experimented with a combination of these feature sets as well, and concluded that a feature set comprised of basic and time-insensitive features produced the most accurate results for the majority of stocks they researched.

Because of this, a proposed new feature vector was introduced that includes the basic and time insensitive features, along with an additional sentiment vector component, and a vector of LOB statistical moments (Table 3.6). The motivation behind this feature vector is to ascertain whether or not a public sentiment component can add predictive value to the feature set of Kercheval and Zhang 2015 when predicting market jumps.

The final feature vector proposed for model comparison explores the viability of a minimalist feature vector, and substitutes the basic feature set for the moment feature set, while keeping the time-insensitive features (Table 3.7). The motivation behind this feature vector was to determine if the sole use of the limit order book moments can provide enough predictive value to disregard the more chaotic raw LOB data utilized in the basic feature set.

All feature vectors include a time dimension, which is rounded to the nearest trading hour. This rounding attempts to curb any over fitting of the time dimension (jumps have shown to be very time-of-day dependent), while still providing a useful feature metric.

3.8.1 Basic features

Basic features are features that are extracted directly from the raw LOB data and require no operations or manipulation before use as model input. These include the best 10 bid/ask level prices for the stock in question f^{ask}/f^{bid}, along with their respective volumes V^{ask}/V^{bid} at a particular instant in time.

3.8.2 Time-insensitive features

Time-insensitive features are derived from the raw LOB data and require processing before feature vector utilization. These features are comprised of the spreads and mid-prices between contrasting levels of the LOB (best bid versus best ask, second best bid versus second best ask, etc.) along with the price and volume means of all levels, and the sum of the LOB spreads for all levels and respective volumes.

3.8.3 Time-sensitive features

Time-sensitive features take advantage of the raw message data alongside the LOB data. Time sensitive LOB features consist of the rate of change of price and volume over the most recent $\Delta t = 1s$. The remaining time sensitive features are derived from the message flow data, and pertain to 'intensity', which is the rate of flow of message data regarding a specific type of order (or cumulative of all orders) over $\Delta t = 1s$. These include limit bid/ask orders, market bid/ask orders, cancellation of limit bid/ask orders, and total flow of participant messages (i.e. orders coming from traders, not system messages such as time stamps and market halts). Next is a relative intensity comparison - this is a binary indicator stating whether or not intensity has increased compared to $\Delta T = 900s$ ago, which is again applied to all forms of order message flow. The final set is comprised of the derivatives of these order intensities over the last $\Delta t = 1s$.

FIGURE 3.5: Kercheval and Zhang Feature Set

Feature Set	Description	Details
Basic		
	$\nu_1 = \{f_i^{ask}, V_i^{ask}, f_i^{bid}, V_i^{bid}\}$	10-level Raw LOB Data
Time-Insensitive		
	$\nu_2 = \{(f_i^{ask} - f_i^{bid}), (f_i^{ask} + f_i^{bid})/2\}$	Spread & Midprice
	$\nu_3 = \{f_n^{ask} - f_1^{ask}, f_1^{bid} - f_n^{bid}, \lvert f_{i+1}^{ask} - f_i^{ask}\rvert, \lvert f_{i+1}^{bid} - f_i^{bid}\rvert\}_{i=1}^n$	Price Differences
	$\nu_4 = \{\frac{1}{n}\sum_{i=1}^n f_i^{ask}, \frac{1}{n}\sum_{i=1}^n f_i^{bid}, \frac{1}{n}\sum_{i=1}^n V_i^{ask}, \frac{1}{n}\sum_{i=1}^n V_i^{bid}\}$	Price & Volume Means
	$\nu_5 = \{\sum_{i=1}^n (f_i^{ask} - f_i^{bid}), \sum_{i=1}^n (V_i^{ask} - V_i^{bid})\}$	Accumulated Differences
Time-Sensitive		
	$\nu_6 = \{df_i^{ask}/dt, df_i^{bid}/dt, dV_i^{ask}/dt, dV_i^{bid}\}_{i=1}^n$	Price & Volume Derivation
	$\nu_7 = \{\lambda_{\Delta t}^{la}, \lambda_{\Delta t}^{lb}, \lambda_{\Delta t}^{ma}, \lambda_{\Delta t}^{mb}, \lambda_{\Delta t}^{ca}, \lambda_{\Delta t}^{cb}, \lambda_{\Delta t}^{tot}\}$	Average Intensity per Type
	$\nu_8 = \{\mathbf{1}_{\lambda_{\Delta t}^{la} > \lambda_{\Delta T}^{la}}, \mathbf{1}_{\lambda_{\Delta t}^{lb} > \lambda_{\Delta T}^{lb}}, \mathbf{1}_{\lambda_{\Delta t}^{ma} > \lambda_{\Delta T}^{ma}}, \mathbf{1}_{\lambda_{\Delta t}^{mb} > \lambda_{\Delta T}^{mb}}, \mathbf{1}_{\lambda_{\Delta t}^{ca} > \lambda_{\Delta T}^{ca}}, \mathbf{1}_{\lambda_{\Delta t}^{cb} > \lambda_{\Delta T}^{cb}} \mathbf{1}_{\lambda_{\Delta t}^{tot} > \lambda_{\Delta T}^{tot}}\}$	Relative Intensity Comparison
	$\nu_9 = \{d\lambda^{la}/dt, d\lambda^{lb}/dt, d\lambda^{ma}/dt, d\lambda^{mb}/dt, d\lambda^{ca}/dt, d\lambda^{cb}/dt, d\lambda^{tot}/dt\}$	LOB Activity Acceleration
Time		
	$\nu_{10} = \{\frac{t}{3600}\}$	Hour of Day

3.8.4 Statistical moment features

The statistical moment feature vector is comprised of the volume weighted mean, variance, skewness, and kurtosis of all 10 bid/ask levels, reducing 40 independent features to 8 independent features.

3.8.5 Sentiment features

The sentiment features are derived separately from the LOB and message data features, they are sourced via the `Quandl` API and contain five stock specific sentiment metrics updated on a daily basis:

- **Sentiment High (S_{high}):** The highest recorded sentiment value for the day (unweighted).
- **Sentiment Low (S_{low}):** The lowest recorded sentiment value for the day (unweighted).

- **Sentiment Average (S_{avg}):** The weighted average of sentiment from all recorded sources for the day.
- **Sentiment Volume (S_{vol}):** The total volume of articles sourced in the given day.
- **Sentiment Buzz (S_{buzz}):** The change in standard deviations of periodic news volume. Defined on a scale from 1-10, a high buzz score indicates a fast change in news volume and suggests higher volatility.

Since this data is strictly a daily metric, care was taken to avoid future information being included in the training vectors. For each collective day, metrics collected and amalgamated the previous day [4] were incorporated into the feature vectors.

FIGURE 3.6: Proposed Sentiment/Moment Feature Set

Feature Set	Description	Details
Basic		
	$\nu_1 = \{f_i^{ask}, V_i^{ask}, f_i^{bid}, V_i^{bid}\}$	10-level Raw LOB Data
Time-Insensitive		
	$\nu_2 = \{(f_i^{ask} - f_i^{bid}), (f_i^{ask} + f_i^{bid})/2\}$	Spread & Midprice
	$\nu_3 = \{f_n^{ask} - f_1^{ask}, f_1^{bid} - f_n^{bid}, \lvert f_{i+1}^{ask} - f_i^{ask}\rvert, \lvert f_{i+1}^{bid} - f_i^{bid}\rvert\}_{i=1}^{n}$	Price Differences
	$\nu_4 = \{\frac{1}{n}\sum_{i=1}^{n} f_i^{ask}, \frac{1}{n}\sum_{i=1}^{n} f_i^{bid}, \frac{1}{n}\sum_{i=1}^{n} V_i^{ask}, \frac{1}{n}\sum_{i=1}^{n} V_i^{bid}\}$	Price & Volume Means
	$\nu_5 = \{\sum_{i=1}^{n}(f_i^{ask} - f_i^{bid}), \sum_{i=1}^{n}(V_i^{ask} - V_i^{bid})\}$	Accumulated Differences
Moments		
	$\nu_6 = \{\mu_{ask}, \sigma_{ask}, \gamma_{ask}, \kappa_{ask}, \mu_{bid}, \sigma_{bid}, \gamma_{bid}, \kappa_{bid}\}$	Volume Weighted Moments
Time		
	$\nu_7 = \{\frac{t}{3600}\}$	Hour of Day
Sentiment		
	$\nu_8 = \{S_{avg}, S_{high}, S_{low}, S_{vol}, S_{buzz}\}$	Daily Sentiment Metrics

[4]The cutoff time for data accrued was 23:00 (11:00 PM), which was as close a time as possible available without introducing future data.

FIGURE 3.7: Proposed Minimalist Feature Set

Feature Set	Description	Details
Time-Insensitive		
	$\nu_2 = \{(f_i^{ask} - f_i^{bid}), (f_i^{ask} + f_i^{bid})/2\}$	Spread & Midprice
	$\nu_3 = \{f_n^{ask} - f_1^{ask}, f_1^{bid} - f_n^{bid}, \lvert f_{i+1}^{ask} - f_i^{ask}\rvert, \lvert f_{i+1}^{bid} - f_i^{bid}\rvert\}_{i=1}^{n}$	Price Differences
	$\nu_4 = \{\frac{1}{n}\sum_{i=1}^{n} f_i^{ask}, \frac{1}{n}\sum_{i=1}^{n} f_i^{bid}, \frac{1}{n}\sum_{i=1}^{n} V_i^{ask}, \frac{1}{n}\sum_{i=1}^{n} V_i^{bid}\}$	Price & Volume Means
	$\nu_5 = \{\sum_{i=1}^{n}(f_i^{ask} - f_i^{bid}), \sum_{i=1}^{n}(V_i^{ask} - V_i^{bid})\}$	Accumulated Differences
Moments		
	$\nu_6 = \{\mu_{ask}, \sigma_{ask}, \gamma_{ask}, \kappa_{ask}, \mu_{bid}, \sigma_{bid}, \gamma_{bid}, \kappa_{bid}\}$	Volume Weighted Moments
Time		
	$\nu_7 = \{\frac{t}{3600}\}$	Hour of Day

3.9 Oversampling of Jump Classified Data

Due to the large classification imbalance between jump/no-jump for all time steps, oversampling and/or undersampling of the positive/negative data are considered useful applications to reduce this imbalance (Yap et al. 2014). In this case, the scenario naturally caters to choice of oversampling, since the feature set is extracted from a larger set of data, so there already exists unused samples that belong to the positive jump classification. When a positive jump was encountered during feature extraction, the extraction algorithm found additional data events (if they existed) which also fell under the categorization of "a jump existing within the next 30 seconds/5 minutes". To avoid the model training to recognize a cluster of similar time steps in succession, randomization was applied in both positive and negative sampling, while still maintaining chronological order.

3.10 Normalization of Feature Vectors

With the feature vectors compiled for both time spans, normalization of the data is necessary due to orders of magnitude differences between the features (for instance, a stock price level at time t being \$100 while a mean price difference between levels being \$0.03 will reduce the influence of the latter if kept at their respective scales). Due to the lack of stationarity in the data, extra normalization steps need to take place (Längkvist et al. 2014). If the naïve approach of normalizing the entire time series dataset is taken, the intraday dynamics on which the model is reliant will be heavily outweighed by long term drift components. Instead, a moving z-score normalization window is applied to the majority of features to obtain a normalized feature set $\mathbf{z}$, where:

$$z_i = \frac{x_i - \bar{\mathbf{x}}}{\sigma_{\mathbf{x}}}$$

where $\mathbf{x}$ is a moving window in the time dimension (2 hours long for the 30 second feature set, and 20 hours long for the 5 minute feature set) of a single feature, and $\bar{\mathbf{x}}$ and $\sigma_{\mathbf{x}}$ are mean and standard deviation of the moving window. Each feature in the feature vector is normalized separately, again due to substantial differences in magnitudes between features. The remaining features, namely "hour of day" and sentiment metrics, can be normalized throughout the entire time series due to their stationarity.

Chapter 4

Neural Network Setup

4.1 Model Construction

The neural network was built in a Python framework using Keras, a high level neural network API (Chollet et al. 2015). The backend to Keras' API utilizes Tensorflow, an open source machine learning framework capable of high performance numerical computation (Martin Abadi et al. 2015). The "sequential" construction model was chosen, as it grants the ability to construct, edit, and test multi-layered networks in a streamlined manner. The model constructed follows guidelines initially proposed by Tsantekidis et al. 2017 for time series prediction in a financial context, and utilized by Mäkinen 2017 in the specific context of jump detection when comparing different neural network model setups.

The model (Figure 4.1) starts with the jump/no jump classified, normalized, time series input. The input is fed directly into an LSTM layer of 40 nodes (Tsantekidis et al. 2017 recommends between 32 and 64 hidden neurons for the

LSTM layer). This LSTM layer utilizes a hyperbolic tangent activation function, and returns a sequence output. The output from the LSTM layer is fed into a dropout layer to prevent over-fitting of the model, where the dropout value for testing ranged between 0.3 and 0.5 depending on the stock and model in question. The output of the dropout layer was then fed into a regular dense layer of 40 neurons, which was then fed into a single dense neuron layer with a sigmoidal activation function to produce a binary classification output.

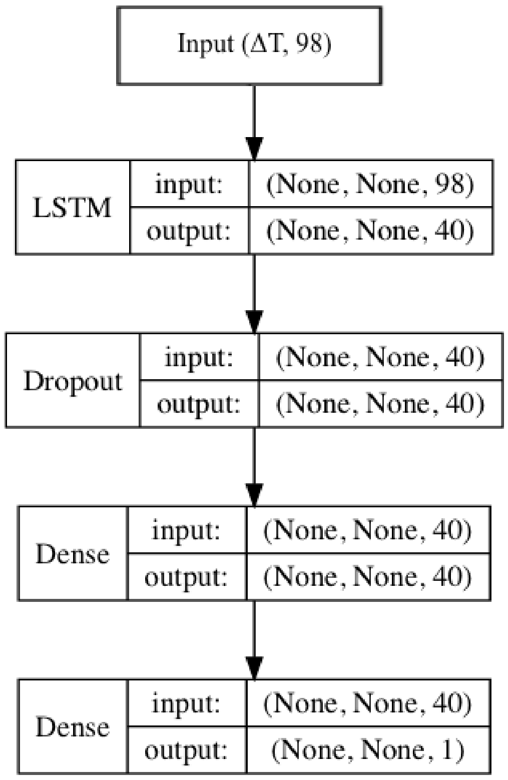


FIGURE 4.1: Schematic of the neural network model's layers and their input/output dimensions, a shape of 'None' specifies to the network to observe the entire time series data sequence while learning.

A batch size of 1000 was chosen for the 5 minute time series data, while a batch size of 30000 was chosen for the 30 second time series data (equating to the same total time spans). Due to the size of the data time spans and the model possessing an LSTM component, a large number of epochs was generally needed for effective weight tuning within the model, ranging between 200/400 epochs for 5 minute data, and 300/1000 epochs for 30 second data. The model was trained using binary cross-entropy as a loss minimizing function, and the "Adam" algorithm was used as the optimizer while training (Kingma and Ba 2014).

Due to the temporal dependencies of the time series data, a standard k-fold cross validation method not feasible due to the inevitability of testing the model on data that exists chronologically before training data. Instead a nested cross validation method was used (Figure 4.2). This method consisted of defining a percentage of the data at the end of the time series devoted to testing beforehand, and choosing the number of trials for cross validation. Once the number of trials k is chosen, the data used for testing is divided into k equal portions. For the first trial, the chronologically first portion is used for training, and is tested on a percentage of data immediately after it, where the percentage of testing data with respect to the entire data set is equal to that of the reserved testing data. For the next consecutive trials, the next $k-1$ equal portions of training data are included in sequence, while maintaining a consistent percentage of testing data appended temporally after. For each trial's training set, a percentage of the data is reserved as validation data. This validation percentage does not remain consistent with respect to the entire data set between trials, and will scale up in size proportional to the testing set as trials progress.

For this experiment, 15% of the total data was chosen for testing, while 20% of each trial's testing set was used for model validation while training. The number

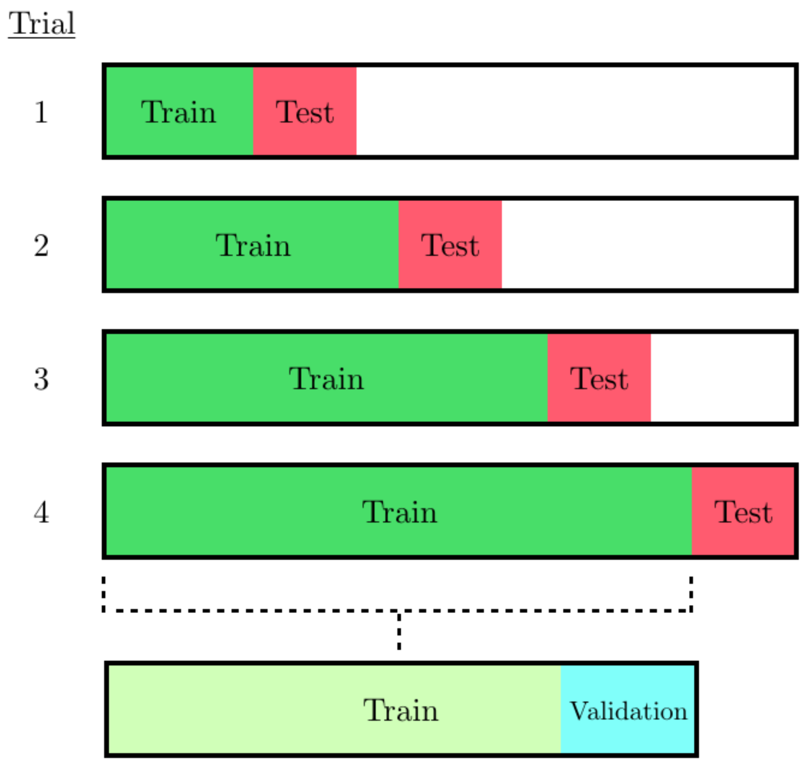


FIGURE 4.2: Nested cross validation method used (portions of training/validation/testing split percentages represented accurately).

of trials set for nested cross-validation was 4 - this trial amount was chosen to be relatively small due to the large data imbalance granting the possibility of a trial portion possessing very little positively classified data if the number of trials is too large. Leaving the trials portions proportionally larger gives less chance of this occurring and better chance of proper model fitting.

4.2 Measurement Metrics and Training

Measuring the predictive performance of a jump detection model can be difficult using standard methods due to the substantial inequality in the data between samples classified as jumps and samples classified as normal. This poses the problems such as a model using simple accuracy methods can achieve a near perfect predictive accuracy where:

$$\frac{\sum t_p + \sum t_n}{N}, \quad \sum t_p \ll \sum t_n$$

by simply being trained to classify every event as not possessing a jump - this is what is known as the "accuracy paradox" (Abma 2009). To counter the classification imbalance in the problem at hand, the F_1 score is used as a performance metric (Lipton et al. 2014):

$$F_1 = \frac{2t_p}{2t_p + f_p + f_n}, \quad F_1 \in [0, 1]$$

which is otherwise known as the harmonic mean of precision and recall:

$$F_1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

where:

$$\text{Precision} = \frac{t_p}{t_p + f_p}, \qquad \text{Recall} = \frac{t_p}{t_p + f_n}$$

It is important to note that the numerator is dependent on the existence of true positives. Due to this, under-classification of jumps in training is inherently discouraged. For these reasons, the training of the models focused on minimizing loss, while maximizing F1, precision, and recall scores.

All processing of data, extraction of feature vectors, and training of predictive models in a timely manner was made possible through the utilization of the computational resources of SHARCNET's "Graham" supercomputer cluster (Sharcnet 2017), which allowed extensive parallelization and reduced computational execution times to manageable levels.

Chapter 5

Results and Comparisons

5.1 Jump Detection Results

The choice of $\alpha = 0.1$ discussed in Chapter 3 produced the jump statistic thresholds $|\mathcal{L}_{30s}| \sim 8.55$, and $|\mathcal{L}_{5min}| \sim 7.27$. The distribution of jump sizes (Figure 5.1), where a larger $|\mathcal{L}|$ indicates a larger jump size, remains consistent with the theory of stock prices following geometric Brownian motion - in that the larger a jump from the previous price point is, the rarer its occurrence.

By studying the proportionality of positive/negative jumps, along with differences between individual stocks, some interesting results can be seen. Looking first at the 30 second interval jumps (Figure 5.2/Table 5.1), there is a $\sim$ 47%/53% split between positive and negative jumps, leaning slightly towards negative. For each individual stock, the proportions between positive and negative remain generally consistent (stocks with fewer positive jumps tend to have fewer negative jumps as

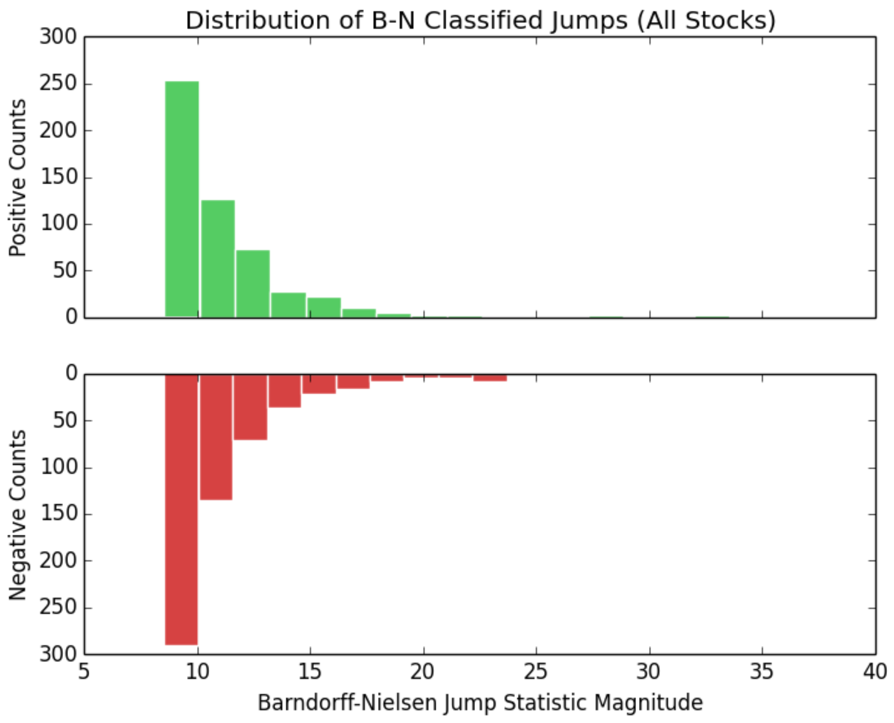


FIGURE 5.1: The absolute valued $\mathcal{L}$-statistic counts for all 30 second midprice movements classified as jumps

well). The highest market cap stocks in the group, Apple, Microsoft, and Google[1], all appear to have similar jump rates, while Intel Corp. possesses the highest rate of both positive and negative jumps.

For the 5 minute interval jumps (Figure 5.3/Table 5.2), there was actually a slight positive lean in the 55%/45% positive/negative split, attributed mostly to the imbalance between the positive/negative split of Intel Corp jump counts. Aside from Intel Corp, the proportionality between positive and negative jumps remains again consistent, however jump proportions for all stocks have seen a positive shift in proportionality.

[1]These are the three highest market cap stocks in the time period of 2013, not necessarily still the highest in the current time period

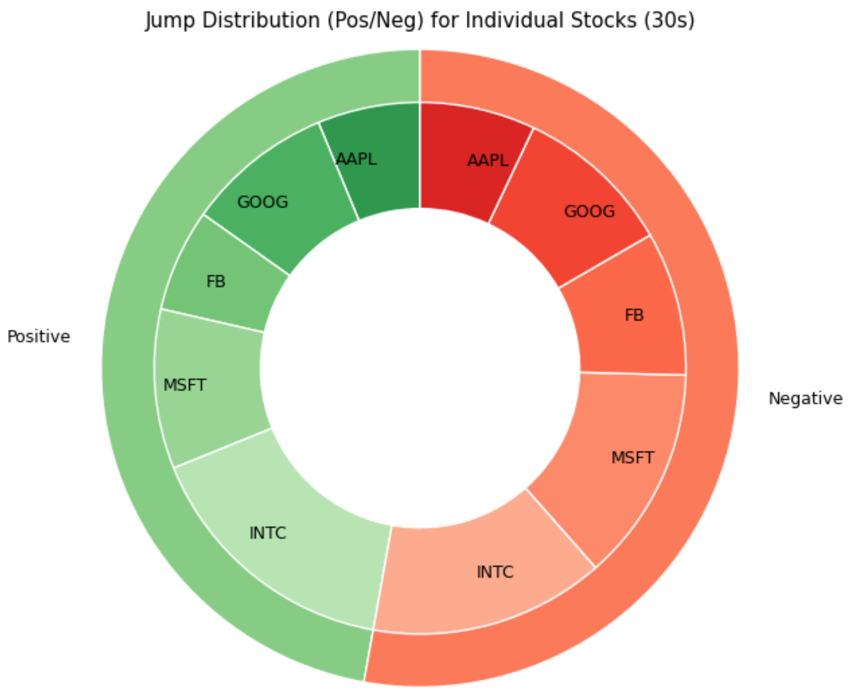


FIGURE 5.2: The $\mathcal{L}$-statistic positive/negative proportions for all 30 second jumps between all stocks.

TABLE 5.1: 30 Second Jump Statistics

	Positive	Negative
All Stocks	47.18%	52.82%
Apple	6.24%	7.02%
Google	8.94%	9.63%
Facebook	6.24%	8.76%
Microsoft	9.71%	13.10%
Intel	16.05%	14.31%

TABLE 5.2: 5 Minute Jump Statistics

	Positive	Negative
All Stocks	55.00%	45.00%
Apple	11.11%	8.33%
Google	9.44%	9.44%
Facebook	7.77%	7.22%
Microsoft	10.56%	10.00%
Intel	16.11%	10.00%

Regarding time of day, the likelihood of jumps is heavily skewed towards the first hour of the day, when trading volume is heaviest (Figure 5.4). There is also a slight increase in jumps at 14:00 (2:00PM), this is consistent with the findings of Mäkinen 2017.

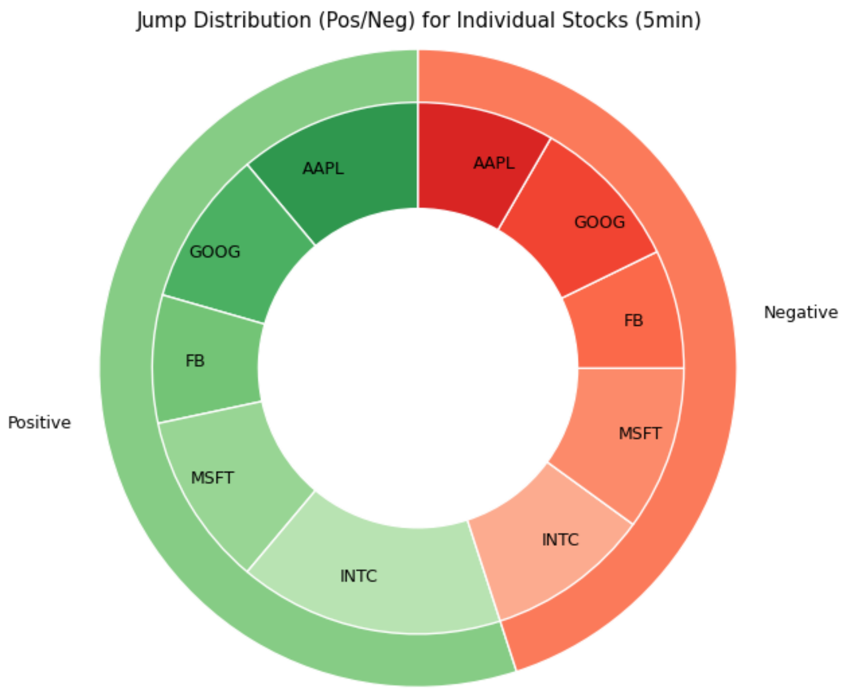


FIGURE 5.3: The $\mathcal{L}$-statistic positive/negative proportions for all 5 minute jumps between all stocks.

5.2 LSTM Model Results

The models were fit and trained via the methods and hyperparameters stated in Chapter 4. For each model, weights were optimized according to the best validation data F1 score. A Keras callback function "checkpoint" utility was used throughout training to save the model weights from the epoch possessing the best validation F1 score. The model was then tested utilizing the designated testing portions shown in Figure 4.2. Careful consideration was taken to ensure no training/testing/validation splits possessed unusually skewed classification imbalances which could affect training performance and test results. This was done by reviewing the individual jumps for each stock throughout the year (Figure 5.5)

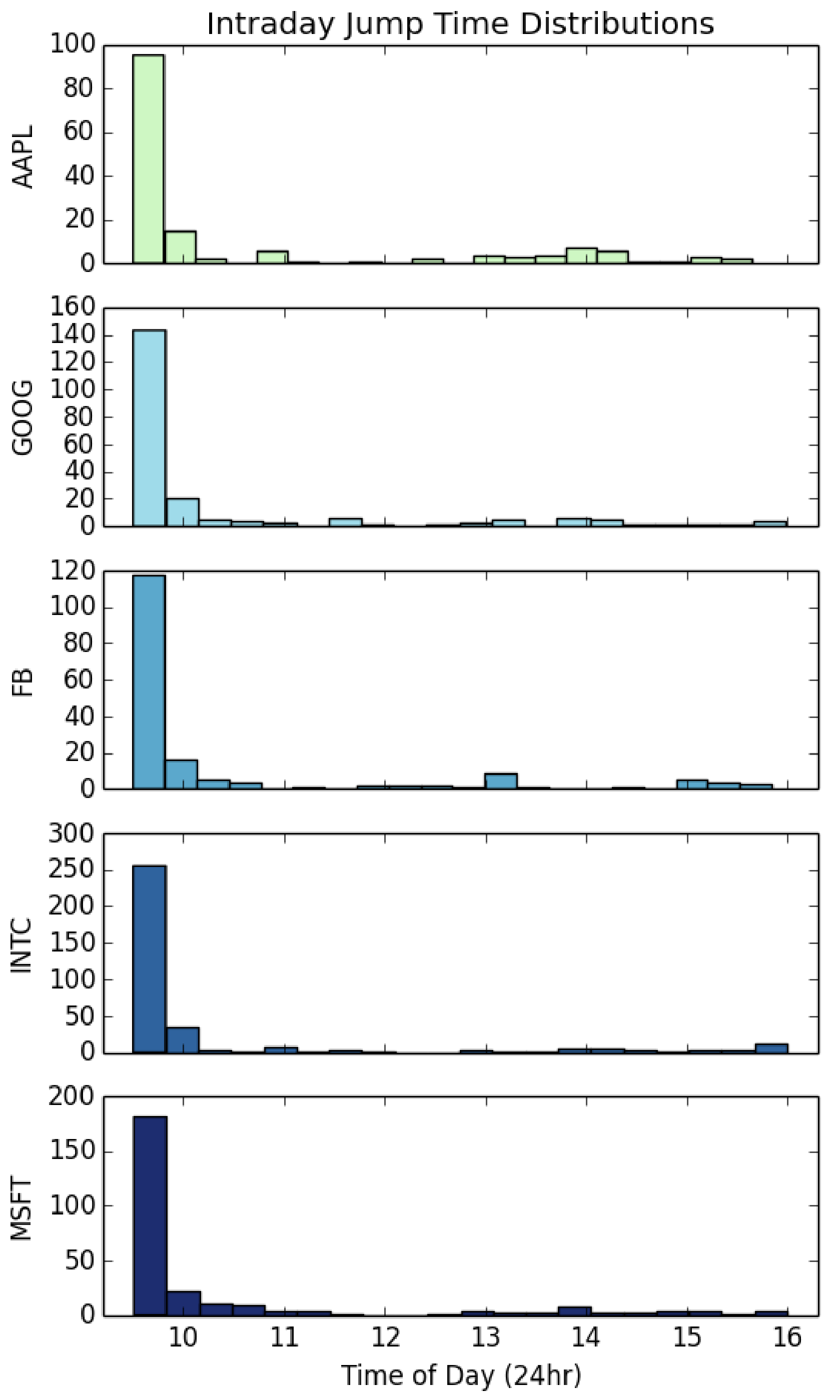


FIGURE 5.4: Intraday jump time distributions for all five stocks (from 9:30AM to 4:00PM).

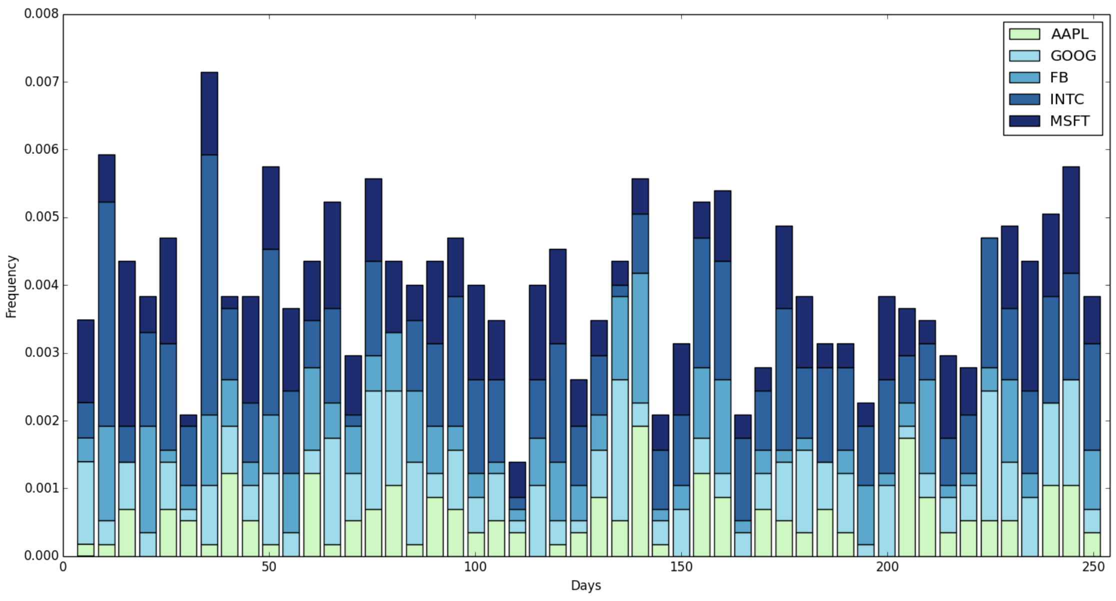


FIGURE 5.5: Histogram of jump proportions for each stock throughout the 2013 trading year.

For all models, the F1 score, precision, recall, and loss metrics were recorded throughout training and testing (Figure 5.6/5.7/5.8). While the main validation metric observed was the F1 score, precision and recall were also recorded throughout the training process in order to ensure a properly trained model. In Figure 5.6, the precision, recall, and F1 scores for each training epoch can be observed[2]. Initially, there is a noticeable spread between precision and recall scores, this indicates that while the model is initially predicting positive jump results with accuracy (low false positive rate), it is not predicting a large portion all jumps within the data set (high false negative rate). This spread converges to a mean test score of approximately 0.94 as the training epochs continue. Reviewing Figures 5.7 and 5.8, metric comparisons between the training and validation data can be seen. It is

[2]While this training plot in particular is for the GOOG ticker, 5 minute, sentiment/moment feature data, similar plots were produced for all models.

evident that throughout the training process the validation loss remains slightly lower than the training loss, and the validation F1 score remains higher than its training counterpart. This is consistent with neural network models that possess a dropout layer, due to a consistent percentage of feature weights being set to zero throughout the training process, whereas when testing all features are used, usually leading to better testing metrics.

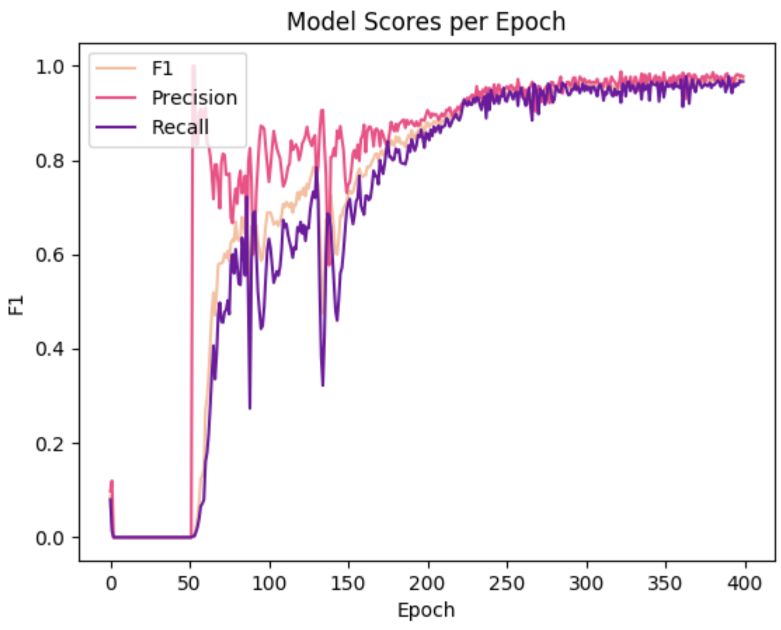


FIGURE 5.6: Training metric progression of GOOG Sentiment 5 minute data

5.3 Feature Set Results Comparison

Test scores for the majority of models throughout the nested cross validation process (Tables 5.3 and onward) were lower than those of the validation test scores. This is normal however when testing on data unseen to the model. For all models,

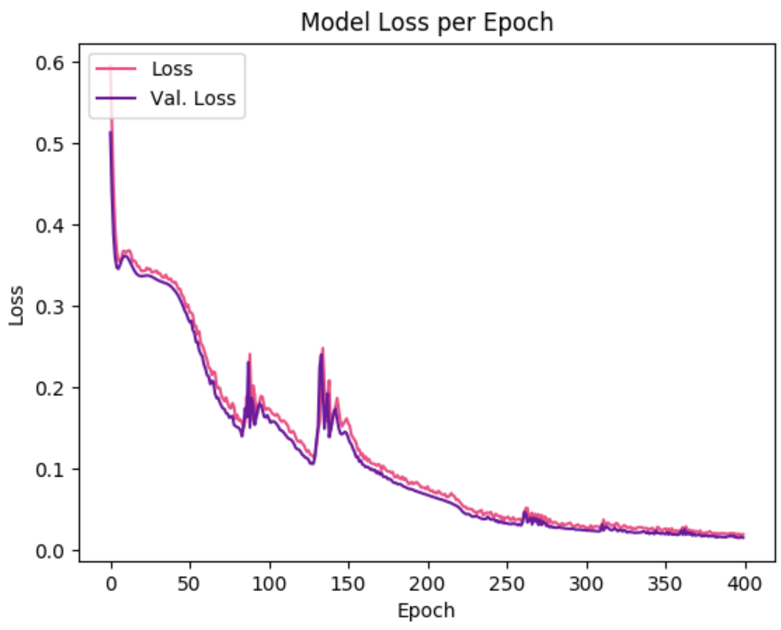


FIGURE 5.7: Training loss progression of GOOG Sentiment 5 minute data

the F1 and precision scores exceeded that of a random classifier, where the random classification precision and F1 scores are naturally quite low due to the large classification imbalance present in the data sets.

The feature set proposed by Kercheval and Zhang 2015, which took the longest to process and train due to its feature set size and reliance on message data as well as limit order book data, distinctly outperformed the sentiment and minimalist feature vectors for both time spans of the FB model, which possessed the largest classification imbalance of all data sets.

The sentiment/moment feature set, which fell between the the aforementioned feature set and the minimalist feature set in terms of data and computational resource requirements, performed best for both AAPL timespans, as well as the

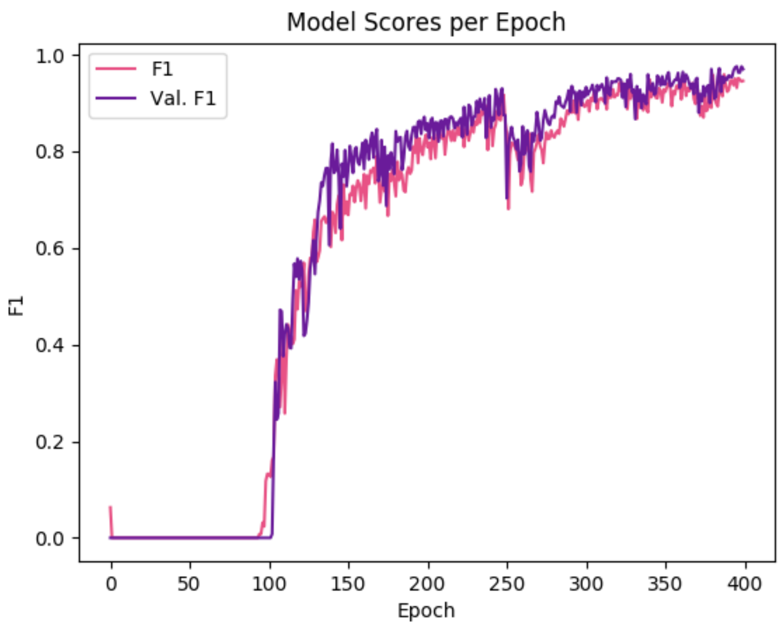


FIGURE 5.8: Training F1 progression of AAPL Sentiment 5 minute data

MSFT 5 minute model.

The most interesting result however was from the minimalist feature set, where its precision, recall, and F1 scores remained competitive with the two previous feature sets despite having feature vector half of the size of the sentiment/moment feature set, and almost a third of the size of the Kercheval and Zhang feature set. The minimalist feature set actually outperformed the other two for both time spans of the GOOG models.

Focusing on all model results divided by the 5 minute and 30 second time spans, it is evident that the 5 minute models on average outperformed the 30 second models in terms of raw predictive ability. When comparing the models with their respective random classification results however, it appears the 30 second time span models possess greater predictive ability.

The best ticker model produced was the AAPL ticker 5 minute time span, with an average F1 score across all models of 0.64, an average precision of 0.67, and an average recall of 0.61. The worst performing ticker model produced was the GOOG ticker 5 minute model, which produced average F1/precision/recall scores of 0.17/0.32/0.12, respectively.

While all models consistently outperformed their random classifiers regarding F1 and precision scores, they usually did not outperform the uniform random classified recall score of 0.50. What this indicates at a macroscopic level is that the models on average possess predictive ability to classify positive jumps with reasonable accuracy, but somewhat lack in the ability to discover all existing jumps.

TABLE 5.3: AAPL Model Results (5 minutes)

	F1	Precision	Recall
K & Z	0.63	0.71	0.59
Sent/Moment	0.65	0.67	0.64
Minimal	0.63	0.64	0.61
Random	0.15	0.09	0.50

TABLE 5.4: GOOG Model Results (5 minutes)

	F1	Precision	Recall
K & Z	0.17	0.27	0.12
Sent/Moment	0.16	0.36	0.10
Minimal	0.19	0.34	0.13
Random	0.14	0.08	0.50

TABLE 5.5: FB Model Results (5 minutes)

	F1	Precision	Recall
K & Z	0.24	0.31	0.27
Sent/Moment	0.19	0.24	0.16
Minimal	0.19	0.18	0.26
Random	0.12	0.07	0.50

TABLE 5.6: MSFT Model Results (5 minutes)

	F1	Precision	Recall
K & Z	0.42	0.49	0.37
Sent/Moment	0.46	0.71	0.34
Minimal	0.42	0.50	0.39
Random	0.15	0.09	0.50

TABLE 5.7: INTC Model Results (5 minutes)

	F1	Precision	Recall
K & Z	0.37	0.44	0.34
Sent/Moment	0.37	0.57	0.28
Minimal	0.26	0.57	0.24
Random	0.18	0.11	0.50

TABLE 5.8: AAPL Model Results (30 seconds)

	F1	Precision	Recall
K & Z	0.21	0.28	0.17
Sent/Moment	0.23	0.31	0.19
Minimal	0.21	0.29	0.16
Random	0.02	0.01	0.50

TABLE 5.9: GOOG Model Results (30 seconds)

	F1	Precision	Recall
K & Z	0.26	0.33	0.22
Sent/Moment	0.23	0.29	0.19
Minimal	0.29	0.41	0.23
Random	0.03	0.02	0.50

Table 5.10: FB Model Results (30 seconds)

	F1	Precision	Recall
K & Z	0.24	0.34	0.21
Sent/Moment	0.17	0.18	0.16
Minimal	0.18	0.29	0.17
Random	0.03	0.01	0.50

Table 5.11: MSFT Model Results (30 seconds)

	F1	Precision	Recall
K & Z	0.29	0.35	0.25
Sent/Moment	0.22	0.26	0.20
Minimal	0.27	0.30	0.26
Random	0.04	0.02	0.50

Table 5.12: INTC Model Results (30 seconds)

	F1	Precision	Recall
K & Z	0.32	0.31	0.35
Sent/Moment	0.33	0.26	0.43
Minimal	0.29	0.29	0.29
Random	0.05	0.03	0.50

Chapter 6

Conclusions

In this project, three unique feature vectors were used as input to a LSTM recurrent neural network model to predict jumps classified through the model of Lee and Mykland 2007. The three feature vectors were compared on their predictive abilities, and produced similar predictive results, with each feature vector possessing a top performance in at least one model.

The results of this project proved interesting, namely showing that the minimalist feature vector input of the LSTM model produced results on par with the other two feature vectors. This indicates that a large, computationally intensive, feature vector may not be necessary to achieve predictive results regarding jump detection. This outcome has the additional benefit of greater scalability, where future models utilizing much larger data sets can reduce construction times considerably when opting for a minimalist feature vector instead.

When considering the feature vector with incorporated daily sentiment metrics from InfoTrie 2016, it appears that, in the time scales observed, the addition of this sentiment provides little to no predictive value when constructing a jump prediction model.

Even with the preventative measures of significance level reduction for jump classification and oversampling these positive samples during the feature vector construction, the models were still hindered by a substantial classification imbalance. While the 5 minute time span models could further oversample while still meeting the classification criteria, the 30 second models approach an oversampling limit quickly due to the lack of unique available feature vectors within a 30 second time window. A possible technique to reduce this imbalance in future research would be the addition of simulated 'positive' feature vectors through generative adversarial networks (Goodfellow et al. 2014), which would have the ability to generate enough positive classified feature vectors to overcome a substantial imbalance. Additional future research considerations could involve exploring the efficacy of the models in other electronic markets with different company stocks and different time spans.

Glossary of Neural Network Terms

Activation function: A function that determines the output of a node. Common choices are the sigmoid and ReLU functions.

Backpropagation: Also known as the backward propagation of errors, backpropagation is used to calculate the gradient descent algorithm to adjust neuron weights and better tune the network model.

Dropout layer: A regularization technique which can randomly omit certain nodes to prevent overfitting.

Epoch: One complete presentation of the training set to the network during the training portion.

Input layer: The beginning layer of a neural network, the input layer is what is derived from the training set and fed into the first hidden layer and multiplied by the hidden layer's weights.

Loss function: A function which maps the values of the variables onto a real number value representing the "training loss" or "cost". The loss function is minimized during the training process.

Neuron: An elementary unit in a neural network which receives one or greater weighted inputs and sums them. The sum is then passed through an activation function.

Recurrent neural network: A class of neural network which allows a temporal aspect towards training and classification.

Weight: Applied to a specific unit, a weight is a representation of the strength of connection between other units.

Bibliography

Abma, B. (2009). Evaluation of requirements management tools with support for traceability-based change impact analysis. *Master's thesis, University of Twente, Enschede.*

Barndorff-Nielsen, O. E. and Shephard, N. (2004). Power and bipower variation with stochastic volatility and jumps. *Journal of financial econometrics* 2(1), 1–37.

Biais, B. et al. (1995). An empirical analysis of the limit order book and the order flow in the Paris Bourse. *the Journal of Finance* 50(5), 1655–1689.

Boulanger-Lewandowski, N. et al. (2012). Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392.*

Burnham, K. P. and Anderson, D. R. (2003). *Model selection and multimodel inference: a practical information-theoretic approach.* Springer Science & Business Media.

Cartea, Á. et al. (2015). *Algorithmic and high-frequency trading.* Cambridge University Press.

Chollet, F. et al. (2015). *Keras.* `https://keras.io`.

Davis, W. (2008). Analyzing and Applying Existing and New Jump Detection Methods for Intraday Stock Data. PhD thesis. Duke University; Durham.

Duchi, J. et al. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12(Jul), 2121–2159.

Fletcher, T. and Shawe-Taylor, J. (2013). Multiple kernel learning with fisher kernels for high frequency currency prediction. *Computational Economics* 42(2), 217–240.

Friedman, J. et al. (2001). *The elements of statistical learning.* Vol. 1. 10. Springer series in statistics New York, NY, USA:

Garcia, D. (2013). Sentiment during recessions. *The Journal of Finance* 68(3), 1267–1300.

Gençay, R. et al. (2001). *An introduction to high-frequency finance.* Elsevier.

Gers, F. A. et al. (1999). Learning to forget: Continual prediction with LSTM.

Goodfellow, I. et al. (2014). Generative adversarial nets. In: *Advances in neural information processing systems*, 2672–2680.

Graves, A. and Schmidhuber, J. (2009). Offline handwriting recognition with multidimensional recurrent neural networks. In: *Advances in neural information processing systems*, 545–552.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation* 9(8), 1735–1780.

Hochreiter, S. et al. (2001). *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.*

Huang, X. and Tauchen, G. (2005). The relative contribution of jumps to total price variance. *Journal of financial econometrics* 3(4), 456–499.

Hull, J. C. (2003). *Options futures and other derivatives.* Pearson Education India.

InfoTrie (2016). *FinSentS Web News Sentiment.* `http://www.infotrie.com/finsents/`.

Jaffe, J. F. (1974). Special information and insider trading. *The Journal of Business* 47(3), 410–428.

Janocha, K. and Czarnecki, W. M. (2017). On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659.*

Kercheval, A. N. and Zhang, Y. (2015). Modelling high-frequency limit order book dynamics with support vector machines. *Quantitative Finance* 15(8), 1315–1329.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Längkvist, M. et al. (2014). A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters* 42, 11–24.

Lee, S. S. and Mykland, P. A. (2007). Jumps in financial markets: A new nonparametric test and jump dynamics. *The Review of Financial Studies* 21(6), 2535–2563.

Lin, J.-C. and Howe, J. S. (1990). Insider trading in the OTC market. *The Journal of Finance* 45(4), 1273–1284.

Lipton, Z. C. et al. (2014). Thresholding Classifiers to Maximize F1 Score. *arXiv preprint arXiv:1402.1892.*

Mäkinen, M. (2017). Neural Networks in Stock Price Jump Prediction. *Master's thesis, Tampere University of Technology.*

Malkiel, B. G. and Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *The journal of Finance* 25(2), 383–417.

Mamaysky, H., Glasserman, P., et al. (2016). Does Unusual News Forecast Market Stress? *The Office of Financial Research Working Paper* (16-04).

Martin Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org.

Merton, R. C. (1976). Option pricing when underlying stock returns are discontinuous. *Journal of financial economics* 3(1-2), 125–144.

Murphy, K. P. (2012). Machine learning: a probabilistic perspective.

Nasdaq (2014). *NASDAQ TotalView ITCH 4.1 Interface Specification.*

Nasdaq (2018). *The Nasdaq Opening and Closing Crosses.*

Pang, B., Lee, L., et al. (2008). Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval* 2(1–2), 1–135.

Sharcnet (2017). *Graham Cluster.* `https://www.sharcnet.ca/help/index.php/Graham`. Accessed: 2019-01-28.

Singwi, K. and Sjölander, A. (1960). Resonance absorption of nuclear gamma rays and the dynamics of atomic motions. *Physical Review* 120(4), 1093.

Sirignano, J. A. (2018). Deep learning for limit order books. *Quantitative Finance*, 1–22.

Srivastava, N. et al. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1), 1929–1958.

Sun, A. et al. (2016). Trade the tweet: Social media text mining and sparse matrix factorization for stock market prediction. *International Review of Financial Analysis* 48, 272–281.

Swaney, C. (2018). *Prickle.* `https://github.com/cswaney/prickle`.

Tetlock, P. C. et al. (2008). More than words: Quantifying language to measure firms' fundamentals. *The Journal of Finance* 63(3), 1437–1467.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* 4(2), 26–31.

Tsantekidis, A. et al. (2017). Using deep learning to detect price change indications in financial markets. In: *Signal Processing Conference (EUSIPCO), 2017 25th European.* IEEE, 2511–2515.

Vapnik, V. N. (1999). An overview of statistical learning theory. *IEEE transactions on neural networks* 10(5), 988–999.

Vu, L. and Le, T. (2017). A lexicon-based method for Sentiment Analysis using social network data.

Yap, B. W. et al. (2014). An application of oversampling, undersampling, bagging and boosting in handling imbalanced datasets. In: *Proceedings of the First International Conference on Advanced Data and Information Engineering (DaEng-2013).* Springer, 13–22.

Zhou, Y. et al. (2016). ECNU at SemEval-2016 Task 4: An empirical investigation of traditional NLP features and word embedding features for sentence-level and topic-level sentiment analysis in Twitter. In: *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, 256–261.