VRIJE UNIVERSITEIT AMSTERDAM

An ensemble of Recurrent Neural Networks for High-Frequency Stock Market Classification

by

Ioannis Tsiamas

supervised by Dr. S. Borovkova

A thesis submitted in partial fulfillment for the
degree of Master of Science

in the
Business and Economics
Department of Finance

March 2018

Declaration of Authorship

I, Ioannis Tsiamas, declare that this thesis titled, 'High-Frequency Stock Market Classification using an ensemble of Recurrent Neural Networks' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Giannis Tsiamas

Date: March 2018

VRIJE UNIVERSITEIT AMSTERDAM

Abstract

Business and Economics
Department of Finance

Master of Science

by Ioannis Tsiamas

Predicting the stock market is a complicated task that has attracted the interest of researchers and investors for many years. Although new contributions are made constantly to the field, no apparent solution exists. The task at hand is even more complex when it comes to applying short-term, high-frequency predictions. The difficulty stems from the extreme stochastic nature of the markets, as well as their rapidly changing statistical properties. For the purpose of this thesis, we approach the topic of high-frequency stock market predictions, by proposing an ensemble of Recurrent Neural Networks. The proposed ensemble operates in an online way, weighting the individual models proportionally to their performance in predicting the past observations. The performance of the models is measured by Area Under the Curve of the Receiver Operating Characteristic. We evaluate the predictive power of our model on several US large-cap stocks and benchmark it against Lasso and Ridge logistic classifiers. The proposed model is found to perform better when compared to the before-mentioned benchmarks and equally weighted ensembles.

Contents

List of Figures

List of Tables

Abbreviations

AI	**A**rtificial **I**ntelligence
ANN	**A**rtifical **N**eural **N**etwork
ARIMA	**A**uto**R**egressive **I**ntegrated **M**oving **A**verage
AUC	**A**rea **U**nder the **C**urve
BPTT	**B**ack**P**ropagation **T**hrough **T**ime
EMH	**E**fficient **M**arket **H**ypothesis
FPR	**F**alse **P**ositive **R**ate
LR	**L**ogistic **R**egression
LSTM	**L**ong **S**hort-**T**erm **M**emory
MLP	**M**ulti**L**ayer **P**erceptron
ROC	**R**eceiver **O**perating **C**haracteristic
RMSProp	**R**oot **M**ean **S**quare **Prop**agation
RNN	**R**ecurrent **N**eural **N**etwork
SGD	**S**tochastic **G**radient **D**escent
SVM	**S**upport **V**ector **M**achine
TPR	**T**rue **P**ositive **R**ate

Chapter 1

Introduction

1.1 Efficient Market Hypothesis

Financial markets have been in the center of attention for many decades, for investors and researchers alike. In that time, a rich, in contributions, research field has been created, but the answer to a particular question remains debatable. People have been arguing whether the underlying nature of stock markets, allows for one to make accurate and consistent predictions. This long-lasting debate has produced a significant amount of research on the subject, with papers, most of the times, showing contradictory evidence to one another.

The first, and probably most important theory revolving around that debate is the Efficient Market Hypothesis (EMH), developed by E. Fama in 1970 [1], for which he later received the Nobel price in economics. The core idea of the EMH is that all available information is already incorporated into market prices and thus, prices reflect their true values. In other words, no individual can receive profit by making predictions on future prices, since future information is not yet available. In that sense, prices are not predictable, but random. Prior to the EMH, Fama also argued that the prices of financial assets follow a process of Random Walk [2]. According to the Random Walk theorem, future prices are basically the cumulative sum of a random series of numbers, rendering them unpredictable in any way.

Critics of the theories of Fama, step on the strong assumptions that lay behind the EMH. Most notable of them is the assumption that market agents are rational. Fama suggests that even if some participants are irrational, the vast majority being rational will drive the price towards the true value. On the other hand, there is supporting evidence that there are times, at which, even the collective actions of people are irrational. The field

of Behavioural Finance has shown light to many such phenomena, of agents following certain behaviors, causing the temporal emergence of patterns, that in turn "break" the EMH and allow for price predictability. These deficiencies are the outcomes of a mixture of behavioral fallacies and cognitive biases. A notable example of such pattern is the January effect [3], according to which, prices tend to increase on average more on January, than on any other month. Although the effect of these patterns diminishes or even disappears after they become widespread [4], their emergence, by itself, shows that the market can be predictable.

With the age of information, the increasing availability of trade-to-trade data and the development of machine learning algorithms that can handle large amounts of data, research in stock market predictability has moved from daily, to high-frequency basis. Accordingly, in this thesis, we employ a collection of machine learning algorithms to identify and take advantage of small "January effects" in the US stock market.

1.2 Machine Learning Algorithms and Artificial Neural Networks

Machine learning incorporates algorithms that can learn patterns and make predictions on data, without an explicit set of rules. More formally, T.M. Mitchell in [5] describes the process as "*an algorithm learns from experience E, with respect to some class of tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with experience E*".

Machine learning can be primarily categorized to either unsupervised or supervised learning. In unsupervised learning, the task of the algorithm is to infer hidden properties of the given unlabeled data. Popular applications are clustering, dimensionality reduction, and feature engineering. On the other hand, in supervised learning, the algorithm learns a function $f(\cdot)$ that maps inputs x to targets y. Tasks of supervised learning include regression and classification.

In this research, we primarily applied supervised learning and classification. In classification tasks, targets are categorical and the different categories that an instance can belong to, are called classes, while the attributes of the instance that are used to classify it, are called features. For m classes and n features, the process can be defined as:

$$f : x \to y \quad x \in \mathbb{R}^n, y \in \mathbb{N}^m \tag{1.1}$$

1.2.1 Artificial Neural Networks

A popular class of machine learning algorithms is the Artificial Neural Network (ANN). The Perceptron (neuron), which represents a neural network at its most basic structure, was first introduced by Rosenblatt in 1950 [6]. Given a vector of inputs $\boldsymbol{x}$, a vector of weights $\boldsymbol{w}$, a bias b and an activation function $f(\cdot)$, the output $\boldsymbol{o}$ of the Perceptron is produced by $f(\boldsymbol{x} \times \boldsymbol{w} + b)$, where $\boldsymbol{x} \times \boldsymbol{w}$ denotes the inner product of the two vectors. In classification tasks, the activation is an m-step function, with m being the number of possible classes. Weights and biases are called the parameter of the network and are usually denoted by $\boldsymbol{\theta}$. Multiple perceptrons can create a single layer neural network, or Multi-layer Perceptron (MLP), which can be further generalized to N layers. Classic ANN architectures, where the inputs flow only one way, are called feed-forward.

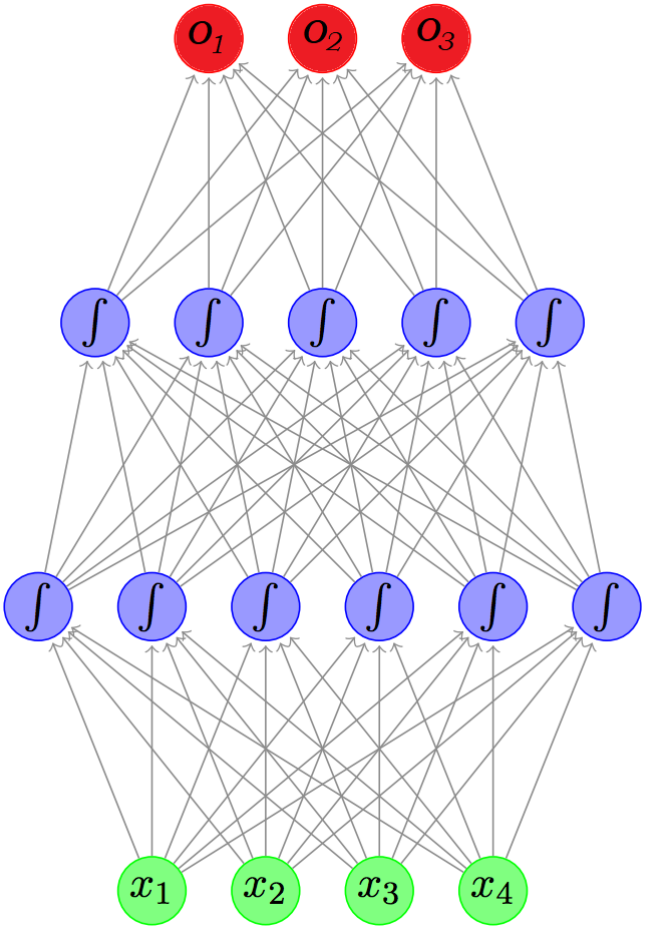

FIGURE 1.1: A feed-forward neural network

The network is composed of an input layer(green) of size 4, 2 hidden layers(blue) of size 6 and 5, and an output layer(red) that can classify the instance to 3 different classes.

The training method for such models, the Backpropagation algorithm, was invented in 1970 [7] and is a special case of Automatic Differentiation. Firstly, a loss function $\mathcal{L}$ is defined as a measure of divergence between outputs $\boldsymbol{o}$ and targets $\boldsymbol{y}$. Then, backpropagation basically uses the chain rule 1.2 of differentiation to distribute the error signals from the derivatives of the loss function, with respect to the parameters $\boldsymbol{\theta}$, all the way back to the input layer.

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \frac{\partial g(x)}{\partial x} \tag{1.2}$$

Each parameter gets updated using its error signal and a learning rate η. The process is repeated T times, until convergence is achieved.

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}_{ij}^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} \tag{1.3}$$

Given the above, the ultimate purpose of a neural network boils down to an optimization task, that of minimizing the error from the loss function with respect to parameters θ of the network.

$$\theta^* = argmin_{\boldsymbol{\theta}} \sum_{x,y \in \boldsymbol{X},\boldsymbol{Y}} \mathcal{L}(y, f(x; \boldsymbol{\theta})) \tag{1.4}$$

Where $f(\cdot)$ is the neural network as a function.

Although the foundations of neural networks have been around for almost half a century, they only became popular some years ago. The case was that they were models, too computationally expensive to be trained, and at the same time were outperformed by other machine learning algorithms, such as Support Vector Machines (SVM) [1]. The advances in the computational capabilities of modern machines, that came through the decreasing prices of Graphical Processing Units (GPUs) allowed for deeper and deeper architectures that gave birth to deep learning, which eventually outperformed the classic machine learning algorithms. Many people attribute the growth of Deep Learning to the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012. ILSVRC is an annual challenge, where participating teams compete for the best accuracy in a number of computer vision tasks. In the 2012 competition, AlexNet, a network of 650,000 neurons, achieved the highest accuracy scores, by far surpassing all the other contenders [9]. This

[1] supervised learning algorithm that uses a set of hyperplanes to classify data points in a higher-dimensional space [8]

was a turning point for deep learning, and research shifted towards it, in computer vision and other fields.

1.2.2 Recurrent Neural Networks

While feed-forward networks work mainly with static data, Recurrent Neural Networks (RNN) can work with temporal (sequential). An RNN keeps a memory of past inputs and thus can achieve the dynamic temporal behavior needed to model financial time series. In this thesis, we use a particular type of RNNs, the Long Short-term Memory (LSTM) network, which is an updated version of the plain RNN that solves the issue of fading memory. The concepts previously described for feed-forward architectures can be extended to the LSTM and training can be carried out by the Backpropagation Through Time (BPTT) algorithm [10]. A more detailed description of LSTM networks is provided in later sections.

1.2.3 Ensemble learning

Deep networks and Recurrent Neural Networks have a large capacity for discovering patterns and are able to learn really complex functions. One such demonstration is their use in the Autoencoder [11][12] family of models, where a neural network learns a representation of input x to a latent space $\mathcal{Z}$ and then reconstructs x', similarly to x. Although they are powerful models, the stock market requires special consideration due to its highly stochastic and dynamic nature. In machine learning, this is known as concept drift (or non-stationarity in econometrics), according to which, the statistical properties of the target variable change over time, in unforeseen ways. Therefore, a single model is incapable of accurately capturing the dynamics of financial time series. Thus, in order to achieve better predictions, we used an ensemble of LSTMs. An ensemble is a collection of some base classifiers (LSTMs in this case), that are usually trained on subsets d of dataset $\mathcal{D}$, and can collectively achieve higher performance metrics than an individual classifier [13]. The most popular ensemble method is the Random Forest classifier [14], where the prediction of the ensemble is the equally weighted combination of the predictions of several decision trees.

1.3 Predictive framework

For the purpose of this thesis, 22 US stocks were selected, to apply our predictive framework. We used one year of raw trade data, that was cleaned and aggregated into 5-minute

intervals, amounting to roughly 19,000 observations per stock. At every moment, each stock was labeled as "Buy" (positive class) or "Sell" (negative class), according to the direction of the price with respect to the previous one. For every stock, we furthermore used its primary competitor, thus establishing a universe of 44 stocks. By cross-sectional aggregation, we additionally created eight sector datasets and a summarizing, universe dataset. We relied heavily on feature engineering by constructing a large number of technical indicators, on different time-frames and also on stock and sector level.

The first month of our dataset was solely used for feature engineering. For the other 11 months, we operated on a rolling way, in which one month worth of data are used for training the models, one week for validating their performance and then predictions are applied on the following week. This amounts to 21 training-validation-testing periods per stock.

For every period and for each stock, we trained 12 stacked-LSTMs[2], on different subsets of the training periods. The predictions of each model were evaluated by the Area Under the Curve (AUC) score of the Receiver Operating Characteristic (ROC)[15]. The ensemble predictions for each testing period were a weighted combination of the 12 trained models. The weights assigned to each model were proportional to their AUC score on the past week of instances. Finally, the overall performance of our predictive framework on a stock is measured by the average AUC score of the ensembles for the 21 testing periods.

The rest of the thesis is structured as followed. 2nd chapter is devoted to related works on machine learning methods for predictive modeling in financial markets, the 3rd chapter is for data collection and feature engineering, in the 4th chapter we analyze in detail the methods used and 5th chapter we present and discuss our results. The last two chapters are dedicated to our conclusions and future research.

[2]LSTMs with more than one layer

Chapter 2

Related Work

2.1 Statistical time-series modelling

Time-series models have been the traditional choice for modeling financial time series, treating the task of prediction as a regression problem. The Autoregressive-Moving Average (ARMA) [16] is one the most popular, among a large family of such models. Proposed by P. Whittle in 1951, ARMA models are able to capture the stochastic nature of financial time series. They are composed of two parts, the Autoregressive part of order p, AR(p), which involves p lagged versions of the target variable, and the Moving Average part of order q, MA(q), which involves q error terms in the past. These two are usually coupled with an Integrated part of order d, where the target variable is differenced d times until stationarity is achieved. ARIMA models became widely known, after Box-Jenkins methodology [17] was proposed for choosing the best p and q parameters. ARMA models were used to successfully forecast returns in a not so competitive market, like the Helsinki Stock Exchange [18].

Autoregressive Conditional Heteroskedasticity (ARCH) [19] is another time series model, in which the variance of the error terms is described as a function of the past errors. Its generalized version (GARCH) [20], also involves an ARMA model for the error variance. A. Goyal found that the simple ARMA outperforms GARCH models for predicting stock the S&P500 Index volatility [21].

Vectorized Autoregressive-Moving Average (VARMA) and Vectorized Generalized Autoregressive Conditional Heteroskedasticity (V-GARCH) are the multivariate versions of the previously mentioned models, which can benefit greatly from the correlations of the time series included. Oil prices and volatility were found to have predictability over stock market returns, using vectorized models [22]. Several versions of V-GARCH models were tested for daily predictions on exchange rates and stock indices [23]. Predictions

in financial time series have also been achieved using state-space representations [24][25] and Generalized Autoregressive Score (GAS) models [26].

2.2 Machine Learning applications in Business

Machine learning methods have been used in a wide range of business applications, with most of the times, outperforming traditional methods. One of their first use was in the domain of credit scoring, where the task is to classify loan applications as "good" (accepted) or "bad" (rejected). In a 1992 study, Tam et al. [27] compared the performance of ANNs and k-Nearest Neighbours (k-NN) [1] against the industry standard of Logistic Regression (LR), and found that the ANNs were better in terms of both robustness and accuracy. Lai et al. [29] used an ensemble, with ANNs as base classifiers, to further enhance predictability in loan classification. Bagherpour [30] analyzed 20 million mortgages between 2001-2016 and achieved the highest and most stable AUC score, ranging from 90% to 95%, by applying Factorization Machines [2]. In an even larger scale study, Sirignano et al. [32] used deep networks and a dataset of 120 million mortgages, to achieve an overall AUC score of 94.4%.

Another application of machine learning algorithms in business, is in the task of predicting bankruptcy outcomes. Zang et al. [33] reported accuracy rates of 2-3% higher when compared fully connected ANNs with the plain LR for predicting firm bankruptcies. In two recent studies regarding firm bankruptcy prediction, there have been contradictory results in terms of model comparison, however, different evaluation metrics were used. Brabozaa et al. [34] argued that bagging and boosting methods were more accurate, while Iturriaga et al. [35], achieved better performance using a combination of MLPs and Self Organizing Maps [3].

A heavily machine learning applied field is, also, that of fraud detection, which is characterized by highly skewed data. Awoyemi et al. [37], compared k-NN, Naive Bayes [4] and LR and found that k-NN achieved better performance. Another field, that was developed fairly recent, is that of Recommender Systems (RS), with applications in most web-based services. In a 2013-study, Bobadilla et al. [38] compiled a comprehensive survey of machine learning algorithms with use in RS.

[1] non-parametric algorithm that classifies an object according to the majority class of its k nearest neighbours [28]

[2] non-linear algorithm that models variables according to a lower dimensional mapping [31]

[3] class of neural networks that achieves dimensionality reduction by mapping variables to a two-dimensional space [36]

[4] probabilistic classifier based on Bayes' probability theorem

2.3 Machine Learning for financial time series prediction

Most of the research regarding machine learning and deep learning applications in financial time series predictions is quite recent. Some early works include that of Baestanes et al. [39] and Refenes et al. [40] who used simple ANN architectures and compared their performance to the LR. Huang et al. [41] found that SVMs can achieve better results than traditional methods, while Pai et al. [42] proposed a hybrid ARIMA - SVM method.

Latter contributions include Hegazy et al. [43], who used Particle Swarm Optimization [5] to fine-tune the hyperparameters of an SVM regressor, achieving significantly smaller Mean Squared Errors (MSE) on several US stocks. Nelson et al. [45] trained LSTM networks on 15-minute-interval observations, for several BOVESPA (Sao Paolo stock exchange) stocks, and reported accuracy metrics of 53-55% regarding the next direction of the prices. Fisher et al. [46] conducted a large-scale research, using daily S&P500 data from 1992 to 2015. They used LSTMs, Random Forests, deep networks and LRs, and found that a trading strategy, based on the predictions of LSTM, was the most profitable.

Qu et al. [47], assuming that high-frequency returns trigger periodically momentum and reversal, designed a new SVM kernel method [6] appropriate to forecast high-frequency market directions and applied their method on the Chinese CSI300 index. Their results were significantly better when compared with the Radial Basis Function(RBF) [7] kernel and the Sigmoid kernel.

One of the most remarkable contributions to deep learning for stock predictability is that in Bao et al. 2017 [48]. They proposed a predictive framework composed of three parts. First, they apply a Wavelet Transformation(WT) [8] to the financial dataset(price data, technical indicators, macroeconomic data). Then the de-noised data are passed through stacked-Autoencoders [9] to generate meaningful features and finally LSTM networks are used for forecasting. They used daily observations from six stock indices and test the profitability of their models by purchasing the corresponding future indices, according to the predictions of the model. They showed that their proposed framework was better in terms of both accuracy and predictability when compared to simpler methods.

[5] nature-inspired algorithm that uses a collection of candidate solutions as well as their respective positions and velocities to find the optimal solutions [44]

[6] kernel functions enable higher dimensional operations without operating in the actual higher dimensional space

[7] maps inputs to higher dimensional space using the Euclidean distance

[8] signal processing, transforms a series from time to the frequency domain.

[9] many-layered networks that reconstruct the input

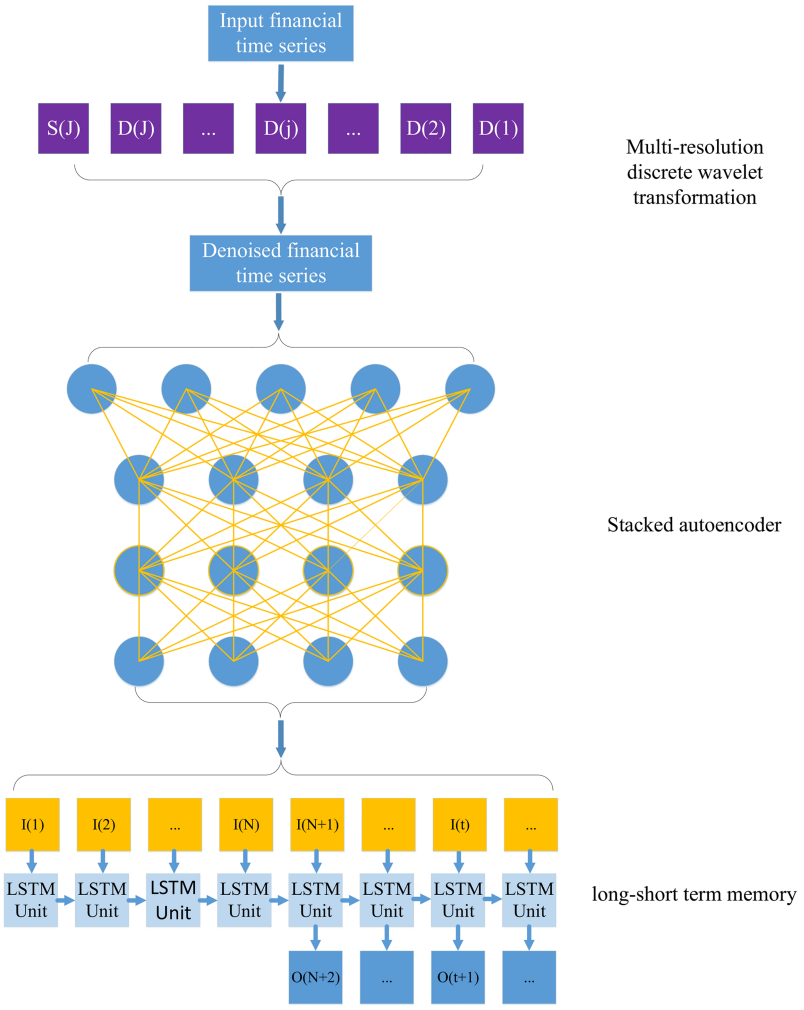

FIGURE 2.1: Proposed predictive framework in Bao et al. 2017

Another impressive research, that inspired this thesis, is that by M.D. Rechenthin in his Ph.D. dissertation [49]. He demonstrates the existence of predictability in high-frequency trading by analyzing certain patterns, using their conditional probabilities. Furthermore, he extensively touches upon the subjects of concept drift and the importance of prediction-speed in a high-frequency strategy, His proposed framework involves creating a pool of thousands of base classifiers(SVMs, ANNs, DTs), and interchanging between models depending on their performance. The target labels in his research are "strong sell", "hold" or "strong buy". He reported AUC scores from 0.518 to 0.579 (average of 0.531) for 34 US energy stocks, operating on 1-minute intervals in the last seven months of 2012. He also found that the introduction of trained classifiers from within a sector, in the stock's pool, can significantly increase predictability, achieving an average AUC score of 0.548.

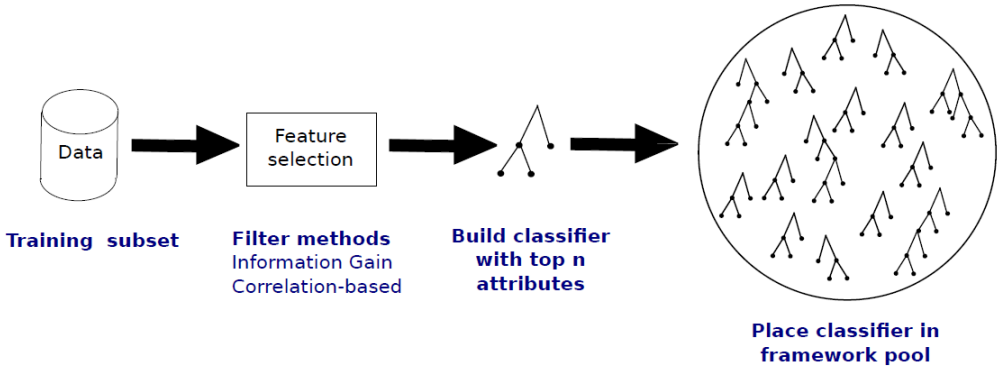

FIGURE 2.2: Creating the classifier pool in M. D. Rechenthin 2014

Chapter 3

Data

3.1 Choice of data

For the purpose of this research we used one year (2014) of high-frequency historical data, for 22 large-cap [1] US stocks, traded in either in NYSE [2] or NASDAQ [3]. Aiming to take advantage of strong cross-correlations between them, we also included the primary competitor for each of the 22 stocks. In addition, these stocks belong to one of the eight major sectors of the US stock markets [4], which allowed us to construct sector datasets, by aggregating the stock datasets cross-sectionally. A summary of the selected stocks is presented in Table A.1.

Our choice of using large-cap US stocks for our research is motivated mainly by two reasons; lots of data and high liquidity. The abundance of data that characterizes US large-cap stocks, even in small time intervals, is essential for training our models, since a shortage or absence of trading data can disrupt the learning procedure. Of utter importance is also the liquidity of a stock. High liquidity means that an individual trader's decisions will have a negligible impact on the market, which in turn translates to low trading costs. Maintaining low trading costs is essential for a high-frequency trading strategy to be effective. Although in this research, we do not examine profitability, it ensures that our methods are as realistic, as possible.

[1] more than 10$ billion in market capitalization

[2] New York Stock Exchange https://www.nyse.com

[3] National Association of Securities Dealers Automated Quotations https://www.nasdaq.com

[4] according to market capitalization: Capital Goods, Consumer Non-Durables, Consumer Services, Energy, Finance, Health Care, Public Utilities and Technology

3.2 Collecting & Cleaning the data

Data were obtained from the NYSE Trade-and-Quote(TAQ) database, which is accessible through the WRDS [5] interface. The TAQ database contains raw trade information for US stocks at a micro-second accuracy. We consider only trades that occurred between the regular opening hours of NYSE and NASDAQ, more specifically, between 09:30 and 16:00.

Raw trade data files contain the date and time that the trade took place, the price of the trade and its size (in stocks). Of importance are additionally the correction indicator and the trade conditions of the trade. A correction indicator can take several values, with 0 and 1, meaning that the trade was valid. Other values indicate that there was a complication with the trade and it was canceled. The trade condition specifies the existence and nature of special conditions, under which the trade took place. The vast majority of trades are completed under normal conditions or are intermarket sweep orders [6]. An overview of the possible trade condition indicators, along with their effects on open/close prices, can be found in Table A.2. Below we present a snapshot of raw trade data for stock ABT.

TABLE 3.1: Raw trade data

A random part of the raw data for ABT at the opening of 02-01-2014. A correction indicator of 0 suggests that the trade is valid. A trade condition of @ means that the trade took place under normal conditions, while *F* indicates that the trade was a intermarket sweep order.

Date & Time	Price	Correction Indicator	Trade Condition	Size
02-01-2014 9:32:23	38.37	0	@	100
02-01-2014 9:32:23	38.37	0	@	100
02-01-2014 9:32:23	38.37	0	@	400
02-01-2014 9:32:24	38.37	0	@	100
02-01-2014 9:32:24	38.37	0	@	100
02-01-2014 9:32:24	38.37	0	@	200
02-01-2014 9:32:24	38.37	0	F	100
02-01-2014 9:32:24	38.37	0	@	100
02-01-2014 9:32:24	38.36	0	F	100
02-01-2014 9:32:24	38.36	0	F	100
02-01-2014 9:32:24	38.36	0	F	100

[5] Wharton Research Data Services https://wrds-web.wharton.upenn.edu/

[6] lets a trader to purchase stock from several exchanges until the given order size is satisfied

The raw data were processed, to clean them from "bad" trades, and aggregate them into 5-minute intervals. "Bad", are considered the trades that are either out-of-sequence, late [7], corrected, or completed under special conditions. The removal of these trades ensures that data contains the least amount of noise. Thus, during the aggregation process, we took into account only trades that were correct, were completed under regular conditions or were intermarket sweep orders. For each stock, 5-10% of the total trades were cleaned.

Furthermore, during data aggregation, we constructed some basic features, regarding prices, price differences, and trading volumes. An overview of these features is presented in Table A.3. In addition, we did not consider early closing days [8], in order to maintain a consistent number of observations per day (78). Prices were also adjusted for stock splits [9]. Spit information can be found in Table A.4. After aggregation, a total of 18252 observations per stock were produced. A snapshot of the new, aggregated data is shown in Table 3.2.

TABLE 3.2: Data after aggregation

Part of the aggregated data for stock ABT. The triple dot indicates the presence of more features. More info can be found in Table A.3

Date & Time	**Open**	**Close**	**High**	**Low**	**Trades**	**Volume**	**. . .**
02-01-2014 09:30	38.15	38.33	38.4	38.0901	687	93993	...
02-01-2014 09:35	38.34	38.25	38.34	38.21	796	137549	...
02-01-2014 09:40	38.251	38.31	38.31	38.13	929	176642	...
02-01-2014 09:45	38.31	38.3	38.4	38.3	331	51007	...
02-01-2014 09:50	38.2973	38.33	38.34	38.22	354	41597	...
02-01-2014 09:55	38.33	38.295	38.33	38.26	423	48978	...
02-01-2014 10:00	38.29	38.14	38.295	38.12	530	78376	...
02-01-2014 10:05	38.14	38.155	38.18	38.13	351	50904	...
02-01-2014 10:10	38.16	38.11	38.2	38.11	540	100286	...
02-01-2014 10:15	38.113	38.1	38.13	38.09	323	48358	...

It is worth mentioning that the data are characterized by only a few missing values. Specifically, due to a network failure on 30 October 2014, between times 13:07 and 13:34 [50], half of the NYSE stocks were not trading. Furthermore, GM and GILD stocks have no trading activity in periods 30 June 2014 14:30-14:50 and 27 October 2014 09:45-10:25 accordingly; both for unclear reasons. We corrected missing data, by propagating forward the last valid price, in price related features, and filled with zeros the rest.

[7] outside of normal trade hours

[8] regular market hours end at 13:00, usually Thanksgiving, or Christmas

[9] corporate action that decreases the number of shares

3.3 Sector and Universe datasets

One of the goals of this research is to take advantage of the possible interconnectedness in markets, to boost our models. Subsequently, we constructed sector datasets and an overall universe dataset using the basic features of the 44 stocks. The shares outstanding for each stock were obtained from Ycharts [10] and used to create market capitalization series, that can be then translated into weighting factors for each stock within its sector and within the universe.

Market capitalization at any moment t, for stock s_i, is calculated as the product of shares outstanding and price at t. Furthermore, market capitalization for sector $\mathcal{S}_j$ is calculated as the sum of market capitalization for each stock s_i within it. Finally, the weighting factors for each stock are determined by the fraction of its market capitalization to the overall market capitalization of the sector and the universe accordingly.

$$Mcap_t^{(s_i)} = SharesOutstanding_t^{(s_i)} \; P_{close,t}^{(s_i)} \tag{3.1}$$

$$Mcap_t^{(\mathcal{S}_j)} = \sum_{s_i \in \mathcal{S}_j} Mcap_t^{(s_i)} \tag{3.2}$$

$$Weight_t^{(s_i)} = \frac{Mcap_t^{(s_i)}}{Mcap_t^{(\mathcal{S}_j)}} \tag{3.3}$$

Using the weighting factors, we constructed the basic features [11] presented in Table A.3 on sector and universe level. For example, price-related features were calculated as the weighted average of individual stock prices, while volume related were calculated by summing the individual volumes.

3.4 Target Variable

The objective of this research is to provide a framework that, at every instance, assigns probabilities of an upward or downward move, regarding the direction of the price in the next 5 minutes. Thus, periods of positive returns are labeled as "Buy", while periods with zero or negative returns are labeled as "Sell".

[10] financial data platform https://ycharts.com/

[11] with the exception of price difference related features

$$R_t = \frac{P_{close,t} - P_{open,t}}{P_{open,t}} \tag{3.4}$$

$$Y_t = \begin{cases} 1 & R_t > 0 \qquad \text{class "Buy"} \\ 0 & R_t \leq 0 \qquad \text{class "Sell"} \end{cases}$$

In high-frequency settings, an important number of instances results in zero returns, thus causing the class "Sell" to be overpopulated in comparison to class "Buy". This phenomenon is called class imbalance and may affect the training procedure if not treated. It is further discussed in Chapter 4. Approximately every stock experiences an average ratio of 46% Buy and 54% Sell, although it may vary from time to time. In Figure 3.1 we provide an illustration of the class "Buy" probabilities for stock ABT, considering rolling windows of 78, 234 and 1326 past observations. Given the fast-shifting probability distributions of our target variable, it becomes clear how important the use of multiple models becomes, in order to capture this variability.

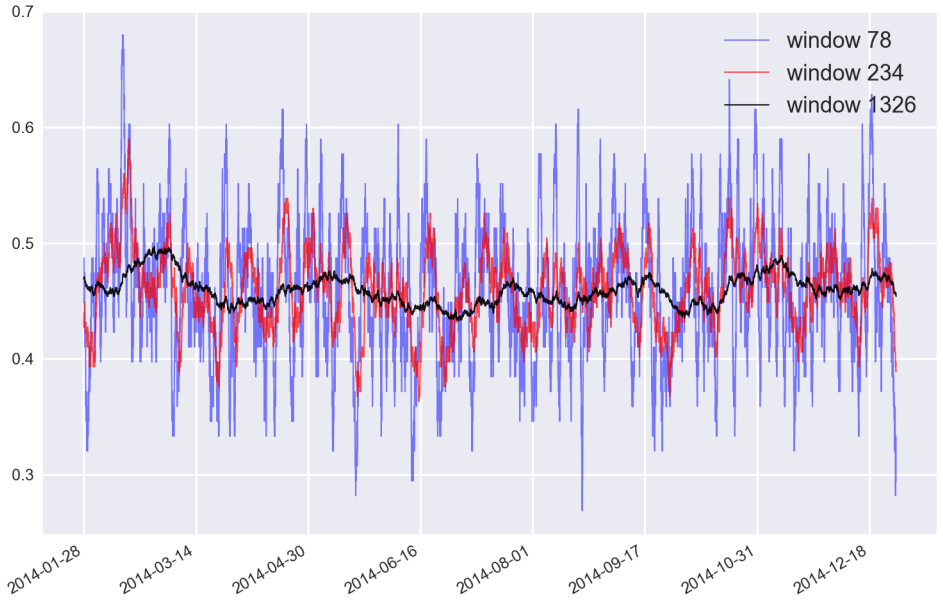

FIGURE 3.1: Probability distribution of class "Buy"

Although the relativelly short period of 78 observations is really volatile and can not provide much insight, some clear trends can be found by looking at the 234 and 1326 windows.

To further demonstrate the difficulty of our task, we provide a scatter plot of the 3 major principal components [12] for stock ABT. The classes "Buy" and "Sell" do not seem to be separable, at any obvious way, at least in this three-dimensional space.

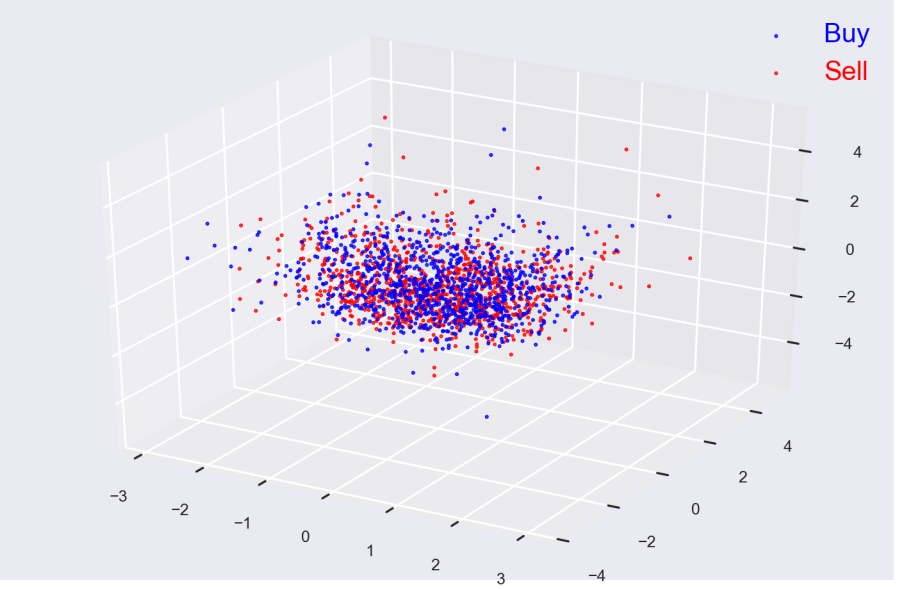

FIGURE 3.2: Scatter-plot of "Buy" and "Sell" classes

1000 observations from each class of stock ABT, scattered according to the first three principal components.

3.5 Feature Engineering

In order to enhance our models' predictability, we constructed a large number of technical indicators. Technical indicators are measures used in technical analysis, that aim to identify "buy" or "sell" signals in charts and graphical representations of stocks. They can be categorized in trend, volatility, volume and momentum indicators. People have been trading based on them by using heuristic rules. Although their true value has been argued, a model should be able to benefit from them, since they can provide a "summary" of a stock on different time-windows. Using technical indicators as features in machine learning is known as quantitative technical analysis. The indicators used in this research, along with their formulas and selective figures are presented in Section A.1.

[12] Principal Components Analysis (PCA) is a linear method of dimensionality reduction, according to which a set of n features are transformed into m uncorrelated features

Aside from technical indicators, we used rolling regressions, as features, on the percentage change in prices (P_{close}, P_{high}, P_{low}, P_{vwap}). More specifically we applied AR(1) 3.5 models with a rolling window of 1326 (approximately one month of observations).

$$R_t = c + \phi R_{t-1} + \epsilon_t \qquad \epsilon_t \sim \mathcal{N}(0,1) \tag{3.5}$$

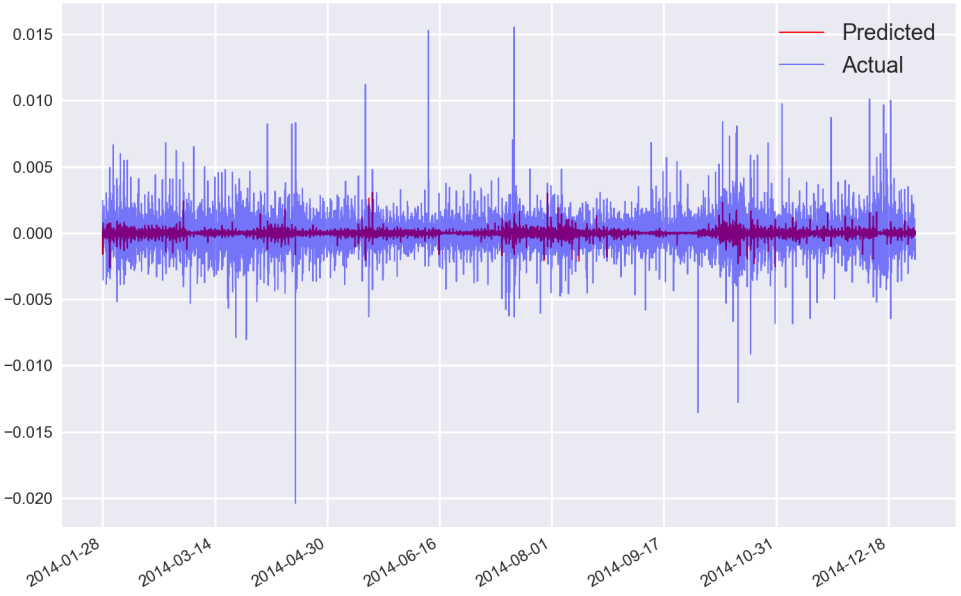

FIGURE 3.3: AR(1) Predictions

Rolling AR(1) model for the returns of ABT. Mean Squared Error(MAE) = 0.00075

Furthermore, the probability distributions and conditional probability distributions of target class "Buy" were included as features. We used rolling windows of 36, 72, 234, 1326 observations, for probabilities $\mathcal{P}(y_t = Buy)$ and 72, 234, 1326 observations, for conditional probabilities $\mathcal{P}(y_t = Buy|y_{t-1} = Sell)$, $\mathcal{P}(y_t = Buy|y_{t-1} = Buy)$. Certain trends can be observed along the span of one year, which can potential aid our models' predictive power.

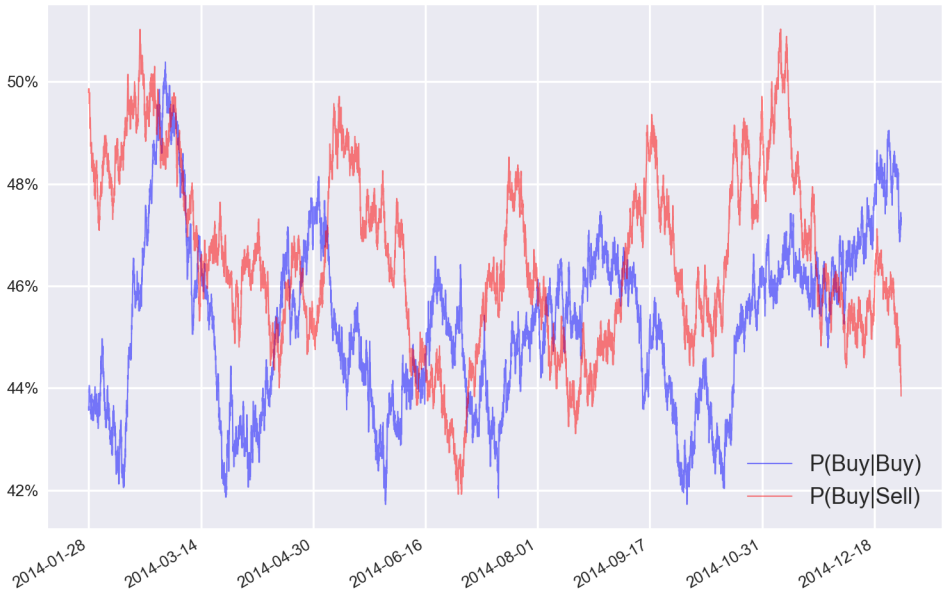

FIGURE 3.4: Conditional Probabilities

Conditional Probabilities for ABT, using a rolling window of 1326 observations

Trading intensity can vary from time to time in the stock markets, especially in a high-frequency setting. There are specific parts of the day, that are more trading intensive than other. Figures 3.5 3.6 indicate the presence of time-specific patterns during a trading day. For that reason we included dummies that indicate the specific time of each trade (minutes, hours, days). That was also suggested in [49], and was found to aid the predictive power of models.

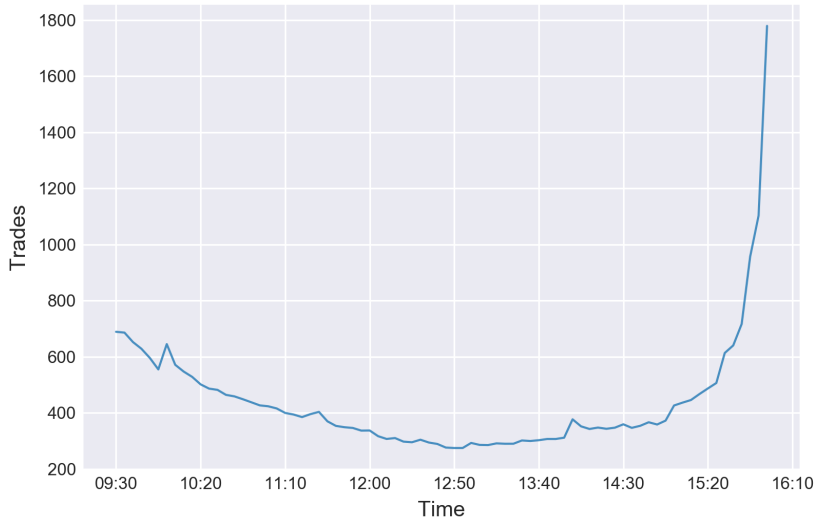

FIGURE 3.5: Trading Intensity

Average number of trades during the day as measured by taking into account all the 44 stocks at our disposal. A clear trend can be observed throughout the day, with trade intensity decreasing until 13:00 and then increasing exponential when approaching close. The irregularity at 10:00 happens due to the "10 o'clock rule", according to which, most traders avoid the first 30 minutes of the day due to high volatile price movements.

FIGURE 3.6: Price Movements

Class buy probabilities through-out the day, as measured by taking into account all the 44 stocks of this research. Although really volatile, by observing the moving average through the day, we can identify a small decreasing trend until 13:00, which is followed by an increasing one.

Chapter 4

Methods

4.1 Recurrent Neural Networks

RNNs have been effectively used to model sequential data, where distributions are not stationary. In sequential data, the probability of a certain event depends on the probabilities of past events.

$$\mathcal{P}(x_t) = \Pi_{i=1}^{t} \mathcal{P}(x_{t-i}|x_{t-i-1}, ..., x_1) \tag{4.1}$$

A popular application of RNNs is in the field of Natural Language Processing (NLP). Given the sentence "*The weather is cloudy today, it is probably going to ...*", in order for a model to be able to predict the last word, it needs to remember the occurrence of past words, as well as their respective order. Thus, the need of memory is apparent.

RNNs have the ability to remember past observations by maintaining a cell state c, which is constantly updated. At every time-step t, when input x_t is fed to the network, it is used to update the state from c_{t-1} to c_t. Then the output o_t is calculate using the new state c_t.

$$c^{(t)} = f(\mathbf{U}x^{(t)} + \mathbf{W}c^{(t-1)} + \mathbf{b}_c) \tag{4.2}$$

$$o^{(t)} = g(\mathbf{V}c^{(t)} + \mathbf{b}_o) \tag{4.3}$$

Where $\mathbf{U}$ is the input weight matrix, $\mathbf{W}$ is the recurrent weight matrix, $\mathbf{V}$ is the output weight matrix and $\mathbf{b}_c, \mathbf{b}_o$ are two bias vectors. Usual choices for f and g, in classification tasks, are the Hyperbolic Tangent Sigmoid (tanh) and Softmax functions accordingly.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4.4}$$

$$softmax(x_k) = \frac{e^{x_k}}{\sum_{i=1}^{m} e^{x_i}} \qquad , k \in [1, m] \tag{4.5}$$

Both of them are heavily used in deep learning. The *tanh* function is zero centered, with strong gradients around that point A.1, which ensures stable and robust training. Since $\sum_{i=1}^{m} softmax(x_i) = 1$, the softmax function is used to transform input x to a probability distribution of m classes.

An RNN is basically a feed-forward network that loops over itself. It can also be presented in an unfolded way for better visualization. These two version are presented in Figure 4.1, where c is the cell state, x is the input, o is the output, U is the input weight matrix, W is the recurrent weight matrix and V is the output weight matrix.

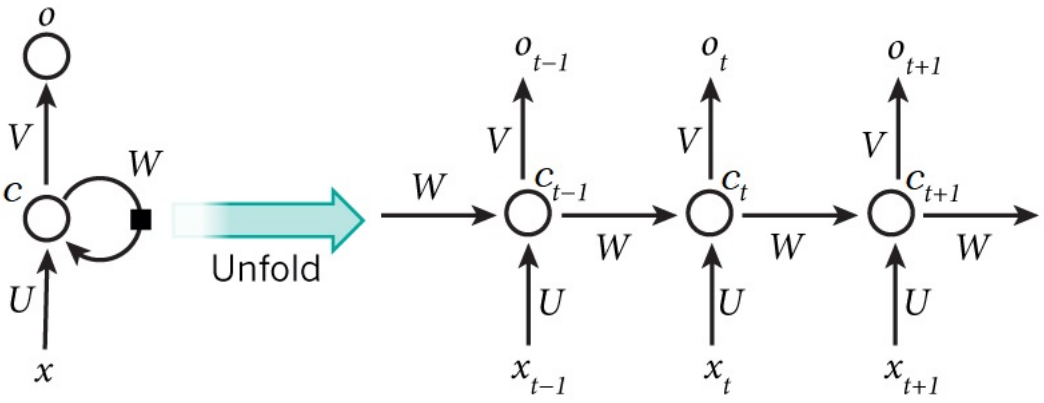

FIGURE 4.1: A Recurrent Neural Network

Left: RNN folded, Right: RNN unfolded. Biases are omitted for ease of representation. Source: http://www.wildml.com

Backpropagation Through Time is used for training RNNs. It works in the same way as regular Backpropagation, using the chain rule to estimate error signals, but also along the time dimension. The problem arises from the fact that the error signals coming from past observations become weaker and weaker, and thus long term-dependencies cannot be modeled efficiently. The primary reason for the weak signals is the derivative of the cell states with respect to previous cell states. If $|\frac{\partial c_t}{\partial c_{t-1}}| < 1$, for consecutive states, the derivatives of the long-term states become really small 4.6 This difficulty can be overcome by using LSTM networks.

$$\frac{\partial c_t}{\partial c_{t-j}} = \frac{\partial c_t}{\partial c_{t-1}} \cdot \frac{\partial c_{t-1}}{\partial c_{t-2}} \cdot \ldots \cdot \frac{\partial c_{t-j+1}}{\partial c_{t-j}} \approx 0 \tag{4.6}$$

4.2 Long Short-term Memory Networks

The LSTM is an updated version of the vanilla RNN. The first difference between the two networks is that the cell states in the LSTM are connected in a linear way. That is how the vanishing gradient problem is solved, since the derivative now is just the weight $\boldsymbol{W}$, which connects previous and current states. The second difference is the use of multiple sigmoid (σ) gates that control the information flow through the model.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.7}$$

The mathematical configuration of the LSTM is given in the following equations.

$$g^{(t)} = tanh(\mathbf{W}_{gx}\mathbf{x}^{(t)} + \mathbf{W}_{gh}\mathbf{h}^{(t-1)} + \mathbf{b}_g) \tag{4.8}$$

$$i^{(t)} = \sigma(\mathbf{W}_{ix}\mathbf{x}^{(t)} + \mathbf{W}_{ih}\mathbf{h}^{(t-1)} + \mathbf{b}_i) \tag{4.9}$$

$$f^{(t)} = \sigma(\mathbf{W}_{fx}\mathbf{x}^{(t)} + \mathbf{W}_{fh}\mathbf{h}^{(t-1)} + \mathbf{b}_f) \tag{4.10}$$

$$o^{(t)} = \sigma(\mathbf{W}_{ox}\mathbf{x}^{(t)} + \mathbf{W}_{oh}\mathbf{h}^{(t-1)} + \mathbf{b}_o) \tag{4.11}$$

$$c^{(t)} = g^{(t)} \odot i^{(t)} + c^{(t-1)} \odot f^{(t)} \tag{4.12}$$

$$h^{(t)} = tanh(c^{(t)}) \odot o^{(t)} \tag{4.13}$$

Where $\odot$ denotes the element-wise multiplication operator, $\mathbf{W}_{*x}$ denotes input weight matrices, $\mathbf{W}_{*h}$ denotes recurrent weight matrices and $\mathbf{b}_*$ denotes bias vectors.

The basic structure of an LSTM contains four essential gates, that control the flow of information in every training iteration. These are the input modulation gate $g^{(t)}$ 4.8, input gate $i^{(t)}$ 4.9, forget gate $f^{(t)}$ 4.10 and output gate $o^{(t)}$ 4.11. [1]

[1] a slight change in notation from the RNN, since o denotes the output gate, the actual output of the LSTM is denoted as h

The input modulation gate transforms the previous state and the new input via a *tanh* activation function. It represents the new candidate values that could be added to the cell state. The choice of the non-linearity for this gate stems from the fact that should always add value to the cell state. Thus the output of the *tanh*, which is in $[-1, 1]$, suits perfectly.

The input gate, weights the new candidate values coming from the modulation gate, before being added to the cell state. The sigmoid function, with outputs in $[0, 1]$ will provide weight according to the relevance of the information.

The forget gate serves the cause of filtering the current information contained in the cell state. An output of 0 from the sigmoid function means that the model should forget everything, while an output of 1 means that it should retain this information in its entirety.

Lastly, the output gate is of similar function, as the other two sigmoids. It highlights the information that will be passed to the next hidden state.

After the calculation of the gates, the cell state is updated 4.12 and the output at time-step t is calculated 4.13.

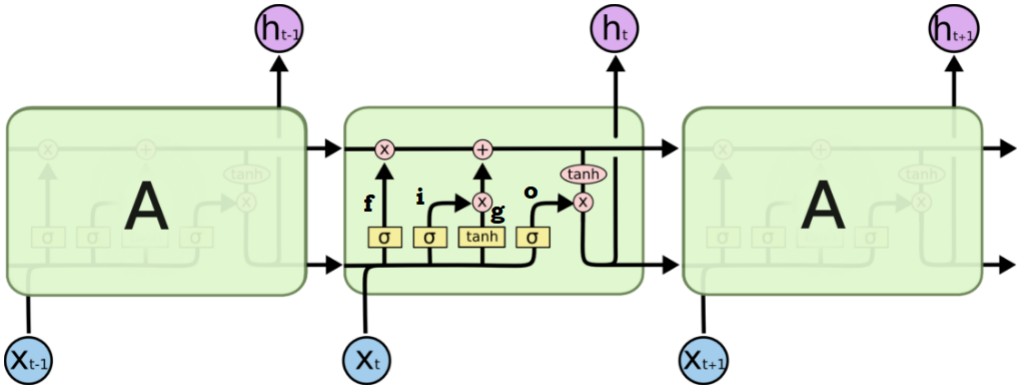

FIGURE 4.2: A Long Short-term Memory network

An unfolded version of the LSTM cell denoting the four gates, inputs and outputs. Source: http://colah.github.io

In the model architecture implemented in this thesis, we used two LSTMs stacked on one another (stacked-LSTM). The two LSTMs are of different size, with the first one being larger than the one the one that follows it. This is a common practice, aiming to identify different features in terms of specificity. The first layer is able to recognize more general features, while the second one targets at more specific ones. For example, in the computer vision task of animal classification, first layers detect features like edges and lines, while the latter ones detect more specific ones, like eyes or ears. In our

implementation, the respective sizes of the two LSTMs were set after tuning to 64 and 32.

Regarding the maximum memory of the LSTMs, which is usually called sequence length, we set it at 5. This decision is part of several trial-and-error experiments that were conducted. The gains from using longer sequences, were not justified by the extra computational cost that comes with them. It also ensured, that long-term information is fed into the model, by the use of technical indicators computed over various time-windows. Thus, the use of large sequence lengths is rendered obsolete.

4.3 Layer Normalization

While training many-layered networks, it is important to maintain the inputs of the activation functions centered around zero. Due to the nature of the non-linearities *tanh* and σ, activating at extreme values, cause really weak gradients during Backpropagation, which in turn will cause certain neurons to saturate [2]. The behavior of the non-linearities, along with their derivatives can be observed in Figures A.1 A.2. Normalizing the data before feeding it to the model is not enough, as after passing through many layers and can diverge from a standard normal distribution. To counter a potential shift in the distribution of the inputs of every layer, a number of techniques have been developed, with Batch Normalization [51] being the most widely used.

According to Batch Normalization, before applying the activation functions at every layer, its inputs are normalized according to their mean and standard deviation, across the batch size dimension d_1. Batch Normalization has been found to speed up training, while also acting as a regularizer 4.5. Although efficient, it can not be properly applied to recurrent networks, since the appearance of the extra sequence length dimension d_2 complicates the calculation of normalization statistics.

Layer Normalization was proposed in Ba et al. 2016 [52], as a variant of Batch Normalization that can be applied to recurrent networks. According to Layer Normalization, moments are calculated across the sequence length d_2 dimension instead. Layer Normalization can be defined as:

$$LayerNorm(\mathbf{Z}; \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{\mathbf{Z} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \odot \boldsymbol{\alpha} + \boldsymbol{\beta} \tag{4.14}$$

[2] The result of extreme compression of the input between bounds [-1, 1] for *tanh* and [0,1] for σ is small derivatives, A saturated neuron, passes a really weak error signal and does not contribute to training.

Where $\boldsymbol{\alpha}$ (gain) and $\boldsymbol{\beta}$ (shift) are trainable parameters that are updated according, to fit the distribution of the output, while $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are the mean and standard deviation, across the dimensionality d_2 of input $\mathbf{Z}$.

$$\boldsymbol{\mu} = \frac{1}{d_2}\sum_{i=1}^{d_2} \boldsymbol{z_i} \tag{4.15}$$

$$\boldsymbol{\sigma} = \sqrt{\frac{1}{d_2}\sum_{i=1}^{d_2}(\boldsymbol{z_i} - \boldsymbol{\mu})} \tag{4.16}$$

With the inclusion of layer normalization, equations 4.8, 4.9, 4.10, 4.11, 4.13 become:

$$g^{(t)} = tanh(LayerNorm(\mathbf{W}_{gx}\mathbf{x}^{(t)}; \boldsymbol{\alpha}_g, \boldsymbol{\beta}_g) + LayerNorm(\mathbf{W}_{gh}\mathbf{h}^{(t-1)}; \boldsymbol{\alpha}_g, \boldsymbol{\beta}_g)) \tag{4.17}$$

$$i^{(t)} = \sigma(LayerNorm(\mathbf{W}_{ix}\mathbf{x}^{(t)}; \boldsymbol{\alpha}_i, \boldsymbol{\beta}_i) + LayerNorm(\mathbf{W}_{ih}\mathbf{h}^{(t-1)}; \boldsymbol{\alpha}_i, \boldsymbol{\beta}_i)) \tag{4.18}$$

$$f^{(t)} = \sigma(LayerNorm(\mathbf{W}_{fx}\mathbf{x}^{(t)}; \boldsymbol{\alpha}_f, \boldsymbol{\beta}_f) + LayerNorm(\mathbf{W}_{fh}\mathbf{h}^{(t-1)}; \boldsymbol{\alpha}_f, \boldsymbol{\beta}_f)) \tag{4.19}$$

$$o^{(t)} = \sigma(LayerNorm(\mathbf{W}_{ox}\mathbf{x}^{(t)}; \boldsymbol{\alpha}_o, \boldsymbol{\beta}_o) + LayerNorm(\mathbf{W}_{oh}\mathbf{h}^{(t-1)}; \boldsymbol{\alpha}_o, \boldsymbol{\beta}_o)) \tag{4.20}$$

$$c^{(t)} = g^{(t)} \odot i^{(t)} + c^{(t-1)} \odot f^{(t)} \tag{4.21}$$

$$h^{(t)} = tanh(LayerNorm(c^{(t)}; \boldsymbol{\alpha}_c, \boldsymbol{\beta}_c)) \odot o^{(t)} \tag{4.22}$$

Note that equation 4.12 is unchanged, but was included for convenience. Although not mentioned in [52], bias vectors $\boldsymbol{b}$ become redundant with the introduction of Layer Normalization, since $\boldsymbol{\beta}$ parameters replace their function. Thus, the bias vectors are excluded from equations 4.17, 4.18, 4.19, 4.20. Lastly, no alteration is made to the softmax layer, as being the last layer, ensures strong error signals.

4.4 Cross-Entropy loss

The outputs of the stacked LSTM cell are passed through a softmax layer that transforms them into probabilities for class "Sell" and "Buy". To account for the issue of class imbalance that was mentioned in Section 3.4, when estimating the loss, $\boldsymbol{h^{(t)}}$ is weighted by w_{class}, which is estimated from the ratio of positive and negative examples in the training set. Note that the weighting of the stacked-LSTM's output takes only place during training and not in inference. Finally, the cross entropy loss function is used to measure the error of the probabilities $\boldsymbol{p_{weighted}}^{(t)} = softmax(w_{class}\ \boldsymbol{h}^{(t)})$ with respect to the one-hot representation of the true value $\mathbf{y}^{(t)}$. The cross entropy loss for one instance t between $\boldsymbol{p}^{(t)}_{weighted}$ and $\mathbf{y}^{(t)}$ is defined as:

$$\mathcal{L}(\boldsymbol{y}^{(t)}, \boldsymbol{p}^{(t)}_{weighted}) = -\sum_{k\in[Sell,Buy]} y_k^{(t)} log(p^{(t)}_{k,weighted}) \tag{4.23}$$

Where $y^{(t)}_{Buy} = 1$ and $y^{(t)}_{Sell} = 0$, if true class is "Buy". Basically, for every instance, the loss is the negative log probability of the "wrong" class.

Inputs of the model, are in the form of a three-dimensional tensor, with dimension d_1 corresponding to batch size[3], d_2 to sequence length and d_3 to the number of features. The first two dimensions remain the same as input travels through the model. The last one (d_3) changes according to the size of each layer and finally, after the softmax layer, becomes two (number of classes).

Equation 4.23 can be generalized, to account for the general loss in each training iteration, averaging it over the batch size and sequence length dimensions.

$$\mathcal{L}(\boldsymbol{Y}, \boldsymbol{P_{weighted}}) = -\frac{1}{d_1 d_2}\sum_{i=1}^{d_1}\sum_{t=1}^{d_2}\sum_{k\in[Sell,Buy]} y_k^{(i,t)} log(p^{(i,t)}_{k,weighted}) \tag{4.24}$$

4.5 Regularization

One of the most frequent issues, happening when training deep or recurrent networks, is overfitting. Overfitting the data means that the network learns a function that maps the training inputs to the outputs, in such a good way, that is not generalizable. Thus, even though performance on the train set is really high when the network is introduced to new

[3] B :number of instances from the training set that is used in every iteration. 32 is used for this research, more in Section 4.9.1

data, it performs poorly. Two measures are taken in this thesis to counter overfitting and make the trained models as generalizable as possible.

4.5.1 Weight decay

Weight decay aims to penalize weight-complexity and prevent certain weights to be used more than others, by modifying the loss function. The most popular weight decay method is the $\ell 2$ regularization, which uses the Euclidean norm of the weights as an extra term in the loss functions. A parameter λ is chosen to control the magnitude of the weight decay.

$$\mathcal{L}_{reg} = \lambda \sum_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} ||\boldsymbol{\theta}||_2 \tag{4.25}$$

Where $\boldsymbol{\Theta}$ denotes the collection of parameter matrices in the model, except biases [4] . Namely, the regularized parameters are $\boldsymbol{W}_{softmax}$, $\boldsymbol{W}_{ij}$ ($i \in [g, i, f, o]$ and $j \in [x, h]$) and $\boldsymbol{\alpha_i}$ ($i \in [g, i, g, o, c]$). Regarding the regularization of the Layer Normalization parameters, no clear answer exists in the literature. We decided to regularize the gain parameters $\boldsymbol{\alpha}$, as they are of multiplicative nature. Shift parameters $\boldsymbol{\beta}$ are not regularized since they act like biases. An overview of the parameter dimensions and regularization is provided in Table 4.1.

The averaged regularization loss is added to the cross-entropy loss function and the collective loss of the model is defined as followed.

$$\mathcal{L}(\boldsymbol{Y}, \boldsymbol{P}_{weighted}) = -\frac{1}{d_1 d_2} \sum_{i=1}^{d_1} \sum_{t=1}^{d_2} \sum_{k \in [Sell, Buy]} y_k^{(i,t)} log(p_{k,weighted}^{(i,t)}) + \frac{\lambda}{d_1 d_2} \sum_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} ||\boldsymbol{\theta}||_2 \tag{4.26}$$

Via trial-and-error, the appropriate parameter for the weight decay was found to be 0.1.

4.5.2 Dropout

Dropout, introduced in Srivastava et al. 2014 [53], is another technique that counters overfitting and enhances model generalizability. According to dropout, in each training iteration, every neuron has a probability p to be turned off. This way, the architecture of the network is different in every iteration, preventing it from overvaluing certain

[4] Since biases are additive terms, they do not contribute significantly to the complexity of the model, unlike the weight terms, which are multiplicative. Thus, biases are usually not regularized.

activations, restricting co-dependencies between neurons, while also improving training speed.

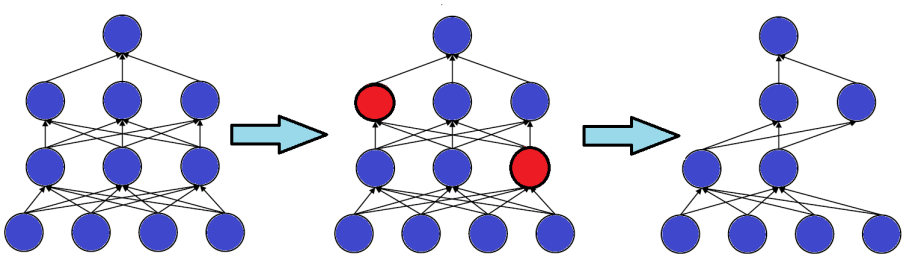

FIGURE 4.3: Dropout

Demonstration of dropout applied to hidden layers. Left: Original network, Middle: Neurons chosen to be turned-off, Right: New network for this training iteration

It is worth mentioning that dropout affects the network only during training. When applying the trained model, dropout is deactivated, and the full network is used for predictions. Although dropout may be applied to recurrent layers as well [54], we found that its use on top of weight decay was restricting significantly the memory capacity of the model. Thus, dropout is used only to non-recurrent layers. More specifically a dropout layer is introduced before inputs go through the stacked LSTMs, and another one between the stacked-LSTM and the softmax layer.

Since a large number of features is used, we applied an aggressive dropout rate to prevent overfitting. For the input dropout, we used a different rate for stock specific features [5] . The reason for biasing the procedure towards stock specific features, is that they contain most of the relative information, in contrast to competitor or sector features. Only about 40% ($\frac{97}{97+160}$) of the features used in training are coming from the stock specific dataset. Thus, by applying different dropout rates, we ensure that the ratio of active features per training iteration is approximately 50%-50% for stock related and the non-stock related. The rates used for the input dropout are 50% and 70%, while for the output is 50%.

4.6 Input Construction

As mentioned in Chapter 3, we engineer a great number of features. Using all of them in training our models, would cause a significant decrease in training time and would also hurt the generalizability of the models. After all, our predictive framework is quantity-based, meaning that we aim to produce many simpler models, that will collectively

[5] These are features coming directly from the stock dataset. The rest are coming from the competitor and sector datasets.

provide the best possible predictions. For the ensemble to be effective, we need to boost the variability of the produced models. To achieve that, for each of the 12 models trained, to apply predictions for time period T, we chose a different subset $D_i, i \in [1, 12]$ of the training set $\mathcal{D}^{(T)}$ as input. Subsets D_i differ between in both the number of instances and features.

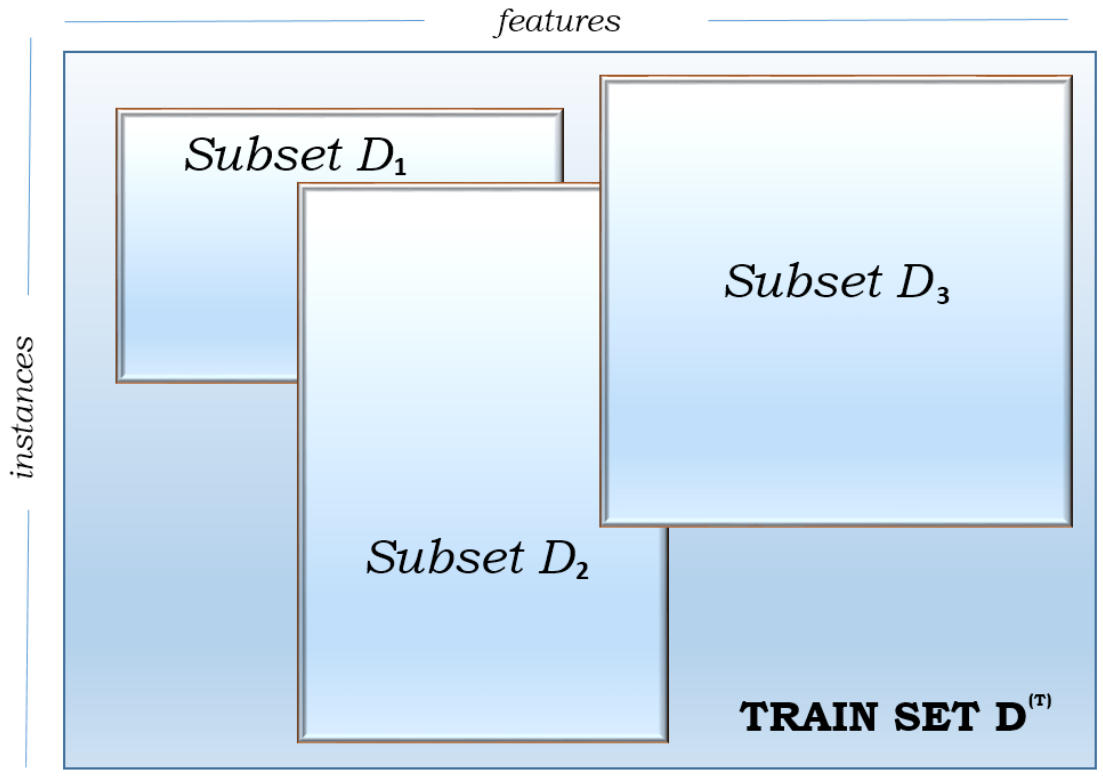

FIGURE 4.4: Randomized input selection

Subsets D_i differ with each other in both dimensions of the Train set $\mathcal{D}^{(T)}$. Note: Instances are always ordered, while features are not.

4.6.1 Instances selection

The length of the train set is 4 weeks, approximately 1560 instances (L). To randomize the period size of the train set used for each $model_i, i \in [0, 12]$, we select periods of 8, 10 and 12 days, which translates to 624, 780 and 936 observations in length (l_i). Four models are build for every period. A starting point is chosen at random between $L - \frac{10}{11}l_i$ and the ending point is the length of the period (l_i) plus a random integer in range $(-\frac{1}{10}l_i, \frac{10}{11}l_i)$.

4.6.2 Feature selection

Unfortunately, unlike other machine learning algorithms, the nature of the LSTM does not allow for a fast feature selection method, based on some information gain criteria. Thus, feature selection is also randomized, and biased towards stock-specific features. For training $model_i$, in time-period T, for some stock s, every time we include all the features from the stock s dataset, and randomize the selection from the rest of the datasets. We have at our disposal 4 datasets categories from which features can be selected. These are the main competitor dataset, highly correlated stocks to stock s dataset, sector S of stock s dataset and the universe dataset. Each dataset is paired to a time-window, based on which features were engineered [6]. Thus, every dataset gets assigned to features from certain time-frames. By denoting with r a vector containing the 4 time-frames in random order, we can summarize the procedure as:

- Select all stock-specific features
- Select features from main competitor dataset on time-frame r_1
- Select features from sector dataset on time-frame r_2
- Select features from universe dataset on time-frame r_3
- Pick the 3 stocks with the highest cross-correlation between them and stock s and select from their respective datasets features on time-frame r_4
- As mentioned in Section 3.5, included are also dummies indicating clock time and day of each instance (minute, hour and day)

4.7 Implemented Architecture

Here we provide a summary of the implemented model architecture. As mentioned in Section 4.5.2, there two different streams of data entering the model, the stock related and the non-stock related. Two different dropout rates are applied and then the two streams are concentrated along d_3 dimension. The concentrated tensor flows through the stacked LSTM and its outputs are weighted according to the class weights derived from the train set. Another dropout layer is applied before the data is transformed into probabilities via the softmax layer. Finally, the loss for this training iteration is calculated as the sum of the cross-entropy loss and the regularization loss, averaged along dimensions d_1 and d_2.

[6] technical indicators, rolling regressions and probability distributions were produced using look-back windows of 6, 12, 36, or 78 instances; we excluded the long-term window of 236

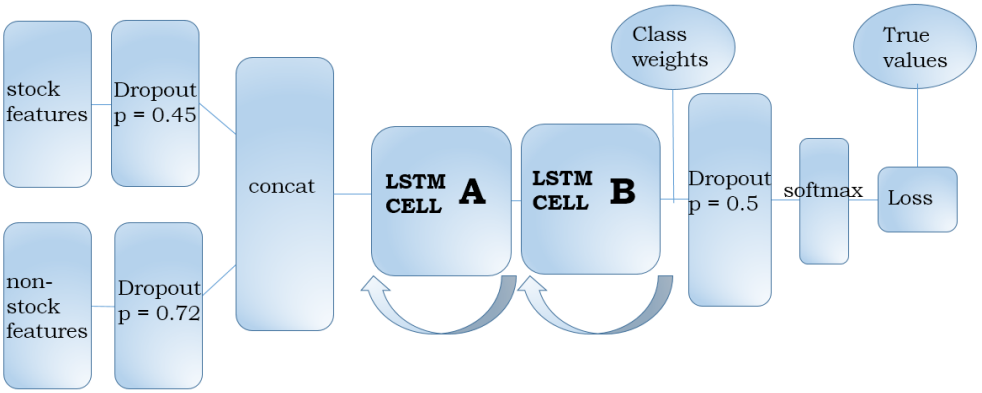

FIGURE 4.5: Model architecture (training)

A slightly different architecture takes place when performing inference, as dropout and class weights are deactivated.

4.8 Parameter initialization

The initialization of the model parameters $\boldsymbol{\theta}$ defines the starting point at the loss surface and can play a crucial role in the optimization process. Ill-conditioned initialization may slow down training by adding too much noise, or even prevent the algorithm from convergence. The concept that drives weight initialization is that they need to be initialized in way, that ensures strong gradients in the first training iterations and thus preventing dead neurons.

The derivatives of the activation functions $tanh$ and $sigmoid(\sigma)$ must be taken into consideration.

$$\frac{\partial tanh(x)}{\partial x} = 1 - tanh^2(x) \tag{4.27}$$

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x)) \tag{4.28}$$

From equations 4.4, 4.27 and 4.7, 4.28 we can derive that in order to have strong gradients, the inputs of x of $tanh(x)$ and $\sigma(x)$ need to be centered around zero. This can also be observed in the graphical representation of the two functions in Figures A.1, A.2. Usually, since data is also normalized, initial weights are sampled from a normal distribution $\mathcal{N}$ with zero mean and small variance. A common practice is to use the Glorot initialization [55]. The idea, behind the Glorot initialization, is to sample the weights from a distribution that will ensure preserving the variance of the signal through each layer.

Given that incoming data $\boldsymbol{X}$ is normalized and independent from $\boldsymbol{W}$, the variance of the output is equal to $n_{l,in} \cdot \ Var(\boldsymbol{W})$, with $n_{l,in}$ being the number of incoming neurons to the layer l. Thus, in order to preserve the variance during the forward pass, weights should be initialized with zero mean and $\frac{1}{n_{l,in}}$ variance. The same applies to preserving the variance through the backward pass (Backpropagation). Thus, in order to maintain the variance through both passes, weights should be initialized with variance equal to two times the reciprocal of the sum of the incoming and outgoing neurons of each layer l.

$$Var(\boldsymbol{W}_l) = \frac{2}{n_{l,in} + n_{l,out}} \tag{4.29}$$

Using Layer Normalization in the LSTMs, further ensures that inputs to activations are centered around zero. The parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ of the Layer Normalization are initialized, with the hypothesis that outputs are standard normally distributed. Thus $\boldsymbol{\alpha}$ parameters are initialized to 1 and $\boldsymbol{\beta}$ parameters to 0.

Regarding the softmax layer, its performance is not sensitive to the initialization of its weights. The softmax layer is placed right before the loss function and thus the error signals are really strong. Furthermore, softmax is strictly convex [56] and thus easy to optimize. Given the above, the initial softmax weights are sampled by a normal distribution with zero mean and 0.1 variance.

Lastly, since most of the variability in the model comes from the multiplicative terms (weights), biases are usually just initialized to zeros.

Parameter	**Dimension**	**Initialization**	**Regularized**
$\boldsymbol{W}_{gx}^{(A)}$, $\boldsymbol{W}_{ix}^{(A)}$, $\boldsymbol{W}_{fx}^{(A)}$, $\boldsymbol{W}_{ox}^{(A)}$	250×64	$\mathcal{N}(0, \sqrt{\frac{2}{250+64}})$	True
$\boldsymbol{W}_{gh}^{(A)}$, $\boldsymbol{W}_{ih}^{(A)}$, $\boldsymbol{W}_{fh}^{(A)}$, $\boldsymbol{W}_{oh}^{(A)}$	64×64	$\mathcal{N}(0, \sqrt{\frac{2}{64+64}})$	True
$\boldsymbol{W}_{gx}^{(B)}$, $\boldsymbol{W}_{ix}^{(B)}$, $\boldsymbol{W}_{fx}^{(B)}$, $\boldsymbol{W}_{ox}^{(B)}$	64×32	$\mathcal{N}(0, \sqrt{\frac{2}{64+32}})$	True
$\boldsymbol{W}_{gh}^{(B)}$, $\boldsymbol{W}_{ih}^{(B)}$, $\boldsymbol{W}_{fh}^{(B)}$, $\boldsymbol{W}_{oh}^{(B)}$	32×32	$\mathcal{N}(0, \sqrt{\frac{2}{32+32}})$	True
$\boldsymbol{\alpha}_{g}^{(A)}$, $\boldsymbol{\alpha}_{i}^{(A)}$, $\boldsymbol{\alpha}_{f}^{(A)}$, $\boldsymbol{\alpha}_{o}^{(A)}$, $\boldsymbol{\alpha}_{c}^{(A)}$	64×1	$\mathbf{1}$	True
$\boldsymbol{\beta}_{g}^{(A)}$, $\boldsymbol{\beta}_{i}^{(A)}$, $\boldsymbol{\beta}_{f}^{(A)}$, $\boldsymbol{\beta}_{o}^{(A)}$, $\boldsymbol{\beta}_{c}^{(A)}$	64×1	$\mathbf{0}$	False
$\boldsymbol{\alpha}_{g}^{(B)}$, $\boldsymbol{\alpha}_{i}^{(B)}$, $\boldsymbol{\alpha}_{f}^{(B)}$, $\boldsymbol{\alpha}_{o}^{(B)}$, $\boldsymbol{\alpha}_{c}^{(B)}$	32×1	$\mathbf{1}$	True
$\boldsymbol{\beta}_{g}^{(B)}$, $\boldsymbol{\beta}_{i}^{(B)}$, $\boldsymbol{\beta}_{f}^{(B)}$, $\boldsymbol{\beta}_{o}^{(B)}$, $\boldsymbol{\beta}_{c}^{(B)}$	32×1	$\mathbf{0}$	False
$\boldsymbol{W}_{softmax}$	32×2	$\mathcal{N}(0, 0.01)$	True
$\boldsymbol{b}_{softmax}$	1×2	$\mathbf{0}$	False
Trainable Parameters = 93,698			

TABLE 4.1: Parameter Size & Initialization

All trainable parameters, along with their dimensions, initialization method and regularization. (A) indicates the first LSTM and (B) the second one. We assume that the number of features is 250, while it may slightly vary from model to model since we randomized the procedure.

4.9 Training the model

4.9.1 Stochastic Gradient Descent

Neural networks are trained using gradient descent methods. Gradients are computed for every trainable parameter θ by the derivative of the loss function $\mathcal{L}$, 4.24 with respect to θ. Then the parameters are updated using a magnitude (learning rate) η. Classic gradient descent is defined in 4.30.

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \boldsymbol{g}^{(t)} \tag{4.30}$$

$$\boldsymbol{g}^{(t)} = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{\partial \mathcal{L}(f(x_i; \boldsymbol{\theta}), y_i)}{\partial \boldsymbol{\theta}} \tag{4.31}$$

Where D denotes the training set, $\boldsymbol{g}$, is the average gradient computed over training set D and $f(\cdot)$ is the neural network function.

Using the average gradient of the training set can be computationally expensive and may cause large steps on the loss surface, missing the global minima. Stochastic Gradient

Descent (SGD) is a variant of gradient descent, in which, instead of the true gradient, the expected gradient is used. True gradient means that the gradient is computed using the entire dataset, while the expectation of the gradient is computed using a sample (mini-batch) of the training set. Thus, we can define the expected gradient for a mini-batch B as:

$$E[\boldsymbol{g}^{(t)}] = \frac{1}{|B|} \sum_{i=1}^{|B|} \frac{\partial \mathcal{L}(f(x_i; \boldsymbol{\theta}), y_i)}{\partial \boldsymbol{\theta}} \qquad , B \subset D \tag{4.32}$$

Using the expected gradient has been shown to be not only computationally efficient, but also faster in terms of convergence. For simplicity, from now on, using the term "gradient" we mean the gradient computed over a mini-batch B (denoted as d_1 in previous sections).

For the purpose of this thesis, all models are trained using a mini-batch of 32, which is approximately, $\frac{1}{50}$ of the training set size. This is also suggested in Bengio 2012 [57], as it is usually large enough to take advantage of the computational speed-up of matrix multiplications.

Although SGD is straight-forward and efficient, large networks with thousands of parameters are not easy to train. The reason is that the loss surface is usually non-convex and lies in a very high dimensional space.

4.9.2 Momentum and Adaptive Learning methods

Using momentum [58] ensures smoother training by averaging out the individual gradients of every time-step t. Momentum methods use the velocity of the gradient instead of its actual position. Thus, given the momentum parameter m, the update rule becomes:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - m\boldsymbol{g}^{(t-1)} + \eta \boldsymbol{g}^{(t)} \tag{4.33}$$

Adaptive learning methods are also widely used when training large networks. The idea behind adaptive learning is that every trainable parameter should have its own learning rate. It is natural that parameters in different layers may require more or less steep updates than other ones. Thus, every parameter starts with an initial learning η, that evolves according to the optimization process. Hinton suggests in [59] a method for evolving the individual learning rates. If the sign of the gradient for parameter θ_0 is the same for two consecutive training iterations, the gradient g_0 gets boosted by a small addition, otherwise, its gets a small multiplicative decrease. This method ensures that

if the gradient starts bouncing in the loss surface (opposite signs), the learning rate will decrease fast, and the algorithm will start converging.

4.9.3 Root Mean Square Propagation (RMSProp)

The training algorithm used in this research is the RMSProp [59] with momentum. RMSProp works with adaptive learning, by maintaining a moving average of the square of the gradient for each parameter, which is used to normalize the gradients. For a decay rate d, the update rule for the Moving Average of the Squared Gradient (MASG) is defined as:

$$\mathbf{MASG}^{(t)} = d\,\mathbf{MASG}^{(t-1)} + (1-d)(\boldsymbol{g}^{(t)})^2 \tag{4.34}$$

Now using also a momentum of m and learning rate η, we can define the update rule for parameters $\boldsymbol{\theta}$ as:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - m\boldsymbol{g}^{(t-1)} + \eta\frac{\boldsymbol{g}^{(t)}}{\sqrt{\mathbf{MASG}^{(t)} + \epsilon}} \tag{4.35}$$

Where ϵ is a small positive value to avoid division by zero, set to 10^{-8}. The rest of the parameters of the RMSProp were set by means of trial-and-error. We used an initial learning rate of $\eta = 0.001$, a decay rate $d = 0.9$, a momentum of $m = 0.9$.

4.9.4 Gradient Clipping

As discussed in Section 4.2, although the LSTM solves the problem of the vanishing gradient, it does not address the issue of the exploding gradient. The problem of the exploding gradient arises in the back-propagation through time, when sequential error signals from the state derivatives are much greater than 1. Accordingly, their product creates a large gradient that can potentially lead to unstable training. The solution for this problem is to define a threshold g_{max}, above which, all gradients are reduced to it.

$$g^{(t)} = \begin{cases} g_{max} & , g^{(t)} > g_{max} \\ g^{(t)} & , g^{(t)} g_{max} \end{cases}$$

Although, by observing the training process, we found that gradients were rarely greater than 1, we set a max gradient $g_{max} = 5$, to ensure stable training.

4.10 Performance evaluation

4.10.1 The Area Under the Curve (AUC)

Evaluation of the performance of each model is done by the Area Under the Curve (AUC) score. AUC is calculated, as the area defined by the Receiver Operating Characteristic (ROC). The ROC curve is obtained by evaluating the True Positive Rate (TPR) and False Positive Rate (FPR) at different decision thresholds.

In a binary classification scheme, the TPR (or sensitivity) 4.36 is calculated as the fraction of true positives to ground truth positives. Accordingly, FPR (or fall-out) 4.37 is calculated as the fraction of false positives to ground truth negatives. These can be visualized in the confusion matrix in Figure 4.6, given a threshold $t = t_0$.

$$TPR_{t_0} = \frac{TP_{t_0}}{TP_{t_0} + FN_{t_0}} \tag{4.36}$$

$$FPR_{t_0} = \frac{FP_{t_0}}{FP_{t_0} + TN_{t_0}} \tag{4.37}$$

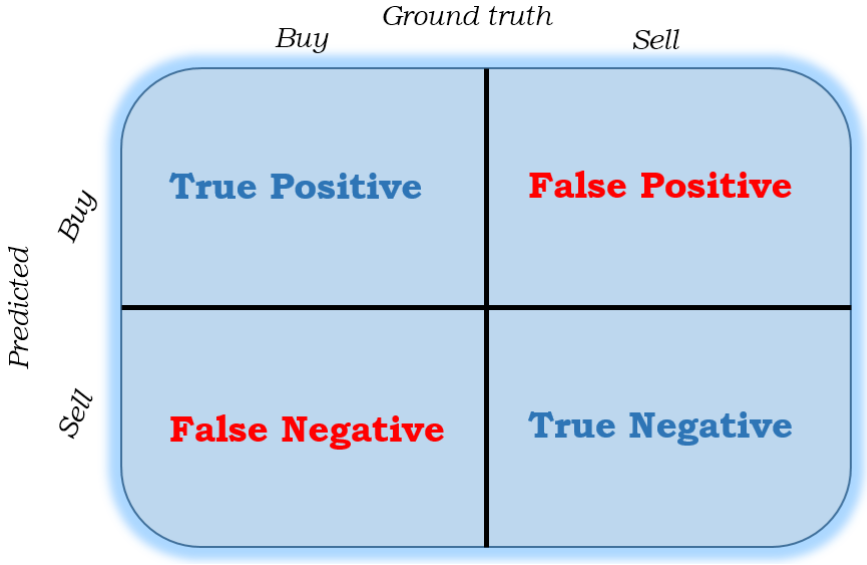

FIGURE 4.6: Confusion Matrix

Using the probabilities that were output by the softmax layer in our model, and given a threshold parameter t, we can define TPR and FPR as followed.

$$TPR(t) = \int_t^1 f_1(x)dx \tag{4.38}$$

$$FPR(t) = \int_t^1 f_0(x)dx \tag{4.39}$$

Where $f_1(x)$ and $f_0(x)$ are the probability density functions of x if it belongs to class 'Buy', or 'Sell' accordingly. The density functions and their intersections, as defined by the varying threshold t, can be observed in the example of Figure 4.7.

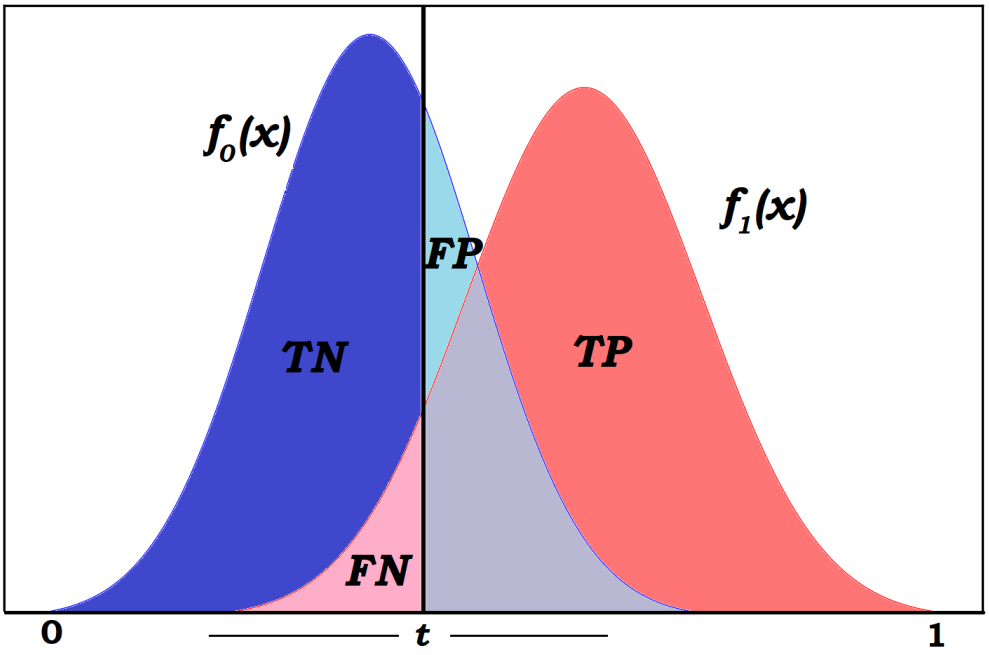

FIGURE 4.7: False Positive Rate and True Positive Rate

By varying threshold t over the range $[0, 1]$, the FPR is the ratio of the area covered by FP to the area defined by $f_0(x)$. On the other hand, the TPR is the ratio of the area covered by TP to the area defined by $f_1(x)$. Note that the grey area is part of both FP and TP. Also, it has to be noted that this example and not something we derived from our methods. In our case, given its difficulty, the density functions are most probably blended with one another and are not clearly separable, like those in this example.

By plotting the TPR against FPR, for the varying decision threshold t, the ROC curve is obtained and the AUC score is the area defined by this curve. This translates to following integration.

$$AUC = \int_0^1 TPR(t)(-FPR'(t))dt \tag{4.40}$$

Where the prime indicates the reversal of the integration limits. The AUC score is the probability that a model ranks a random chosen "Buy" example, higher than a random chosen "Sell" example. Being a probability, the AUC score is in the range of $[0, 1]$, with a probability of 0.5 indicating a random model.

4.10.2 Motivation for using the AUC score

AUC is arguably the most used evaluation metric when it comes to classification tasks. One of its advantages over other evaluation metrics is its robustness to class imbalances. Class imbalance, in a classification task, is the problem of one class being the majority of the total population. As previously discussed, in high-frequency market classification, the positive class is systematically lower than 50% of total instances, usually around 46%. Evaluating performance using plain accuracy 4.41 would not make clear what the actual predictive power of the model is, since it assumes that all miss-classification errors are the same. An accuracy score of 54% could just be that the model predicts the majority of instances as "Sell", which obviously does not add any real value, and furthermore cannot be used as a means of model comparison.

$$Accuracy_{t_0} = \frac{TP_{t_0} + TN_{t_0}}{TP_{t_0} + FP_{t_0} + TN_{t_0} + FN_{t_0}} \tag{4.41}$$

Other measures, like Sensitivity 4.36, or Precision 4.42, or their harmonic mean, the f1-score 4.43, may be used in order to tackle class imbalance and provide a better view on the predictive power of a model. Yet again, there are some shortcomings. Firstly, the True Negatives(TN) are left out of the equation. Secondly, these measures still evaluate performance over one decision threshold $t = t_0$.

$$Precision_{t_0} = \frac{TP_{t_0}}{TP_{t_0} + FP_{t_0}} \tag{4.42}$$

$$f1_{t_0} = \frac{1}{\frac{1}{Sensitivity_{t_0}} + \frac{1}{Precision_{t_0}}} \tag{4.43}$$

The AUC score is robust to both class imbalance and decision thresholds, and thus, it is an unbiased performance metric, that can be used to capture at full the performance of our models. Furthermore, in high difficulty classification tasks, like ours, it is a benchmark by itself. The weak form of market efficiency suggests that past market information (like technical indicators) hold no predictability power over future events. According to this statement, predictions of a classifier should approach random performance, meaning an AUC score of 0.5. Thus, any classifier that provides AUC scores greater than 0.5, would be evidence against the EMH.

4.11 Ensemble

The final stage of our predictive framework involves the use of an ensemble algorithm to combine the predictions of 12 LSTMs, for every predictive period. A simple way to implement an ensemble is to take the average of the individual predictions. Thus for stock S and for instance t, the prediction of the ensemble is

$$p_{S,t}^{(ensemble)} = \frac{1}{12}\sum_{i=1}^{12} p_{S,t}^{(model_i)} \tag{4.44}$$

where $p_{S,t}^{(model_i)}$ is the probability for class 'Buy' as was output by the softmax layer of $model_i$. For the rest of the thesis, we are referring to this model, as Equally Weighted Ensemble.

Except for an equally weighted ensemble, we considered two more methods. The idea is to take into account the performance of each model in the previous n observations. Thus, the weight of each model is proportional to its AUC score 4.40 in the past n observations.

$$p_{S,t}^{(ensemble)} = \sum_{i=1}^{12} \frac{AUC_{S,t}^{(model_i)}}{\sum_{k=1}^{12} AUC_{S,t}^{(model_k)}} p_{S,t}^{(model_i)} \tag{4.45}$$

where $AUC_{S,t}^{(model_i)}$ indicates the AUC score of $model_i$ in period $(t-n-1, t-1)$. This type of ensemble is refereed to as Performance Weighted Ensemble for the rest of the thesis.

To identify whether predictability comes from a single good model, or from their collective interactions, we furthermore considered an ensemble, which takes into account only the best model and will be referred to as Best Model Ensemble.

$$p_{S,t}^{(ensemble)} = argmax_{\boldsymbol{AUC}_{S,t}} \ \boldsymbol{p}_{S,t} \tag{4.46}$$

Our intuition is that each one of the 12 models for has identified and learned certain patterns from the training set. Whether these patterns can be found also in the test set, or they have disappeared, we are not in a position to now. But we know that if they emerge, they are going to be preserved for some time t. Thus, the ensemble can work in an online way and learn to update the weighting of the models according to their ability to identify some good patterns in the last n instances.

As a look-back window for validating the models, we chose $n = 390$, which corresponds to one week of data. Given that predictions are applied for 2 weeks, 1 week provides enough time to validate and identify the appropriate weights, while the remaining week can be used for forecasting.

4.12 Benchmarks

To provide a benchmark for the performance of the ensemble, we used two additional methods, the Lasso and the Ridge classifiers. They essential are regularized logistic regressions, with $\ell 1$ weight decay applied for Lasso and $\ell 2$ applied for the Ridge. The selection of these methods for benchmarking, stems from the fact that they are simple (compared to LSTMs) and furthermore they output probabilities that can be used to calculate the AUC scores. The unregularized logistic regression can be defined in the following equation.

$$p(\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{w}\boldsymbol{x}+b}} \tag{4.47}$$

Where $\boldsymbol{x}$ is the feature vector for one instance, $\boldsymbol{w}$ is a vector of weights and b is a bias term. By also including regularization of λ, the function for training the model are the following.

$$\boldsymbol{w}^*, b^* = argmin_{\boldsymbol{w},b} \sum_{i=1}^{n} log(e^{-\boldsymbol{y}_i \boldsymbol{X}_i^T \boldsymbol{w}+b} + 1) + \mathcal{L}_{reg} \tag{4.48}$$

Where n is the number of training examples and $\boldsymbol{y}_i$ is the vector of target values for example i. Finally, the regularization loss is $\mathcal{L}_{reg} = \lambda ||\boldsymbol{w}||_1$ for Lasso and $\mathcal{L}_{reg} = \lambda ||\boldsymbol{w}||_2$ for Ridge.

4.13 Implementation

All required data processing was done in Python 3, using *NumPy* and *pandas*. *NumPy* is a package containing high-level mathematical functions that can operate on multi-dimensional arrays, while *pandas* is build on top of it and provides several tools for data analysis and visualization. Setting up, as well as training the LSTM models was done in *TensorFlow*. *TensorFlow* is a low-level, symbolic math and deep learning library, developed by Google, that provides excellent functionality for training and monitoring

neural networks. It essentially works by creating a computational graph for each of the models, allowing for efficient automatic differentiation during training. Finally, the Lasso and Ridge logistic regression models were trained using the *scikit-learn* package, which provides high-level machine learning functions. Other minor packages used in this research are *matplotlib* and *seaborn* for plotting figures and *pandas-market-calendars* for creating customized NYSE and NASDAQ calendars.

The hardware used for the realization of this research was a personal machine with a 2-core 2.3GHz CPU and 8GB RAM.

Chapter 5

Results

5.1 LSTM ensembles

Each model is trained for a total of 15 epochs. One epoch is equal to the number of training iterations need for the algorithm to hypothetically [1] run over all the instances of the trains set. Given a mini-batch B and a training set D, one epoch is equal to $\frac{|D|}{|B|} = \frac{1560}{32} \approx 49$. Consequently, 15 epoch amounts to $15 \cdot 49 = 735$ training iterations.

12 models were trained for each period and for each stock. The trained models were used for inference on the testing periods, outputting the probabilities of the softmax layer. Thus, at each period we have at our disposal the probabilities for each class, according to 12 different models. Using these probabilities, we produced three different type of ensembles. The first two are the Equally Weighted Ensemble and the Performance Weighted Ensemble, as described in 4.11. To demonstrate that the predictability comes from the collective presence of the models, we also produced an ensemble, in which, for predicting each observation, we considered only the best model (according to the AUC of the past 390 observations). In the following figure, we present a visualization of the AUC scores for stock NEE, as obtained by the Performance Weighted Ensemble.

[1] each mini-batch is constructed via a randomized process with replacement

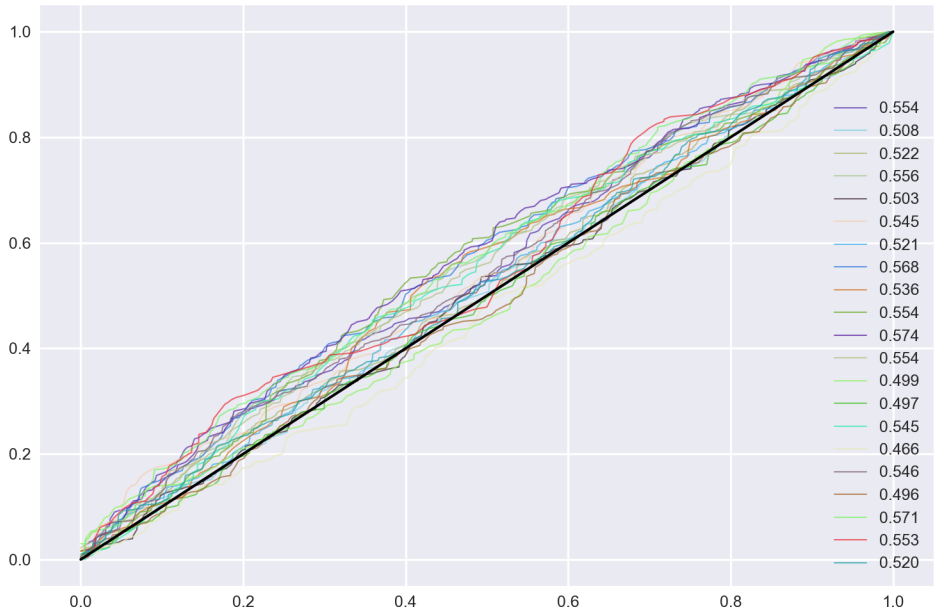

FIGURE 5.1: Receiver Operating Characteristic

The Receiver Operating Characteristic (ROC) curve for 21 periods for stock NEE. Each line is obtained by plotting the False Positive Rate(FPR) against the True Positive Rate(TPR) for a varying decision threshold. The AUC score for each curve indicates the area that covers. The identity line stands for the random classifier (area of 0.5). The majority of the lines remain consistently above the identity line and suggest a good classifier. Additionally, in most models, the area defined by the ROC curve increases in middle thresholds (0.2 - 0.8), suggesting that the optimal decision threshold exists in that area.

The combined AUC scores of the three ensemble methods can be found in Table 5.1. The t-statistic in the table indicate that all our results (AUC score) are significantly greater than an AUC score of 0.50, or in other words better than random. This suggest evidence against the weak form of efficiency in the US markets, since we should not be able to make better than random predictions using past market information. The results in Table 5.1 demonstrate that using the AUC score of past observations to weight the models in the ensemble, can produce a better performing model. Additionally, this provides evidence on the existence of patterns in the high-frequency US stock market that are not only durable, but also discoverable.

TABLE 5.1: AUC scores for the LSTM ensembles

Average AUC scores for the 22 stocks used in the research. The t-statistic is used to measure if each AUC score is significantly better than random (0.50). In bold is the best model for the particular stock. Nearly all AUC scores for the Equally Weighted and Performance Weighted ensembles are significantly better than random at $\alpha = 0.01$. One asterisk indicates significance only at $\alpha = 0.1$, double asterisk indicates significance at $\alpha = 0.05$, while the triple one indicates significance at $\alpha = 0.01$. The ensemble, which is comprised from only the best model at each time, produced inferior results than the other two, in both actual absolute terms and significance.

Stock	**Equally Weighted Ensemble**	**Best Model Ensemble**	**Performance Weighted Ensemble**
BA	0.5153 (4.13)***	0.5017 (0.30)	**0.5224** (4.24)***
F	0.5232 (4.70)***	0.5179 (2.95)***	**0.5355** (5.86)***
DHR	0.5144 (2.77)***	0.5141 (2.14)**	**0.5231** (3.52)***
KO	0.5173 (4.20)***	0.5138 (2.41)**	**0.5251** (3.22)***
MO	0.5110 (2.61)***	0.5042 (0.71)	**0.5201** (3.08)***
MAR	0.5251 (7.17)***	0.5346 (4.29)***	**0.5346** (5.30)***
AMT	0.5086 (1.72)**	0.4959 (-0.53)	**0.5132** (1.99)**
MCD	0.5223 (3.68)***	0.5222 (3.52)***	**0.5248** (3.62)***
DIS	0.5150 (5.03)***	0.5185 (2.72)***	**0.5196** (3.09)***
GE	0.5179 (3.51)***	0.5129 (2.09)**	**0.5251** (3.93)***
EOG	0.5155 (3.48)***	0.5082 (1.15)	**0.5193** (2.96)***
GS	0.5149 (3.28)***	0.5194 (3.28)***	**0.5362** (5.38)***
MET	0.5178 (3.33)***	0.5090 (1.30)	**0.5218** (3.47)***
BK	0.5173 (4.78)***	0.5102 (1.76)**	**0.5254** (3.92)***
AMGN	0.5161 (2.88)***	0.5094 (2.06)**	**0.5224** (2.91)***
ABT	0.5057 (1.55)*	0.5111 (2.07)**	**0.5190** (3.12)***
AET	0.5133 (2.95)***	0.5255 (5.40)***	**0.5263** (3.25)***
NEE	0.5207 (4.73)***	0.5103 (1.47)*	**0.5304** (4.76)***
EXC	0.5102 (2.23)**	0.4920 (-1.26)	**0.5166** (2.19)**
IBM	0.5112 (2.47)**	0.5064 (1.05)	**0.5167** (2.43)**
ATVI	0.5106 (3.39)***	0.5066 (0.94)	**0.5141** (2.40)**
NVDA	0.5186 (4.27)***	0.5065 (1.17)	**0.5242** (4.45)***

5.2 Lasso and Ridge Logistic Regressions

To provide another benchmark for the Performance Weighted Ensemble, other than the random guessing, the Lasso logistic regression and the Ridge logistic regression are used.

A weight decay of $\lambda = 0.1$ was applied for the regularization. We employ both of them on the same periods, that the ensemble was tested. The AUC scores for each stock using the Lasso and Ridge logistic regression can be observed in Table 5.2. The two models produce similar results to one another, which are also significantly better than random, as indicated by the t-statistic. Since, both of them are fairly simplistic models, compared to the LSTM, we may infer that an important part of the predictability comes from the features used in this research.

TABLE 5.2: AUC scores for Lasso and Ridge Logistic Regressions

Average AUC performance for the 22 stocks by employing Lasso and Ridge logistic regressions with a weight decay of $\lambda = 0.1$. In bold is the best model for the particular stock. The t-statistic in the parenthesis indicates whether the result is significantly better than random (AUC = 0.5). One asterisk indicates significance only at $\alpha = 0.1$, double asterisk indicates significance at $\alpha = 0.05$, while the triple one indicates significance at $\alpha = 0.01$. The majority of the scores are significant at $\alpha = 0.05$ or $\alpha = 0.01$. Furthermore we can say that the two methods produce comparable resutls.

Stock	**Lasso LR**	**Ridge LR**
BA	0.5176 (3.27)***	**0.5182** (3.30)***
F	0.5198 (2.90)***	**0.5216** (3.04)***
DHR	**0.5176** (1.99)**	0.5167 (1.88)**
KO	**0.5236** (4.48)***	0.5228 (3.77)***
MO	**0.5190** (2.16)**	0.5142 (1.65)*
MAR	0.5159 (2.14)**	**0.5171** (2.09)**
AMT	**0.5097** (1.42)**	0.5087 (1.20)
MCD	0.5127 (2.63)***	**0.5150** (2.99)***
DIS	0.5104 (1.91)**	**0.5126** (2.24)**
GE	0.5155 (1.83)**	**0.5161** (1.93)**
EOG	0.5069 (1.07)	**0.5070** (1.17)
GS	**0.5270** (3.64)***	0.5236 (3.39)***
MET	0.5168 (3.03)***	**0.5197** (3.58)***
BK	0.5157 (2.62)***	**0.5199** (2.85)***
AMGN	**0.5158** (2.49)**	0.5132 (2.30)**
ABT	0.5099 (2.07)**	**0.5117** (2.40)**
AET	**0.5104** (1.44)*	0.5086 (1.19)
NEE	0.5273 (3.41)***	**0.5276** (3.64)***
EXC	**0.5239** (3.45)***	0.5221 (3.07)***
IBM	0.5095 (1.36)*	**0.5106** (1.59)*
ATVI	**0.5088** (1.41)*	0.5056 (1.00)
NVDA	0.5123 (2.34)**	**0.5138** (2.73)***

5.3 Performance Weighted Ensemble vs Benchmarks

In the following table, we provide a comparison in terms of AUC score, of the Performance Weighted Ensemble, against the other methods. We excluded the Best Model Ensemble, since results were not that good.

TABLE 5.3: Performance Weighted Ensemble against the other methods

The Performance Weighted Ensemble produces consistently better than the other methods, with the only exception being for stock EXC, where it underperformed with respect to Lasso and Ridge classifiers. In bold is the best model for the particular stock and the values in parenthesis indicate the paired t-test between each method and the Performance Weighted Ensemble for 21 periods. One asterisk indicates significance only at $\alpha = 0.1$, double asterisk indicates significance at $\alpha = 0.05$, while the triple one indicates significance at $\alpha = 0.01$.

Stock	**Lasso LR**	**Ridge LR**	**Equally Weighted Ensemble**	**Performance Weighted Ensemble**
BA	0.5176 (0.63)	0.5182 (0.54)	0.5153 (1.09)	**0.5224**
F	0.5198 (1.72)**	0.5216 (1.49)*	0.5232 (1.57)*	**0.5355**
DHR	0.5176 (0.50)	0.5167 (0.58)	0.5144 (1.03)	**0.5231**
KO	0.5236 (0.15)	0.5228 (0.23)	0.5173 (0.88)	**0.5251**
MO	0.5190 (0.09)	0.5142 (0.55)	0.5110 (1.17)	**0.5201**
MAR	0.5159 (1.89)**	0.5171 (1.67)*	0.5251 (1.28)	**0.5346**
AMT	0.5097 (0.37)	0.5087 (0.46)	0.5086 (0.56)	**0.5132**
MCD	0.5127 (1.45)*	0.5150 (1.15)	0.5223 (0.27)	**0.5248**
DIS	0.5104 (1.10)	0.5126 (0.83)	0.5150 (0.65)	**0.5196**
GE	0.5155 (0.90)	0.5161 (0.85)	0.5179 (0.87)	**0.5251**
EOG	0.5069 (1.36)*	0.5070 (1.40)*	0.5155 (0.48)	**0.5193**
GS	0.5270 (0.91)	0.5236 (1.29)	0.5149 (2.63)***	**0.5362**
MET	0.5168 (0.59)	0.5197 (0.25)	0.5178 (0.48)	**0.5218**
BK	0.5157 (1.09)	0.5199 (0.57)	0.5173 (1.09)	**0.5254**
AMGN	0.5158 (0.67)	0.5132 (0.96)	0.5161 (0.67)	**0.5224**
ABT	0.5099 (1.18)	0.5117 (0.93)	0.5057 (1.86)**	**0.5190**
AET	0.5104 (1.47)*	0.5086 (1.63)*	0.5133 (1.41)*	**0.5263**
NEE	0.5273 (0.30)	0.5276 (0.28)	0.5207 (1.24)	**0.5304**
EXC	**0.5239** (-0.71)	0.5221 (-0.52)	0.5102 (0.72)	0.5166
IBM	0.5095 (0.74)	0.5106 (0.64)	0.5112 (0.67)	**0.5167**
ATVI	0.5088 (0.62)	0.5056 (1.04)	0.5106 (0.52)	**0.5141**
NVDA	0.5123 (1.58)*	0.5138 (1.39)*	0.5186 (0.81)	**0.5242**

With the exception of one stock (EXC), the Performance Weighted Ensemble, produces better results than the other methods. Some of them are also significant in $\alpha = 0.1$ or lower thresholds.

Finally, we wanted to test the hypothesis that all the information contained in Lasso and Ridge classifiers is also present in the Performance Weighted Ensemble. Since, both the logistic regression models are fairly simple, we wanted to examine the possibility that the Performance Weighted Ensemble lost some basic information due to its advanced complexity. To do so, we included the Lasso and Ridge classifiers in the ensemble and tested for improvement in the produced AUC scores. The results with the inclusion of the extra models, along with the paired t-test can be be found in Table 5.4.

TABLE 5.4: Inclusion of Lasso and Ridge in the ensemble

AUC scores of the Performance Weighted Ensemble containing only LSTMs and the the Performance Weighted Ensemble containing LSTMs and the Lasso and Ridge logistic classifiers. In bold is the best model for the particular stock. We can only observe minor differences between the two test models and all of them are insignificant at $\alpha = 0.25$.

Stock	**Performance Weighted Ensemble**	**Performance Weighted Ensemble +Lasso + Ridge**	**Paired t-test**
BA	0.5224	**0.5237**	0.18
F	**0.5355**	0.5354	0.00
DHR	0.5231	**0.5242**	0.12
KO	0.5251	**0.5254**	0.03
MO	0.5201	**0.5203**	0.02
MAR	**0.5346**	0.5355	0.10
AMT	0.5132	0.5132	0.00
MCD	0.5248	**0.5252**	0.04
DIS	0.5196	**0.5197**	0.02
GE	0.5251	**0.5254**	0.03
EOG	**0.5193**	0.5191	-0.03
GS	0.5362	**0.5368**	0.07
MET	0.5218	**0.5226**	0.10
BK	0.5254	0.5254	0.01
AMGN	0.5224	0.5224	-0.01
ABT	0.5190	**0.5195**	0.06
AET	0.5263	**0.5269**	0.05
NEE	0.5304	**0.5310**	0.07
EXC	0.5166	**0.5191**	0.23
IBM	0.5167	**0.5168**	0.01
ATVI	0.5141	**0.5145**	0.05
NVDA	**0.5242**	0.5236	-0.07

The inclusion produced minor changes in the AUC scores of the Performance Weighted Ensemble, which are also insignificant. As expected, the largest change was for stock EXC, that both Lasso and Ridge had outperformed the Performance Weighted Ensemble. Considering the results of Table 5.4, we can conclude that no real value can be added by including the Lasso and Ridge classifiers in the ensemble, and thus all the information is already incorporated in the Performance Weighted Ensemble.

Chapter 6

Conclusion

In this thesis, we proposed a framework for predictions in high-frequency, stock market settings. Our framework is based on an ensemble model that combines multiple Long Short-term Memory networks through a robust, performance-evaluated, weighting method.

We conducted a large-scale study, by doing data analysis and processing on 44 US large cap stocks, which can be translated to 30GB in raw data or 420 million in number of trades. The raw trade data of every stock were aggregated into 5-minute intervals, which were used to create a large number of technical indicators on different time frames. From the aggregated stock data, we further created sector datasets that were used in the predictive modeling. The data were used to train a total of 5544 LSTM networks that were merged to create 462 ensemble models.

The LSTM networks used in this research included new, state-of-art deep learning techniques, such as dropout and layer normalization. Additionally, a large part of Chapter 4 can serve as an extensive manual for applying deep and recurrent models.

By testing the predictive power of the proposed ensemble on 22 stocks, it was proved to be an upgrade on regular ensemble weighting methods, like an equally weighted one or one that considers only the best individual model. It furthermore outperformed two simpler models, the Lasso and Ridge logistic regressors. Additionally, by showing that incorporating the simpler models into the ensemble, no significant gain was achieved, we proved that the ensemble dominates them in terms of information.

Finally, all of the trained models were found to be statistically better than random, for the majority of the tested stocks, thus suggesting evidence against the weak form of market efficiency.

Chapter 7

Future Research

Research in Artificial Intelligence(AI) methods is growing rapidly, and most probably, along with Blockchain technologies, will shape the world of Business & Finance in the forthcoming future. Although AI methods, like deep neural networks, are incredibly powerful, with estimates saying that they will reach the computational capacity of a human brain by the year 2025, they lack in terms of efficiency. The issue at hand also applies to this particular research. In spite the fact that our proposed model outperforms the set benchmarks, it is much more computationally expensive than them. The training of the ensemble of a single stock required approximately 150 minutes, while the training of a Lasso or Ridge logistic regression need only some minutes. With this in mind, we believe that research should be aimed at maximizing the ratio of predictive accuracy to energy, instead of predictive accuracy alone.

This is also apparent when it comes to high-frequency trading. One should not be able to make just good predictions, but also fast predictions. We believe that our method can be further optimized by tuning the models. Unfortunately, our research was limited in terms of hardware and thus most of the hyperparameters were chosen by means of trial-and-error experiments or heuristic rules, found in the literature. Correct tuning can be carried out by setting up a grid search through candidate values, or even by an evolutionary algorithm.

The possibility of also including news sentiment data in this framework should also be researched. Although, when it comes to high-frequency data, news sentiment is usually too sparse to be modeled efficiently, it could potentially add more value to the whole framework.

Finally, we would like to point out that this research was aimed at achieving a general predictability, by estimating class probabilities on a varying decision threshold. Profitability of this framework, and in general profitability in high-frequency trading, should also be researched. This can be accomplished by optimizing the decision threshold, for which an optimal value, was shown to exist between 0.2 and 0.8.

Appendix A

Appendix

Table A.1: Stock Universe

Stocks used for this research , along with the sector they belong, their primary competitor and the exchange that they are traded at.

Stock Name (Symbol)	**Sector**	Competitor Name (Symbol)	**Exchange**
The Boeing Company (BA)	Capital Goods	United Technologies Corporation (UTX)	NYSE
Ford Motor Company (F)	Capital Goods	General Motors Company (GM)	NYSE
Danaher Corporation (DHR)	Capital Goods	Thermo Fisher Scientific Inc (TMO)	NYSE
The Coca-Cola Company (KO)	Consumer Non-Durables	Pepsico, Inc. (PEP)	NYSE
Altria Group (MO)	Consumer Non-Durables	Philip Morris International Inc. (PM)	NYSE
Marriott International (MAR)	Consumer Services	Las Vegas Sands Corp. (LVS)	NASDAQ
American Tower Corporation, REIT (AMT)	Consumer Services	Simon Property Group, Inc. (SPG)	NYSE
McDonalds Corporation (MCD)	Consumer Services	Starbucks Corporation (SBUX)	NASDAQ
The Walt Disney Company (DIS)	Consumer Services	Time Warner Inc. (TWX)	NYSE
General Electric Company (GE)	Energy	Emerson Electric Company (EMR)	NYSE
EOG Resources, Inc. (EOG)	Energy	Occidental Petroleum Corporation (OXY)	NYSE
The Goldman Sachs Group, Inc. (GS)	Finance	Morgan Stanley (MS)	NYSE
MetLife, Inc. (MET)	Finance	Prudential Financial, Inc. (PRU)	NYSE
The Bank Of New York Mellon Corporation (BK)	Finance	U.S. Bancorp (USB)	NYSE
Amgen Inc. (AMGN)	Health Care	Gilead Sciences, Inc. (GILD)	NASDAQ
Abbott Laboratories (ABT)	Health Care	Merck & Company, Inc. (MRK)	NYSE
Aetna Inc. (AET)	Health Care	Cigna Corporation (CI)	NYSE
NextEra Energy, Inc. (NEE)	Public Utilities	Dominion Energy, Inc. (D)	NYSE
Exelon Corporation (EXC)	Public Utilities	Duke Energy Corporation (DUK)	NYSE
International Business Machines Corporation (IBM)	Technology	HP Inc. (HPQ)	NYSE
Activision Blizzard, Inc (ATVI)	Technology	Adobe, Inc (ADBE)	NASDAQ
NVIDIA Corporation (NVDA)	Technology	Intel Corporation (INTC)	NASDAQ

TABLE A.2: Trade conditions

Summary of possible trade conditions and their contribution to Open, Close, Low, High and trading Volume. Most of the the conditions do not contribute to the open/close prices, which are of interest. Source: [60], p.118

		CONSOLIDATED		PARTICIPANT			SALE COND UPDATE VOLUME
CODE	SALE CONDITION	LAST	HIGH/ LOW	OPEN	LAST	HIGH/ LOW	
Blank	No Sale Condition required within the Category it appears (Long Trade format only)	N/A					
@	REGULAR TRADE (Indicates a trade with no associated conditions)	YES	YES	#4	YES	YES	YES
B	AVERAGE PRICE TRADE	NO	NO	NO	NO	NO	YES
C	CASH TRADE (Same Day Clearing)	NO	NO	NO	NO	NO	YES
E	AUTOMATIC EXECUTION	YES	YES	#4	YES	YES	YES
F	INTERMARKET SWEEP ORDER	YES	YES	#4	YES	YES	YES
H	PRICE VARIATION TRADE	NO	NO	NO	NO	NO	YES
I	ODD LOT TRADE	NO	NO	NO	NO	NO	YES
K	RULE 127 (NYSE Only) or RULE 155 (NYSE MKT only)	YES	YES	#4	YES	YES	YES
L	SOLD LAST (Late Reporting)	#3	YES	#4	YES	YES	YES
M	MARKET CENTER OFFICIAL CLOSE	NO	NO	NO	YES	YES	NO
N	NEXT DAY TRADE (Next Day Clearing)	NO	NO	NO	NO	NO	YES
O	MARKET CENTER OPENING TRADE	#1	YES	YES	#2	YES	YES
P	PRIOR REFERENCE PRICE	#2	YES	#4	#2	YES	YES
Q	MARKET CENTER OFFICIAL OPEN	NO	NO	YES	NO	YES	NO
R	SELLER	NO	NO	NO	NO	NO	YES
T	EXTENDED HOURS TRADE	NO	NO	NO	NO	NO	YES
U	EXTENDED HOURS SOLD (Out of Sequence)	NO	NO	NO	NO	NO	YES
V	CONTINGENT TRADE	NO	NO	NO	NO	NO	YES
X	CROSS TRADE	YES	YES	#4	YES	YES	YES
Z	SOLD (Out of Sequence)	#2	YES	#4	#2	YES	YES
4	DERIVATIVELY PRICED	#2	YES	#4	#2	YES	YES
5	MARKET CENTER REOPENING TRADE	YES	YES	#4	YES	YES	YES
6	MARKET CENTER CLOSING TRADE	YES	YES	#4	YES	YES	YES
7	QUALIFIED CONTINGENT TRADE	NO	NO	NO	NO	NO	YES
8	RESERVED	NO	NO	NO	NO	NO	TBD
9	CORRECTED CONSOLIDATED CLOSE PRICE as per LISTING MARKET	YES	YES	NO	NO	NO	NO

TABLE A.3: Basic features

Features extracted from the raw trade data. Formulas indicate how these features are extracted from the split into 5-minute intervals datasets. N is the number of trades that took place in this interval, P_i is the price of trade at moment i, S_i is the size of the trade and $\Delta \boldsymbol{P}$ denotes the price differences vector. $\boldsymbol{I}$ stands for indicator function.

Features	**Formula**
Open (P_{open})	P_1
Close (P_{close})	P_N
Low (P_{low})	$min(\boldsymbol{P})$
High (P_{high})	$max(\boldsymbol{P})$
Volume Weighted Average Price (P_{vwap})	$\sum_i^N P_i S_i / \sum_i^N S_i$
Mean Price Difference ($MeanPriceD$)	$(P_1 - P_N)/(N-1)$
Max Price Difference ($MaxPriceD$)	$\max(\Delta \boldsymbol{P})$
Standard Deviation of Price Differences ($StdPriceD$)	$\sqrt{\frac{1}{N-1}\sum_{i=1}^{N-1}(\Delta P_i - MeanPriceD)^2}$
Total number of Trades	N
Number of Market Sweep Trades	$\sum_{i=1}^N \boldsymbol{I}_{cond=F}$
Max Trade Size	$max(\boldsymbol{S})$
Mean Trade Size	$\sum_{i=1}^N S_i/N$
Volume	$\sum_i^N S_i$
Positive Volume ($Volume^+$)	$\sum_{i=1}^N \boldsymbol{I}_{\Delta P_i>0} S_i$
Negative Volume ($Volume^-$)	$\sum_{i=1}^N \boldsymbol{I}_{\Delta P_i<0} S_i$
Neutral Volume ($Volume^0$)	$\sum_{i=1}^N \boldsymbol{I}_{\Delta P_i=0} S_i$

TABLE A.4: Split history

Symbol	**Date**	**Split ratio**
EOG	01/04/2014	2 for 1
OXY	01/12/2014	1042 for 1000
SPG	29/05/2014	1063 for 1000
TWX	09/06/2014	1043 for 1000

TABLE A.5: Advanced features

Here we present a summary of all the features in each dataset and how many different versions of them are included on stock level and on sector/index level.

Feature	**Versions in stock**	**Versions in sector/index**
Percentage change in Price	7	6
Price Difference (on raw level)	3	0
Std of Returns (cross sectionally)	0	1
Volume	4	3
Accumulation/Distribution Index	6	6
True Range	1	1
Probability and Conditional Probability	10	10
Rolling Regression	4	1
Sharpe Ratio	5	5
Price Disagreement	6	6
Price Polarity	6	6
Bollinger Bands	10	10
Stochastic Oscillator	5	5
Relative Strength Index	5	5
Commodity Channel Index	5	5
Average Directional Index	5	5
Double and Triple Exponentially Smoothed Returns	5	5
Moving Average Convergence/Divergence	5	5
Money Flow Index	5	5
	97	90

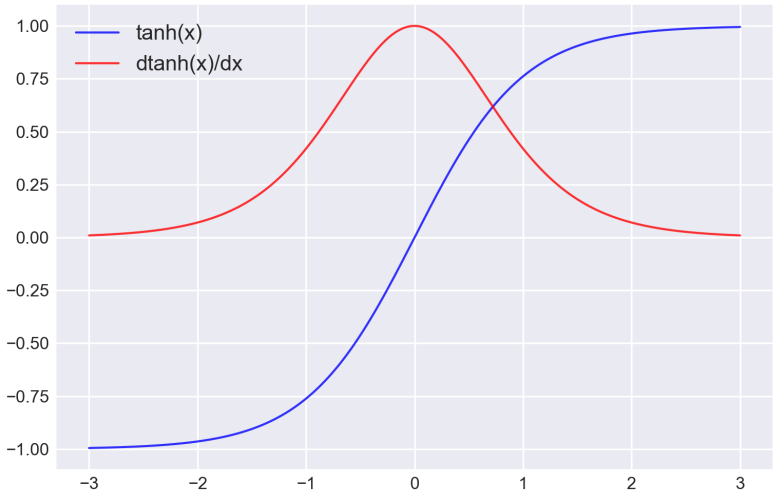

FIGURE A.1: Tanh function and its derivative

Tanh has outputs in $[-1, 1]$ and strong gradients around zero. Neuron saturation may happen if inputs x diverge significantly from it. For that reason we would optimally have its input x to be centered zero. Layer normalization, as well as Xavier initialization will ensure that.

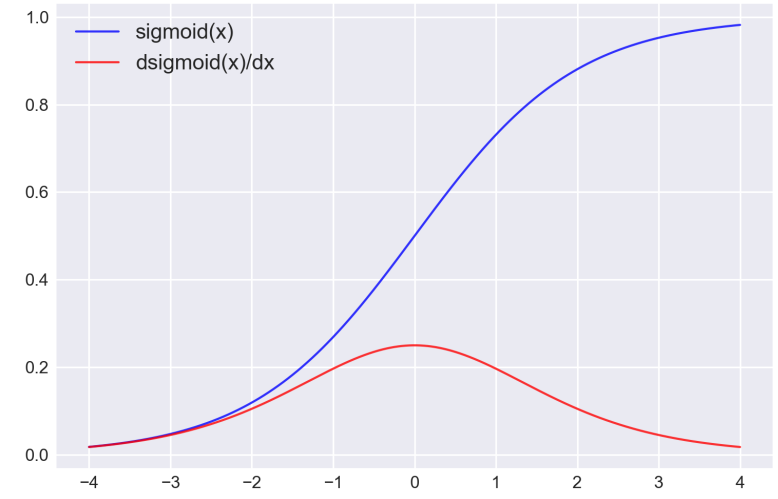

FIGURE A.2: Sigmoid function and its derivative

The sigmoid function, with outputs in $[0, 1]$, has its strongest gradients around zero, while neuron saturation may happen if inputs x diverge significantly from it. For that reason we would optimally have its input x to be centered zero. Layer normalization, as well as Xavier initialization will ensure that.

A.1 Technical Indicators

A.1.1 Moving Averages

Many of the Technical Indicators used in this research involve the calculation of several moving averages. Moving averages are used to smooth noisy time-series and highlight short or long-term trends.

The Simple Moving Average $\text{SMA}(x_t, n)$ is composed from the mean of the last n observations of time-series x_t.

$$SMA(x_t, n) = \frac{1}{n}\sum_{i=1}^{n} x_{t-i} \tag{A.1}$$

The Exponential Weighed Moving Average $\text{EWMA}(x, \alpha)$ is calculated by exponential decreasing the weight of observations x_i with respect to their distance from x_t.

$$EWMA(x_t, \alpha) = \alpha x_t + (1-\alpha)EWMA(x_{t-1}, \alpha) \tag{A.2}$$

A.1.2 Sharpe Ratio

Sharpe ratio is measure that summarizes the trade-off between return and risk. Although mostly used for evaluating investment strategies, it can also be used as a technical indicator. Sharpe ratio is calculated for $n = [6, 12, 36, 78, 234]$.

$$Sharpe_t = \frac{(\Pi_{i=1}^{n}(1 + R_{t+1-i}) - 1) - R_{rf,t}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(R_i - \frac{1}{n}\sum_{k=1}^{n}(1 + R_{t+1-k}))^2}} \tag{A.3}$$

Where R_{rf} is the risk free rate.

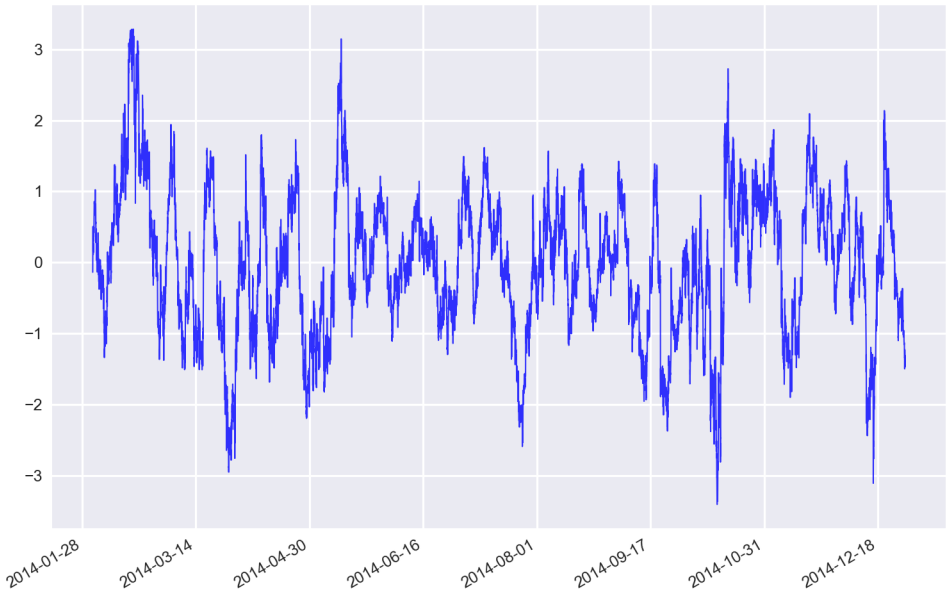

FIGURE A.3: Sharpe ratio for n = 234

A.1.3 Accumulation/Distribution Index (ADI)

ADI is a popular volume indicator, which is calculated as the cumulative sum of the product of Close Location Value (CLV) and the trading volume.

$$CLV_t = \frac{(P_{close,t} - P_{low,t}) - (P_{high,t} - P_{close,t})}{P_{high,t} - P_{low,t}} \tag{A.4}$$

$$ADI_t = \sum_{i=1}^{t} CLV_t \, Volume_i \tag{A.5}$$

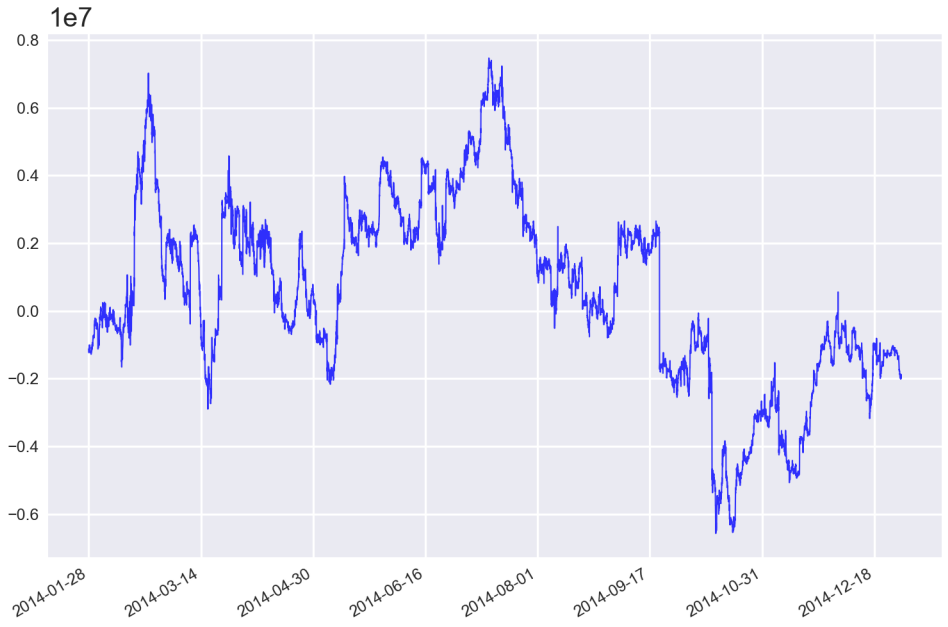

FIGURE A.4: Accumulation/Distribution Index

A.1.4 Bollinger Bands (BB)

Bollinger Bands are used to construct volatility indicators by evaluating the distances of current typical price to the previous n typical prices. For the purpose of this research we are using $d = 3$ and $n = [6, 12, 36, 78, 234]$.

$$MiddleBB_t = SMA(P_{typical,t}, n) \tag{A.6}$$

$$\sigma = \sqrt{\frac{1}{n}\sum_{i}^{n} P_{typical,t-i} - MiddleBB_t} \tag{A.7}$$

$$UpperBB = MiddleBB_t + d\,\sigma \tag{A.8}$$

$$LowerBB = MiddleBB_t - d\,\sigma \tag{A.9}$$

$$\%b = \frac{P_{close,t} - LowerBB_t}{UpperBB_t - LowerBB_t} \tag{A.10}$$

$$Bandwith_t = \frac{UpperBB_t - LowerBB_t}{MiddleBB_t} \tag{A.11}$$

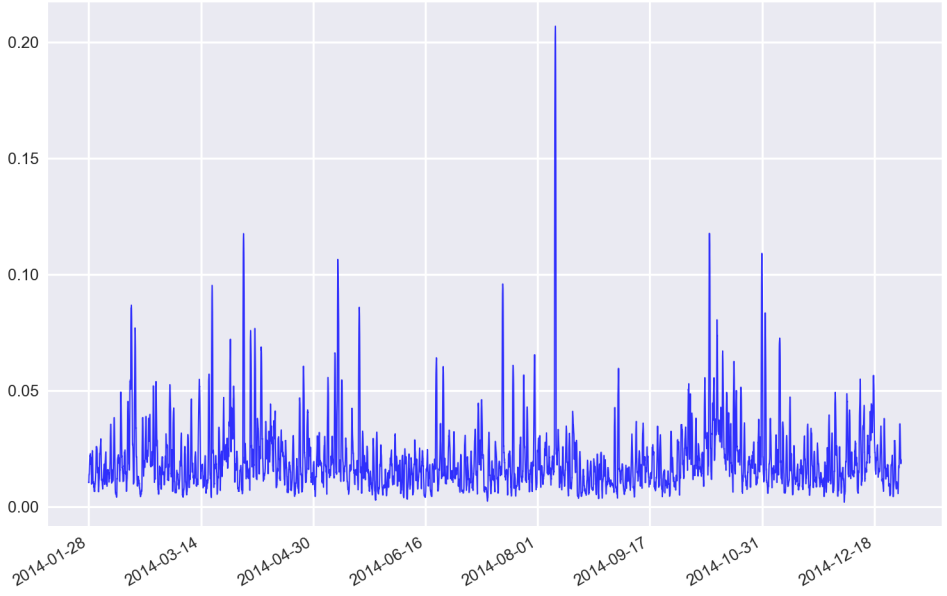

FIGURE A.5: Bandwith for n = 36

A.1.5 Stochastic Oscillator

The Stochastic Oscillator is a momentum indicator that aims to predict the oscillations of the price by taking into account the price range during the last n observations.

$$K_t = \frac{P_{close,t} - L_n}{H_n - L_n} \tag{A.12}$$

Where $L_n = min(P_{low,t-n}, ..., P_{low,t})$ and $H_n = min(P_{high,t-n}, ..., P_{high,t})$.

Finally, the Stochastic Oscillator is a Simple Moving Average of length m.

$$StochOsci_t = 100\ SMA(K_t, m) \tag{A.13}$$

The values used for this research are $n = [6, 12, 36, 78, 234]$ and $m = [3, 3, 2, 2, 2]$ accordingly.

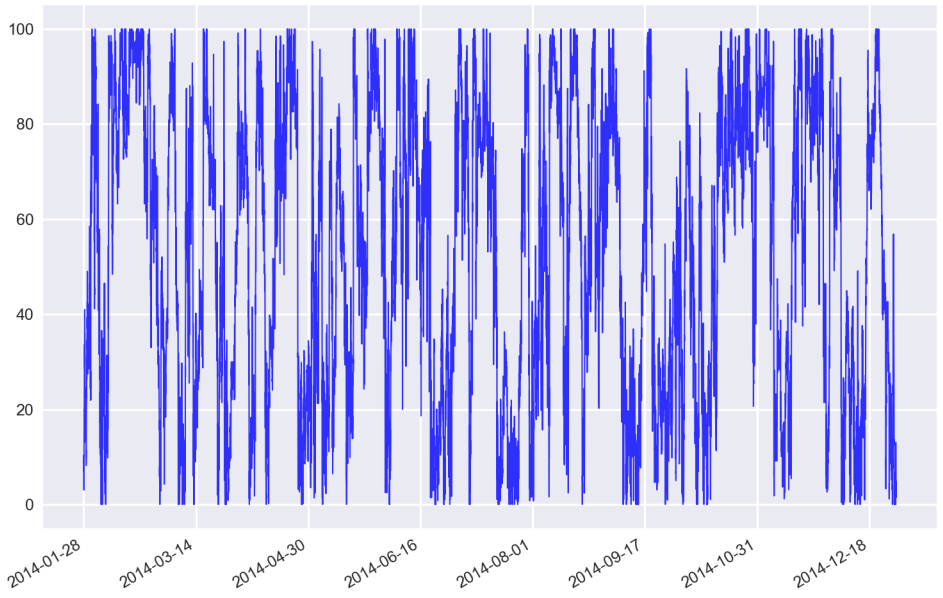

FIGURE A.6: Stochastic Oscillator for n = 234

A.1.6 Relative Strength Index (RSI)

The Relative Strength Index is another momentum indicator that uses the velocity and magnitude of price movements to identify if an asset is overbought or oversold. It ranges from 0 to 100, with low values indicating that the asset is oversold, while high values indicate that is overbought. RSI is calculate for $n = [6, 12, 36, 78, 234]$.

$$U_t = \begin{cases} P_{close,t} - P_{close,t-1} & , P_{close,t} > P_{close,t-1} \\ 0 & , P_{close,t} \leq P_{close,t-1} \end{cases}$$

$$D_t = \begin{cases} 0 & , P_{close,t} > P_{close,t-1} \\ P_{close,t} - P_{close,t-1} & , P_{close,t} \leq P_{close,t-1} \end{cases}$$

$$RSI_t = 100 - \frac{100}{1 + \frac{EWMA(U_t, 1/n)}{EWMA(D_t, 1/n)}} \tag{A.14}$$

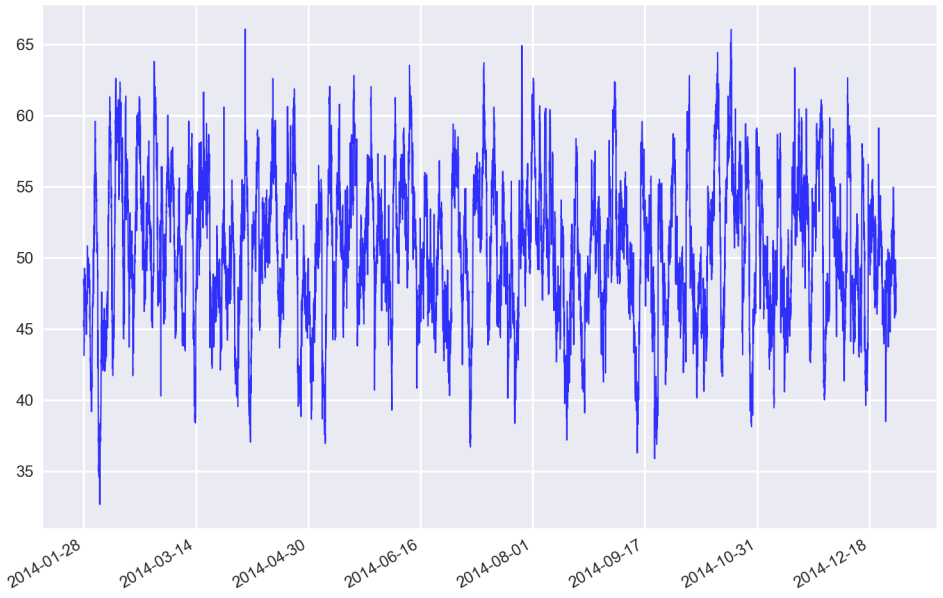

FIGURE A.7: Relative Strength Index for n = 78

A.1.7 Commodity Channel Index (CCI)

The Commodity Channel Index functions similarly to the RSI, indicating whether an asset is overbought or oversold. The main difference is that CCI operates on the typical price. It ranges from -100 to 100, with small values hinting that the stock is oversold, while large hint the opposite. The CCI is calculate for $n = [6, 12, 36, 78, 234]$.

$$CCI_t = \frac{1}{0.015} \frac{P_{typical,t} - SMA(P_{typical,t}, n)}{MAD(P_{typical,t}, n)} \tag{A.15}$$

Where MAD is the Mean Absolute Deviation, calculated as:

$$MAD(x_t, n) = \frac{1}{n} \sum_{i=1}^{n} |x_{t-i} - \frac{1}{n} \sum_{k=1}^{n} x_{t-k}| \tag{A.16}$$

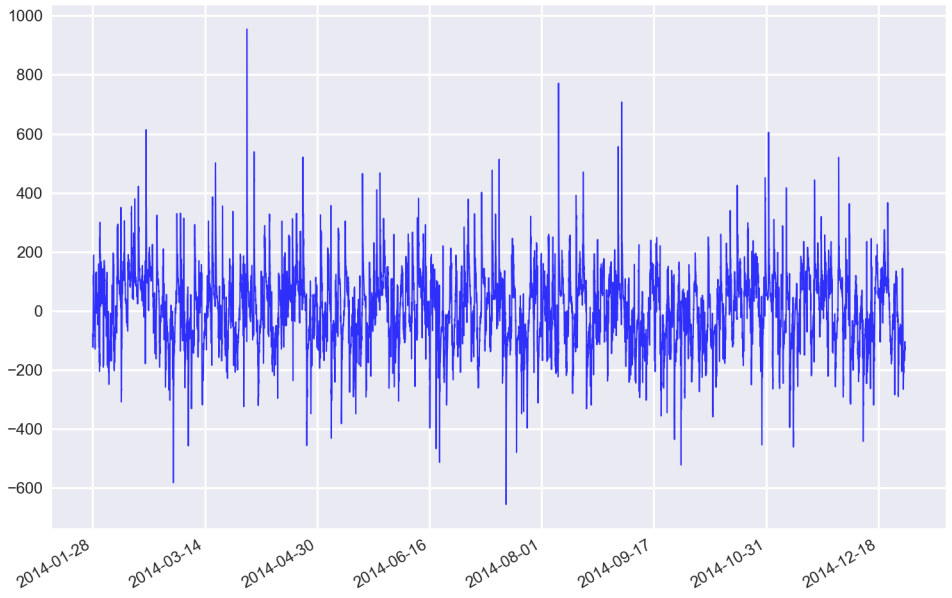

FIGURE A.8: Commodity Channel Index for n = 78

A.1.8 Average Directional Moving Index (ADX)

The Average Directional Moving Index is a combination of two other indicators, the positive directional DI^+ and the negative directional DI^-, and it used a measure of trend strength. The ADX is calculate for $n = [6, 12, 36, 78, 234]$.

$$UpMove_t = P_{high,t} - P_{high,t-1} \tag{A.17}$$

$$DownMove_t = P_{low,t-1} - P_{low,t} \tag{A.18}$$

$$DM_t^+ = \begin{cases} UpMove_t & , UpMove_t > DownMove_t \text{ and } UpMove_t > 0 \\ 0 & , \text{otherwise} \end{cases}$$

$$DM_t^- = \begin{cases} DownMove_t & , DownMove_t > UpMove_t \text{ and } DownMove_t > 0 \\ 0 & , \text{otherwise} \end{cases}$$

$$DI_t^+ = 100 \frac{EWMA(DM_t^+, 1/n)}{SMA(TR_t, n)} \tag{A.19}$$

$$DI_t^- = 100 \frac{EWMA(DM_t^-, 1/n)}{SMA(TR_t, n)} \tag{A.20}$$

Where TR stands for True Range and is defined as

$$TR_t = max(P_{high,t} - P_{low,t}, |P_{high,t} - P_{close,t-1}|, |P_{low,t} - P_{close,t-1}|) \tag{A.21}$$

Finally, ADX is defined as

$$ADX_t = 100\ EWMA(\frac{|DI_t^+ - DI_t^-|}{|DI_t^+ + DI_t^-|}, 1/n) \tag{A.22}$$

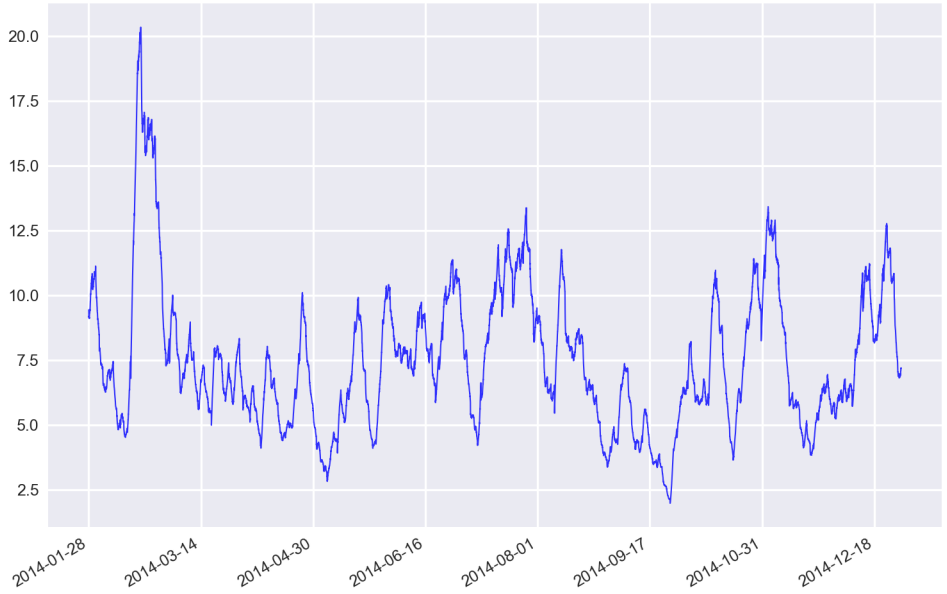

FIGURE A.9: Average Directional Moving Index for n = 78

A.1.9 Double and Triple Exponentially Smoothed Returns

Both these indicators are calculate as the percentage change of the double DIX or triple TRIX exponentially smoothed prices. DIX is calculated for $n = 234$, while TRIX for $n = [6, 12, 36, 78]$.

$$DIX_t = \frac{EWMA(EWMA(P_{close,t}, 1/n), 1/n) - EWMA(EWMA(P_{open,t}, 1/n), 1/n)}{EWMA(EWMA(P_{open,t}, 1/n), 1/n)} \tag{A.23}$$

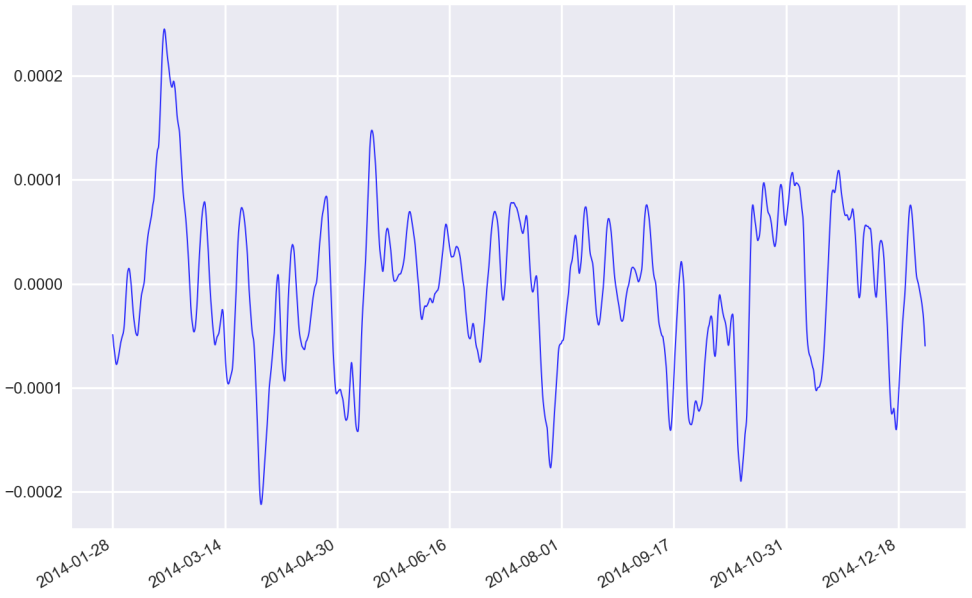

FIGURE A.10: Triple Exponentially Smoothed Returns for n = 78

A.1.10 Moving Average Convergence-Divergence (MACD)

MACD is a technical indicator composed from 3 separate time-series, used for detection changes in the trend of a price, with respect to strength, momentum, duration nd direction. First, the divergence of two $EWMA$ on the typical price is measured and then MACD is calculated as the difference of the divergence and $EWMA$ of the divergence. MACD is calculated for the following sets of $n_.$, $n_1 = [3, 6, 18, 36, 117]$, $n_2 = [6, 12, 36, 78, 234]$ and $n_3 = [2, 4, 12, 26, 78]$.

$$Divergence_t = EWMA(P_{typical,t}, 1/n_1) - EWMA(P_{typical,t}, 1/n_2) \tag{A.24}$$

$$MACD = Divergence_t - EWMA(Divergence_t, 1/n_3) \tag{A.25}$$

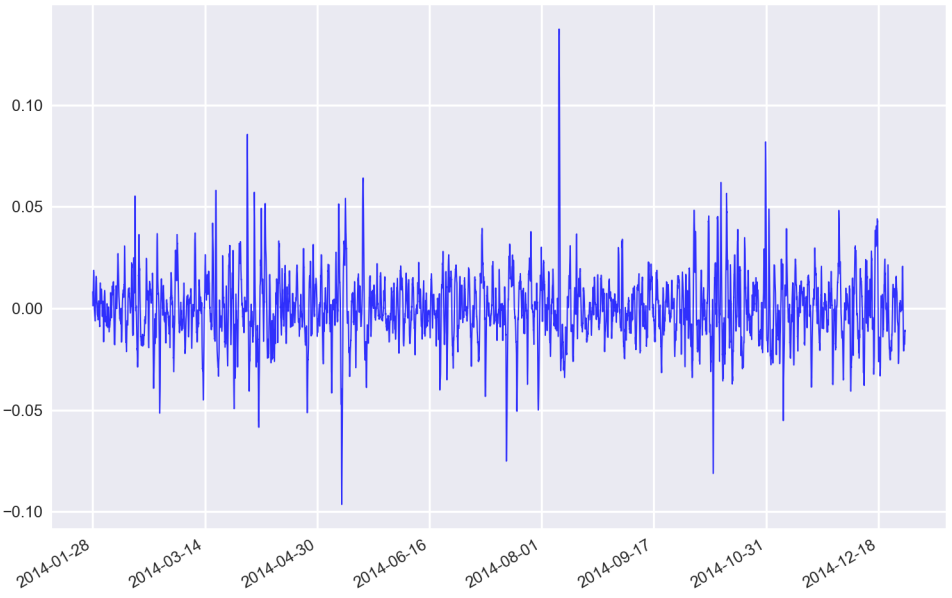

FIGURE A.11: Moving Average Convergence-Divergence for n = 78

A.1.11 Money Flow Index (MFI)

MFI is a momentum indicator based on the product of price and volume, which is defined as money flow. MFI is calculated for $n = [6, 12, 36, 78, 234]$.

$$MF_t = P_{typical,t} Volume_t \tag{A.26}$$

Then Positive Money Flow is defined as the cumulative sum of all the instances where there has been a positive change in the price. Negative Money Flow is calculated using the opposite logic.

$$MF_t^+ = \sum_{i=1}^{n} MF_i \boldsymbol{I}_{P_{typical,t-i} - P_{typicalt-i-1} > 0} \tag{A.27}$$

$$MF_t^- = \sum_{i=1}^{n} MF_i \boldsymbol{I}_{P_{typical,t-i} - P_{typicalt-i-1} < 0} \tag{A.28}$$

Where $\boldsymbol{I}$ is the indicator function.

Finally MFI is calculated as the ratio of the Positive Money Flow to the sum of Positive and Negative flows.

$$MFI_t = 100 \frac{MF_t^+}{MF_t^+ + MF_t^-} \tag{A.29}$$

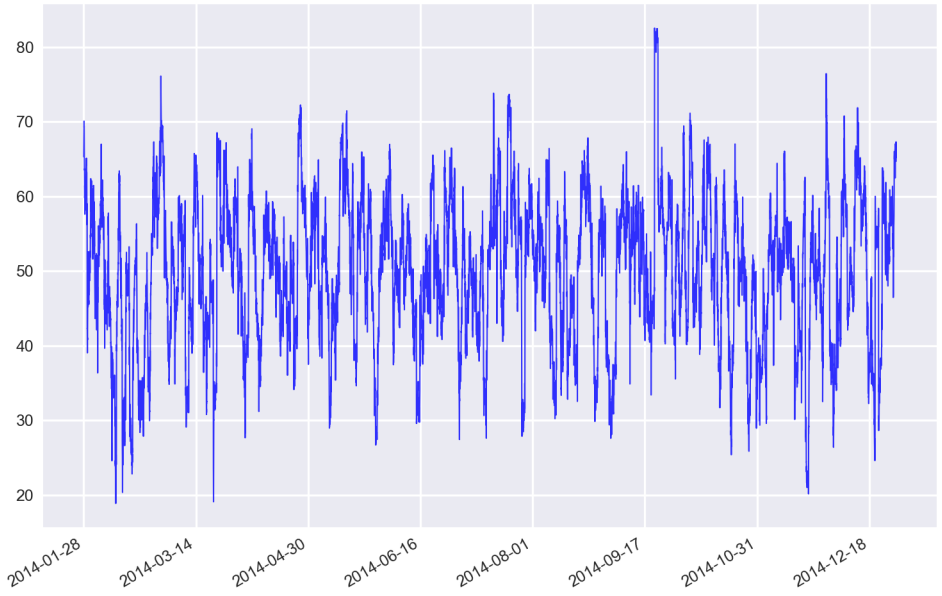

FIGURE A.12: Money Flow Index for n = 78

A.1.12 Price Disagreement and Polarity

Disagreement and Polarity are measures frequently used in sentiment analysis, calculated by the amount positive, negative and neutral news for a period of n. In this research we are calculating Disagreement and Polarity using the positive, negative and neutral trading volume, as defined in A.3. These indicator are calculated for $n = [0, 6, 12, 36, 78, 234]$.

$$Disagreement_t = |1 - |\frac{\sum_{i=1}^{n} Volume_{t-i}^+ - \sum_{i=1}^{n} Volume_{t-i}^-}{\sum_{i=1}^{n} Volume_{t-i}^+ + \sum_{i=1}^{n} Volume_{t-i}^-}|| \tag{A.30}$$

$$Polarity_t = \frac{\sum_{i=1}^{n} Volume_{t-i}^+ - \sum_{i=1}^{n} Volume_{t-i}^-}{\sum_{i=1}^{n} Volume_{t-i}^0} \tag{A.31}$$

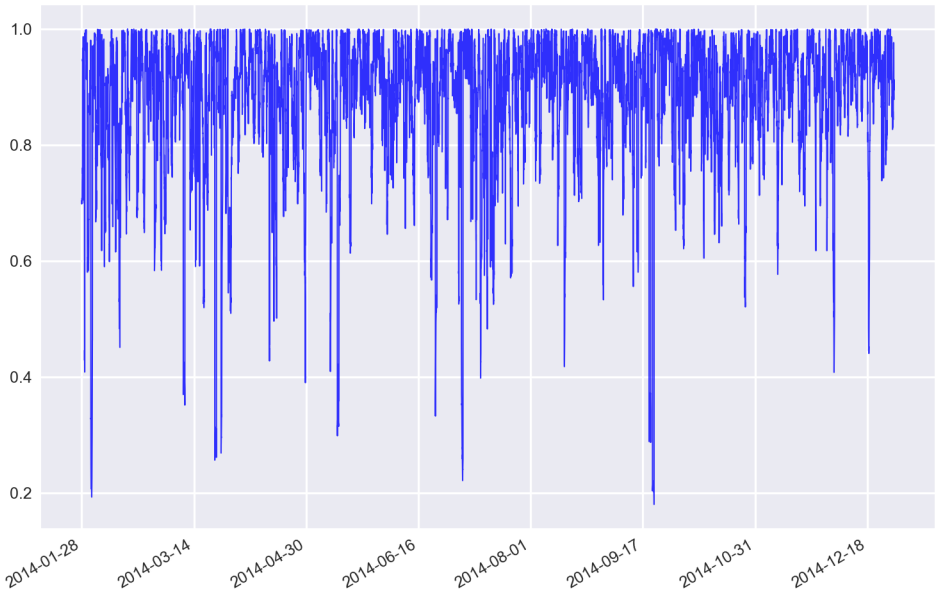

FIGURE A.13: Price Disagreement for n = 36

Bibliography

[1] E. F. Fama. Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, 25(2):383–417, December 1969 (May, 1970).

[2] E. F. Fama. Random walks in stock-market prices. *Financial Analysts Journal*, 21: 55–59, 1965.

[3] M. Haug and M. Hirschey. The january effect. *Financial Analysts Journal*, 62(5): 78–88, October 2006.

[4] J. B. Patel. The january effect anomaly reexamined in stock returns. *The Journal of Applied Business Research*, 32(1):55–59, January/February 2016.

[5] T. Mitchell. Machine learning. 2, 1997.

[6] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[7] S. Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Master's Thesis*, 1970.

[8] C. Cortes and V. N. Vapnik. Support-vector networks. *Machine Learning*, 20(3): 273–297, 1995.

[9] I. Sutskever A. Krizhevsky and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 2012.

[10] M. Mozer. A focused backpropagation algorithm for temporal pattern recognition. *Complex Systems*, 3, 1995.

[11] Y. Bengio. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2, 2009.

[12] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[13] S. Dzeroski and B. Zenko. Is combining classifiers better than selecting the best one? *Machine Learning*, pages 255–273, 2004.

[14] T. K. Ho. Random decision forests. *Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC*, pages 278–282, August 1995.

[15] A. Kent. Signal detection theory: Valuable tools for evaluating inductive learning. *Proceedings of the Sixth International Workshop on Machine Learning*, pages 160–163, 1989.

[16] P. Whittle. Hypothesis testing in time series analysis. *Almqvist Wiksells boktr*, 1951.

[17] G. E. P. Box and G. M. Jenkins. Time series analysis: forecasting and control. 1970.

[18] I. Virtanen and P. Yli-Olli. Forecasting stock market prices in a thin security market. *Omega*, 15(2):145–155, 2000.

[19] R. F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50(4):987–1007, 1982.

[20] T. Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307–327, 1986.

[21] A. Goyal. Predictability of stock return volatility from garch models. *Anderson Graduate School of Management*, 2000.

[22] P. Sadorsky. Oil price shocks and stock market activity. *Energy Economics*, 21(5): 449–469, 1999.

[23] J. Danelsson. Multivariate stochastic volatility models: Estimation and a comparison with vgarch models. *Journal of Empirical Finance*, 2(5):155–172, 1998.

[24] J. Timmer and A. S. Weigend. Modeling financial time series using state space models. *Decision Technologies for Computational Finance: Proceedings of the fifth International Conference Computational Finance*, pages 233–246, 1998.

[25] R. D. Snydera R. J. Hyndmana, A. B. Koehlerb and S. Grosean. A state space framework for automatic forecasting using exponential smoothing methods. *International Journal of Forecasting*, 18(3):439–454, 2002.

[26] S. J. Koopman D. Creal and A. Lucas. Generalized autoregressive score models with applications. *Journal of Applied Econometrics*, 28(5):777–795, 2013.

[27] K. Y. Tam and M. Kiang. Managerial applications of neural networks: the case of bank failure predictions. *Management Science*, 38(7):26–47, 1992.

[28] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[29] W. Shouyang K. L. Kin, Y. Lean and Z.Ligang. Neural network metalearning for credit scoring. *Lecture Notes in Computer Science*, 4133, 2006.

[30] A. Bagherpour. Predicting mortgage loan default with machine learning methods. *economics.ucr.edu*, 2016.

[31] S. Rendle. Factorization machines. *Proceedings of the 2010 IEEE International Conference on Data Mining*, pages 995–1000, 2010.

[32] A. Sadhwani J. A. Sirignano and K. Giesecke. Deep learning for mortgage risk. *arXiv:1607.02470v1 [q-fin.ST]*, 2016.

[33] B. E. Patuwo D. C. Indro G. Zhang, M. Y. Hu. Artificial neural networks in bankruptcy prediction: General framework and cross-validation analysis. *European Journal of Operational Research*, 116:16–37, 1999.

[34] E. Altman F. Barbozaa, H. Kimura. Machine learning models and bankruptcy prediction. *Expert Systems With Applications*, 83:405–417, 2017.

[35] F. J. L. Iturriaga and I. P. Sanz. Bankruptcy visualization and prediction using neural networks: A study of u.s. commercial banks. *Expert Systems with Applications*, 42:2857–2869, 2015.

[36] T. Kohonen. Self-organized formation of topologically correct feature maps. *Cybernetics*, 43(1):59–69, 1982.

[37] A. O. Adetunmbi J O. Awoyemi and S. A. Oluwadare. Credit card fraud detection using machine learning techniques: A comparative analysis. *2017 International Conference on Computing Networking and Informatics (ICCNI)*, pages 1–9, 2017.

[38] A. Hernando J. Bobadilla, F. Ortega and A. Gutirrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013.

[39] W. M. Van Den Bergh D. E. Baestaens and H. Vaudrey. Options as a predictor of common stock price changes. *The European Journal of Finance*, 1(4):325–343, 1995.

[40] A. Zapanis A. N. Refenes and G. Francis. Stock performance modeling using neural networks: A comparative study with regression models. *Neural Networks*, 7(2):375–388, 1994.

[41] S. Y. Wang W. Huanga, Y. Nakamoria. Forecasting stock market movement direction with support vector machine. *Computers & Operations Research*, 32:2513–2522, 2005.

[42] P. F. Pai and C. S. Lin. A hybrid arima and support vector machines model in stock price forecasting. *Omega*, 33(6):497–505, 2005.

[43] O. S. Soliman O. Hegazy and M. A. Salam. A machine learning model for stock market prediction. *International Journal of Computer Science and Telecommunications*, 4(12), 2013.

[44] J. Kennedy and R. Eberhart. Particle swarm optimization. *Neural Networks, 1995. Proceedings., IEEE International Conference on*, 4:1942–1948 vol.4, 1995.

[45] R. A. de Oliveira D. M. Q. Nelson, A. C. M. Pereira. Stock markets price movement prediction with lstm neural networks. *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017.

[46] T. Fischer and C. Krauss. Deep learning with long short-term memory networks for financial market predictions. *AU Discussion Papers in Economics*, 11, 2017.

[47] H. Qu and Y. Zhang. A new kernel of support vector regression for forecasting high-frequency stock returns. *Mathematical Problems in Engineering*, 2016.

[48] J. Yue W. Bao and Y. Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PLOS ONE*, 12(7):1–24, 07 2017.

[49] M. D. Rechenthin. Machine-learning classification techniques for the analysis and prediction of high-frequency stock direction. *PhD (Doctor of Philosophy) thesis, University of Iowa*, 2014. URL http://ir.uiowa.edu/etd/4732.

[50] C. Mikolajczak and H. Lash. Network failure interrupts quotes, trade data for nyse stocks. *Reuters*, OCTOBER 30, 2014.

[51] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

[52] J. R. Kiros J. L. Ba and G. E. Hinton. Layer normalization. *CoRR*, abs/1607.06450v1, 2016. URL http://arxiv.org/abs/1607.06450v1.

[53] A. Krizhevsky I. Sutskever N. Srivastava, G. Hinton and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[54] A. Severyn S. Semeniuta and E. Barth. Recurrent dropout without memory loss. *CoRR*, abs/1603.05118, 2016. URL http://arxiv.org/abs/1603.05118.

[55] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research*, 9(1,2):249–256, 2010.

[56] J. D. M. Rennie. Regularized logistic regression is strictly convex. *Technical report, MIT*, 2005. URL http://qwone.com/~jason/writing/convexLR.pdf.

[57] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, abs/1206.5533, 2012. URL http://arxiv.org/abs/1206.5533.

[58] G. E. Hinton D. E. Rumelhart and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[59] N. Srivastava G. Hinton and K. Swersky. Neural networks for machine learning, lecture 6, university of toronto. 2012. URL https://www.cs.toronto.edu/~hinton/csc321/notes/lec9.pdf.

[60] Consolidated tape systems: Output multicast interface specification. *Securities Industry Automation Corporation (Manual)*, 81, June, 2017. URL https://www.nyse.com/publicdocs/ctaplan/notifications/trader-update/cts_output_spec.pdf.