FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Forecasting stock trends through Machine Learning

**José Diogo Seca**


Mestrado Integrado em Engenharia Informática e Computação

Supervisor: João Mendes-Moreira

Second Supervisor: Luís F. Teixeira

July 18, 2019

# Forecasting stock trends through Machine Learning

**José Diogo Seca**

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: António Coelho
External Examiner: Ricardo Sousa
Supervisor: João Mendes-Moreira

July 18, 2019

# Abstract

In managing their portfolio of investments, managers are constantly seeking the instruments (e.g. stocks) that consistently provide the highest returns on a long-term horizon. One possibility to improve the search for overperforming stocks is to use predictive Machine Learning techniques. Previous studies attempted to predict alpha, i.e. higher than average returns, by developing linear models based on capital market anomalies.

However, the techniques used in these studies lack the capability to capture complex non-linear dependencies. More recently, evidence suggests that some Machine Learning techniques may be able to capture these non-linearities [1], namely Long Short-Term Memory (LSTM) networks. The LSTM networks belong to the class of Recurrent Neural Networks (RNN) and are used to capture patterns in sequential data. Despite the recent technological advances in Deep Learning in the last years, little research has been done to analyze and identify long-term investment opportunities using predictive Deep Learning techniques. Most available research focuses primarily on short-term predictions and only makes use of market data, disregarding other publicly available information, such as fundamentals.

Using the financial statements and market prices of companies listed in the NASDAQ Stock Exchange and the New York Stock Exchange (NYSE) over a span of 30 years, different features (e.g. Book-to-Market ratio, Return On Equity), and different Machine Learning techniques were tested, namely the LSTM neural network. As a baseline, the buy-and-hold investment strategy was used, which comprised of buying a stock at the beginning of the time series, without ever selling. All of the developed models were be tested by simulating trading on an out-of-sample period. The models were then be compared using the Accuracy, Year-over-Year Returns, and Maximum Drawdown for the period tested.

The models tested proved to achieve Accuracy measurements higher than the Naive Classifier. Moreover, the results suggest that the LSTM neural network is not the most suitable learning algorithm for this context, achieving worse results than Random Forests and Logistic Regression (with or without the use of Bagging). Although the portfolios built using the Classifier's predictions were more profitable than the benchmarks, the increased Drawdown indicates a necessity for adequate portfolio risk optimization and risk management.

**Keywords**: Statistical arbitrage, Machine learning, Deep learning, Stock markets, LSTM, Finance, Algorithmic trading, Time series

**ACM CCS**: Mathematics of computing~Time series analysis (#10003693), Computing methodologies~Neural networks (#10010294), Applied computing~Economics (#10010460)

# Resumo

Na gestão de portfólio de investimentos, os gestores estão constantemente à procura de instrumentos (e.g. stocks) que sejam consistentemente dos mais rentáveis a longo prazo. Uma possibilidade de melhorar a pesquisa de stocks com um performance acima da média, é através do uso de técnicas de Machine Learning preditivas. Estudos anteriores tentaram prever alpha, i.e. retornos acima da média, através do desenvolvimento de modelos lineares baseados em anomalias nos mercados de capitais.

No entanto, as técnicas usadas nestes estudos carecem da capacidade de capturar dependências não lineares complexas. Mais recentemente, estudos sugerem algumas técnicas de machine learning poderão ser capazes de capturar estes fenómenos não lineares [1], nomeadamente as redes Long Short-Term Memory (LSTM). As redes LSTM pertencem à classe de Redes Neuronais Recorrentes (RNN) e são usadas na captura de padrões em dados sequenciais. Apesar do progresso tecnológico recente em Deep Learning dos últimos anos, pouca investigação foi ainda feita na análise e identificação de oportunidades de investimentos através de técnicas preditivas de Deep Learning. A maior parte da investigação disponível foca-se principalmente em previsões a curto prazo, e apenas faz uso de dados de mercado, desprezando outras informações publicamente disponíveis, como dados fundamentais.

Fazendo uso de demonstrações financeiras e preços de mercado de empresas listada na NASDAQ Stock Exchange e na New York Stock Exchange (NYSE) durante um período de cerca de 30 anos, diferentes features (e.g. Book-to-Market ratio, Return On Equity) e diferentes técnicas de Machine Learning foram testadas, nomeadamente as redes neuronais LSTM.

Como baseline, foi usada uma estratégia de investimento Buy-and-Hold, que é compreendida por comprar stocks no início do período de estudo, e apenas vender no fim do período de estudo. Todos os modelos desenvolvidos foram testados, simulando trades num período fora de amostra. Finalmente, os modelos foram comparados usando Accuracy, Year-over-Year Returns e Maximum Drawdown, para o período de teste. Os modelos testados provaram atingir medidas de Accuracy maiores que o Classificador Naive. Ademais, os resultados sugerem que a rede neural LSTM não seja o algoritmo de aprendizagem mais apropriado, para o contexto do problema, obtendo piores resultados do que Random Forests e Logistic Regression (com ou sem a utilização de Bagging). Apesar dos portefólios construídos serem mais rentáveis que as benchmarks usada, o elevado Drawdown indica a necessidade de optimizar o risco de portfólio e de gestão de risco.

**Keywords**: Statistical arbitrage, Machine learning, Deep learning, Stock markets, LSTM, Finance, Algorithmic trading, Time series

**ACM CCS**: Mathematics of computing~Time series analysis (#10003693), Computing methodologies~Neural networks (#10010294), Applied computing~Economics (#10010460)

# Acknowledgements

The journey that led me to this result was a steep one. Fortunately, I could count on a few people that made the difference.

Firstly, I am grateful for my parents, who have supported me for over 2 decades, and gave me everything I could possibly ever ask from a parent.

Thank you to Professor João Moreira who guided me towards the right path, every Wednesday, and invested his time into hearing my foolish and naive theories! I don't imagine I could count on a better Supervisor!

Thank you to Professor Luís Teixeira for his support towards improving my understanding of Deep Learning.

Last but by no means least, thank you to BEST Porto and all its amazing members... the impact this organization had in shaping my vision and life mission was crucial. Without them, I most certainly would not be the same person.

Thank you all for your encouragement and support! Hopefully, someday I'll be able to give back as much as you've given me.

José Diogo Seca

*"Truth - more precisely, an accurate understanding of reality - is the essential foundation for producing good outcomes."*

Ray Dalio

# Contents

# Chapter 1

# Introduction

## 1.1 Context

The financial markets serve important functions in our daily lives, even if unnoticed. Financial markets provide a medium for businesses, governments, and banks to trade in financial instruments. Such an example is the stock market, which provides the necessary liquidity for participants to make transactions in companies' shares.

In the business of portfolio management, picking the right group of stocks is crucial. A good pick could result in increased growth of the portfolio. A bad pick could result in large losses. Therefore, portfolio managers seek to invest in stocks that outperform all others at a decreased risk to the overall portfolio. However, this does not seem to be an easy task since not many portfolio management firms are able to showcase a consistent track record of profitable investment decisions.

The problem of attempting to predict asset prices movements, such as stocks, bonds or commodities, is not a new one. However, it is still relevant, especially to portfolio managers who wish to outperform the market, with the lowest risk possible.

## 1.2 Motivation

Portfolio managers look for the best investment strategies. A good strategy should be systematically more profitable than most strategies, limited in exposure to risk, and reproducible along with different assets and different time periods.

If a manager were to be assisted by a good predictive model, perhaps this would increase his ability to beat the market, and achieve better results for his investors. Given the recent advances in Machine Learning, and in particular Recurrent Neural Networks applied to financial time series, perhaps a model that fulfills this task could be produced.

## 1.3 Goals

The main goal of this dissertation is develop a Classifier capable of accurately predicting whether a given company's share price is going to outperform the average stock or underperform the average stock, 1 year ahead.

The secondary goal of this dissertation is to test the feasibility of building an investment portfolio based on the previous Classifier's predictions.

## 1.4 Outline

The rest of the paper is organized as follows. Chapter 2 describes the financial background. Chapter 3 presents Machine Learning and discusses the state of the art in Recurrent Neural Networks applied to financial time series. Chapter 4 describes the methodology. Chapter 5 presents the experiments' results and discusses them. Chapter 6 concludes this work and suggests improvements and guidelines for future work.

# Chapter 2

# Financial Background

Modeling financial data can be a challenge, especially if coupled with ignorance about the properties of financial data. In this chapter, we delve into the financial background to better understand the context of the problem.

## 2.1 Types of Financial Data

### 2.1.1 Fundamental Data

Fundamentals, i.e. fundamental data, are financial data composed of qualitative and quantitative information that affects the evaluation of a company or security [2]. Profitability, revenue, assets, liabilities and growth potential, are some examples of fundamentals. Analysts and investors examine these fundamentals in order to evaluate whether a company's stock is correctly underpriced or overpriced. This type of information can be found in financial statements, i.e. the income statement, the balance sheet, and the cash flow statement. Furthermore, in regards to stocks of publicly traded companies, this type of information is released: quarterly and reports the results for the previous business quarter; and yearly and reports the results for the previous business year. However, it is important to confirm when the data was released. Oftentimes fundamental data is indexed with the last date of the quarter, which oftentimes precedes the date of the release by 1.5 months [3]. Throughout this paper, we focus primarily on fundamentals and ratios obtained from fundamentals, using a yearly timeframe.

### 2.1.2 Market Data

Market Data includes price, volume, dividends, shares outstanding, open interest, among others. Technicals or technical indicators are data derived primarily from historical price movements. Most studies on Deep Learning and financial markets focus primarily on features such as Market Data.

### 2.1.3 Other types of Data

We will not make use of data such as analysis recommendations and expectations, news and microblogging sentiment, satellite imaging, among others. However, these datasets may help improve accuracy.

## 2.2 Efficient Market Hypothesis

The Efficient Market Hypothesis (EMH) [4] states that prices rapidly reflect all the available information. An implication from this is that it is impossible to "beat the market" consistently on a risk-adjusted basis since market prices should only react to new information.

There are 3 forms of the EMH which specify how to interpret "all the available information". Specifically, the three variants of the EMH are the weak form, the semi-strong form, and the strong form.

### 2.2.1 Weak form of Efficiency

The weak form of EMH asserts that stock prices already reflect all information that can be derived by examining market trading data such as the history of past prices, trading volume, or short interest.

### 2.2.2 Semi-strong form of Efficiency

The semi-strong form of EMH states that all publicly available information regarding a firm must be reflected in the stock price. Such information includes, in addition to past prices, fundamental data such as financial statements, management quality.

Evidence that refutes the weak form of the EMH also refutes the semi-strong form of EMH.

### 2.2.3 Strong form of Efficiency

The strong form of EMH states the current price reflects all information, public as well as private. This private information is only available to the insiders, i.e. stakeholders with privileged access to information.

According to the strong form of EMH, insider trading would not be possible because there would be no chance for an insider to profit off the market. Any new information, including private information, would be immediately priced into the asset. However, it is unlikely that the market follows the strong form of EMH, given that insider trading still happens [5].

Evidence that refutes the weak form or the semi-strong form of the EMH also refutes the strong form of EMH.

## 2.3 Behavioural Finance

Contrasting with the Efficient Markets Hypothesis, Behavioral finance studies how people fall short of this ideal in their decisions and how markets are, to some degree, inefficient [6].

Behavioral finance proposes psychology-based theories to explain stock market anomalies, such as severe rises or falls in stock price. The purpose is to identify and understand why people make certain financial choices.

[7] provides surveys comprising more than 100 of such capital market anomalies, which rely on return predictive signals to outperform the market. However, according to [1], the financial models used to establish a relationship between these return predictive signals, and future returns are usually transparent in nature and not able to capture complex non-linear dependencies.

### 2.3.1 Adaptive Market Hypothesis

The Adaptive Market Hypothesis (AMH) can be viewed as a new version of the EMH, derived from evolutionary principles. The AMH states that prices reflect the irrational behavior of investors. This behavior is influenced by the combination of environmental conditions and other market participants [8].

According to this hypothesis, arbitrage opportunities exist. The goal of arbitraging it to profit from mispricing. A similar but broader definition of arbitrage is statistical arbitrage. While arbitrage is the simultaneous purchase and sale of a single asset, statistical arbitrage is neither limited to two securities nor limited to two transactions. Moreover, transactions can be played-out by days, months or years.

Because the relationship between risk and reward is constantly changing over time, an adaptive strategy can thus be deemed better to achieve a consistent level of expected returns. Various conditions may affect market efficiency, allowing for an increased prediction ability in different time-periods.

## 2.4 Metrics for investors

### 2.4.1 Returns

The problem of attempting to predict asset prices movements, such as stocks, bonds, and commodities, is not a new one. However, it is still relevant, especially to portfolio managers who wish to outperform the market, with the lowest risk possible. The measure of performance is typically year-to-year returns. Returns are calculated as described in Equation (2.1).

$$r_{ij} = \frac{p_j - p_i}{p_i} \tag{2.1}$$

**where:**

$r_{ij}$ is the return from time $i$ to time $j$;

$p_i$ and $p_j$ are the prices of a given asset at times $i$ and $j$ respectively.

The benefit of using returns over prices is the normalization. For the purpose of building a model, returns are assumed to be stationary, whereas prices cannot. This also allows the objective comparison of returns between different stock.

Logarithmic returns (log-returns) are calculated as described in Equation (2.2):

$$lr_{ij} = log(r_{ij} + 1) \tag{2.2}$$

or as described in Equation (2.3):

$$lr_{ij} = log(\frac{p_j}{p_i}) \tag{2.3}$$

If prices are assumed to be log-normally distributed, then logarithmic returns are normally distributed. Although this may be a false assumption, it allows more flexibility in the application of statistical techniques. In comparison with simple returns, when logarithmic returns are used, a higher variance will automatically reduce expected returns [9].

However, in the context of investigations into the terminal wealth of investors, it is more appropriate to use the measure of simple returns [9].

### 2.4.2 Risk

A commonly used measure of risk is the standard deviation of the historical daily returns. This can be computed the following way for a stock $s$ as described in Equation (2.4):

$$\sigma_s = \sqrt{\frac{\sum_{i=1}^{N}(r_{is} - \bar{r}_s)^2}{N-1}} \tag{2.4}$$

This can be expanded and applied to the performance of a portfolio, which composes different stocks with different weights respectively. These weights signify the percentages of portfolio value. The portfolio standard deviation can be computed as described in Equation (2.5):

$$\sigma_{port} = \sqrt{\sum_i \sum_j w_i w_j \sigma_i \sigma_j \rho_{ij}} \tag{2.5}$$

where $w_i$ and $w_j$ are weights for stocks $i$ and $j$ respectively, where $\rho_{ij}$ is the Pearson correlation between the stock returns $i$ and $j$, and where $\rho_{ij} = 1$ for $i = j$.

Alternatively, this expression can be written as described in Equation (2.6):

$$\sigma_{port} = W \; K_{XX} \; W^T \tag{2.6}$$

where $W$ is the vector of weights, and the $K_{XX}$ is the covariance of returns matrix.

Another metric which can be useful to the analysis of portfolio risk is the Maximum Drawdown, which measures the relative loss from a peak to a trough of a portfolio, before a new peak

is attained. Drawdown is described in Equation (2.7):

$$\text{Drawdown} = \frac{\text{Trough value} - \text{Peak value}}{\text{Peak value}} \tag{2.7}$$

The lowest the drawdown of a portfolio is, the more stable and reliable the investment is expected to be. Furthermore, a portfolio with a low drawdown could be carefully leveraged in order to magnify the returns of the overall portfolio, achieving a better performance.

A common benchmark used to compare equity investments is the Standard & Poor's 500 index (S&P 500). The index serves as a proxy for overall market returns. The investment can be compared in two ways. Firstly, by looking at the correlation between the investment and the market, which measures how frequently investment returns can be explained by market moves. Secondly, by looking at the linear regression coefficient between the investment returns and market returns. The regression attempts to find the intercept, also known as Alpha, and the regression coefficient, also known as Beta. The regression can be described in the Equation (2.8):

$$R_{i,t} - R_{rf} = \alpha_i + \beta_i(R_{m,t} - R_{rf}) + \varepsilon \tag{2.8}$$

**Whereas:**

$R_{i,t}$ is the return of the investment $i$ for period $t$;

$R_{m,t}$ is the return of the market $m$ for period $t$;

$R_{rf}$ is the risk-free rate, generally set at 2% annually;

$\beta_i$ is the investment's $i$ sensitivity to market moves;

$\alpha_i$ is the return of the investment $i$ not explained by market moves;

$\varepsilon$ is the error to be minimized during the regression.

# Chapter 3

# Machine Learning Background

In this chapter, we will briefly summarize the relevant Machine Learning techniques that relate to this study, including Deep Learning. In chapter 4 we will continue this discussion by describing how these components were applied, and in chapter 5 we study how they affected the results.

## 3.1 Naive Bayes

Naive Bayes (NB) is a probabilistic machine learning algorithm based on the Bayes theorem. The Bayesian theorem can be described by Equation (3.1):

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} \tag{3.1}$$

Naive Bayes assumes that the explanatory variables are independent variables, which may not be the case. Naive Bayes tends to be computationally fast and generalize well when using high dimensional data. However, it will tend to perform poorly when the independence assumption is not met.

A special implementation of Naive Bayes is Gaussian Naive Bayes, which assumes the explanatory variables follow normal distributions.

## 3.2 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is an instance-based machine learning algorithm. An instance is classified by a plurality vote of its neighbors, with the instance being assigned to the class most common among its $K$ nearest neighbors. If $K = 1$, then the object is simply assigned to the class of that single nearest neighbor. KNN is a non-parametric, due to not parameters of a model, and

is often called ïazy" due to memorizing the training instances, and only using them to generalize during predictions. A commonly used distance metric for continuous variables is Euclidean distance.

## 3.3 Logistic Regression

In the context of Machine Learning, Logistic regression is a learning algorithm used for binary classification problems (problems with two class values). The logistic regression name comes from the use of the logistic function, also known as sigmoid function. The sigmoid function can be described by Equation (3.2):

$$\frac{1}{1+e^{-x}} \tag{3.2}$$

This function maps any real value into a number between 0 and 1, as seen in Figure 3.1 When used to make binary predictions, we use the decision bound of 0.5. If above 0.5, then the prediction is rounded up to 1; otherwise, it is rounded down to 0.

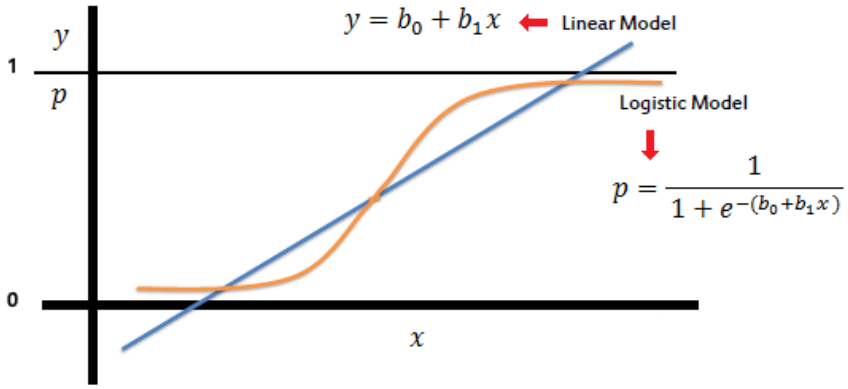


Figure 3.1: Logistic Regression compared to Linear Regression

Source: http://juangabrielgomila.com/en/logistic-regression-derivation

Logistic Regression tends to have a high bias. It also tends to work well with diagonal decision boundaries.

## 3.4 Bagging

Bagging, i.e. Bootstrap aggregating, is an ensemble learning technique designed to improve the accuracy of machine learning algorithms. It reduces variance, hence helping to avoid overfitting. Inevitably, it also leads to an increase in bias.

Bagging draws several bootstrap samples from the original training set and generates multiple versions of a predictor. Afterward, it uses these to get an aggregated predictor. This technique was proposed by Leo Breiman, and is generally associated with and applied to decision tree learning algorithms. [10]. Nevertheless, it can be applied to other learning algorithms.

When samples of the dataset are drawn as random subsets of the features, then the method is known as Random Subspaces [11].

## 3.5 Random Forest

Random Forest (RF) is an learning algorithm based on the application of the Random Subspaces method to Decision Trees. RF fits several decision tree classifiers on various sub-samples of the dataset. Furthermore, it uses averaging to improve the predictive accuracy and control over-fitting. RF usually results in reduced variance relative to a Decision Tree learning algorithm. However, RF are not a easily interpretable.

## 3.6 PCA Whitening

Principal component analysis (PCA), pioneered by Karl Pearson [12] is a pre-processing procedure that linearly combines features in order to obtain new features that are linearly uncorrelated, called principal components. Furthermore, PCA attempts to retain as much variance in the data as possible. It works firstly by finding a component that maximizes variance and then finding other components that also maximize variance but are orthogonal.

PCA is useful in machine learning in transforming correlated features into uncorrelated components and in reducing dimensionality. This procedure is sensitive to the scaling of the data, therefore, we must pay attention to standardize our features before applying the PCA transformation.

Furthermore, after applying PCA, the first components are likely to have larger scales than the last components. This may influence the training process. One improvement to this is to whiten the components, i.e. to standardize them, so that the resulting features have 0 and 1 as average and variance respectively.

PCA features tend to be harder to interpret than the original features.

## 3.7 Deep Learning

### 3.7.1 Multilayer Perceptron

Artificial neural networks are an approximation of the behavior of biological networks of neurons. Artificial neural networks are composed of interconnected layers of artificial neurons. Each neuron can be seen as a node, and each connection can be seen as a weighted edge. The network learns by increasing the weights of some edges while decreasing the weights of other edges.

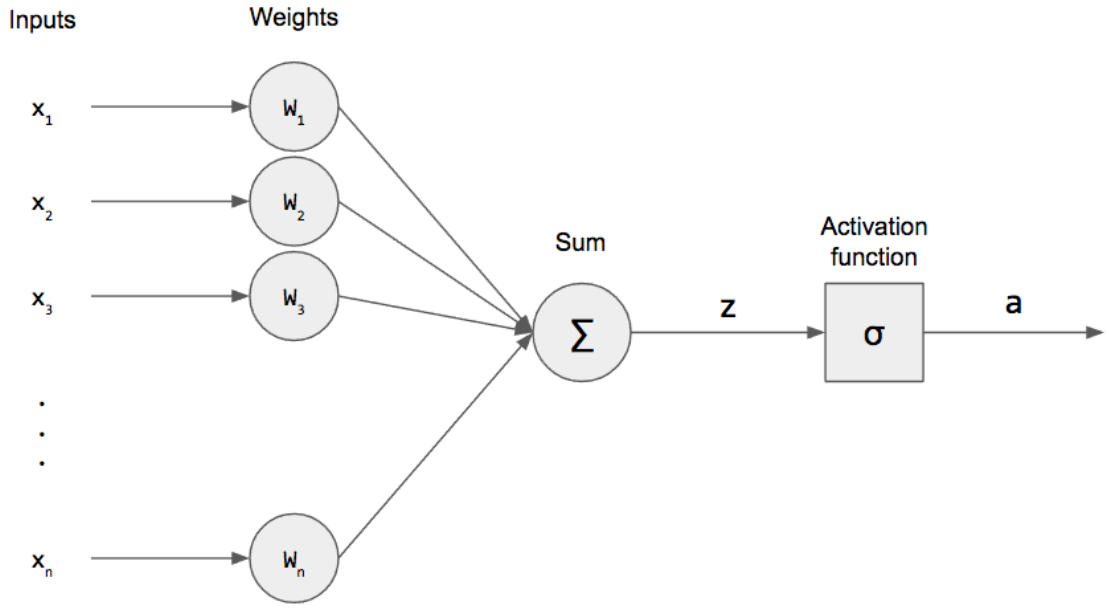


Figure 3.2: The perceptron

Source: https://pythonmachinelearning.pro/perceptrons-the-first-neural-networks

One of the most common neurons used in neural networks is the perceptron. The perceptron, as seen in Figure 3.2 takes an input vector $X$, multiplies it by a vector of weights $W$ and returns an output $Y$. If the dot product is above 0, the neuron returns 1, and 0 otherwise. This last operation is called the activation function.

The perceptron can then be used in a larger network, composed of many neurons. One such example is the Multilayer Perceptron (MLP). The MLP, as seen in Figure 3.3 is formed by a hidden layer of many perceptrons. Every perceptron in this layer takes $X$ as input, multiplies it by its own weight vector $W$, and after applying the activation function, returns its output to the neurons in the following layer. In this case, the following layer is the output layer which is composed of one neuron.

However, more commonly nowadays are neural networks with more than 1 hidden layer. In these types of networks, the hidden neural networks receive their inputs from the previous layer, which may not necessarily be the input layer. Moreover, the output is passed to the subsequent layer which may be another hidden layer or the output layer.

Training a neural network is composed of 3 different steps: 1) the input is passed through the hidden layers, ending in the output layer where a prediction is made (forward pass): 2) this prediction is compared to the target, using a cost function, which outputs the error; 3) this error is back-propagated from the output layer to the preceding layers, successively updating the weights of every neuron, using their respective gradient of the cost function. The gradient is a vector which points in the direction that minimizes the error of the cost function.

Deep learning is an umbrella term typically used to define complex neural networks, which

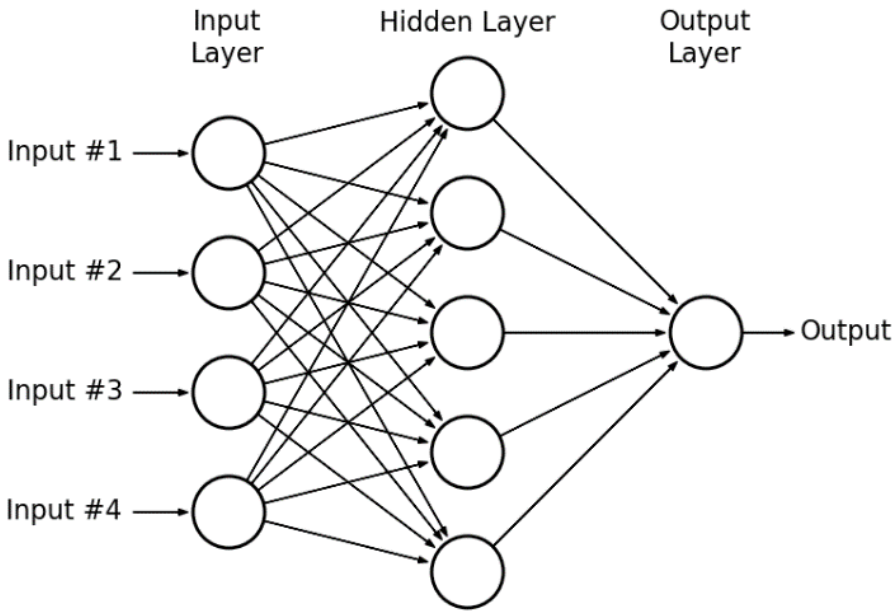


Figure 3.3: The multilayer perceptron (MLP) neural network

Source: [13]

have multiple hidden layers, complex neurons, and architectures. Deep learning's inner mechanisms are more complex, they also tend to be more abstract than simpler neural networks.

### 3.7.2 Activation functions

The activation function is the last operation at the end of a neuron. It determines whether the neuron gets fired, and how intensely it gets fired. Previously, we have seen that the activation function of the perceptron is a binary step function, which either returns 1 or 0 depending on the output of the previous operation. However, there are many more functions, some of which will be described in this section.

The Sigmoid activation function outputs values between 0 and 1, is monotonic, lacks a monotonic derivative and doesn't approximate the identity function near the origin. This activation function can described by Equation (3.3):

$$f(x) = \sigma(x) = \frac{1}{1+e^{-x}} \tag{3.3}$$

The hyperbolic tangent (TanH) activation function outputs values between -1 and 1, is monotonic, lacks a monotonic derivative and approximates the identity function near the origin. This activation function can described by Equation (3.3):

$$f(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})} \tag{3.4}$$

The Rectified Linear Unit (ReLU) [14] activation function outputs values between 0 and $+\infty$, is monotonic, has a monotonic derivative and doesn't approximates the identity function near the origin for negative values. This activation function can described by Equation (3.5):

$$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} \tag{3.5}$$

ReLU has shown a better and more efficient training of deep networks, compared to activation function such as Sigmoid and hyperbolic tangent [15] [16].

### 3.7.3 Recurrent neural networks

The previously shown networks were feed-forward neural networks. In feed-forward neural networks, the information flows only in one direction, from the input layer to the subsequent layers, and finally reaching the output layer.

Recurrent neural networks (RNN), on the other hand, have the presence of loops, as seen in Figure 3.4. RNN can have a memory about past values, and use this information in the calculation of present values. This past information is memorized in a hidden state. For each iteration, the RNN updates its hidden state. This is particularly useful in modeling sequential data such as time series, text or speech.

Many studies have now reported positive results on financial timeseries using variants of RNNs, including [17], [18], [19], [20], [21] and [22] in RNNs, [23] in Gated Recurrent Unit (GRU) networks, and [24] in a hybrid Long Short-Term Memory (LSTM) network with Generalized Autoregressive Conditional Heteroskedasticity (GARCH).

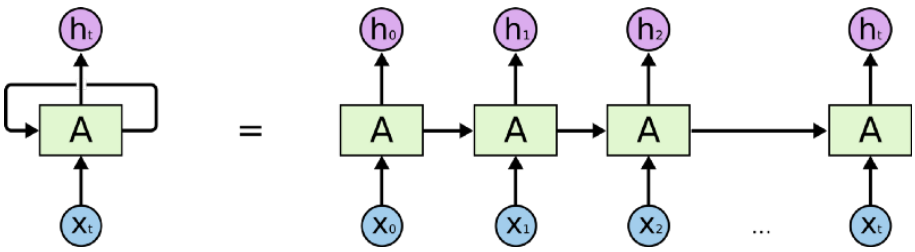


Figure 3.4: A unit of a RNN. On left we have the abstraction of the loop. On the right we have the unrolled version of the unit. The output of the previous timesteps are kept and used to calculate the unit's current output

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs

During back-propagation, the simple RNNS suffer from the problem of vanishing gradients. The vanishing gradients problem occurs when the gradient is reduced too quickly. This problem tends to happen in RNNs, specifically in the earlier timesteps of an RNN unit. As the RNN processes more timesteps, it has difficulty in remembering and retaining information from the earlier timesteps. In a sequence of instances, the instances in the earlier portion of a sequence

have much more importance than instances in the latter portion of a sequence, in terms of updating weights. Simple RNNs are good at learning short-term dependencies in sequential datasets, but bad at learning long-term dependencies in sequential data [25].

Two common variants of RNN that attempt to solve this problem are the Long Short-Term Memory (LSTM) networks, and the Gated Recurrent Unit (GRU) networks. Both these networks derive their respective name from the name of the neuron units present in the architecture, i.e. the LSTM unit and the GRU.

### 3.7.4 Long Short-Term Memory

The Long Short-Term Memory (LSTM) networks were originally proposed by [26], refined in [27] and carefully reviewed and popularized much later in [28]. The LSTM units present in these networks are composed by a cell, an input gate, and an output gate and a forget gate. The cell is where the hidden state of the LSTM unit is stored, and where the calculations with regards to the LSTM occur. The input, output and forget gates regulate the flow of information within the cell. These gates learn which portions of information in the sequence of inputs are relevant and should, therefore, be kept, or are irrelevant and should, therefore, be forgotten. The LSTM is depicted in Figure 3.5.

LSTMs are more prone than simple RNNs to learn dependencies between important values separated by an unknown duration.

However, the learning rate should still be chosen carefully as the LSTM still suffers from the exploding gradient problem. According to [28] the "learning rate and network size are the most crucial tunable LSTM hyperparameters".

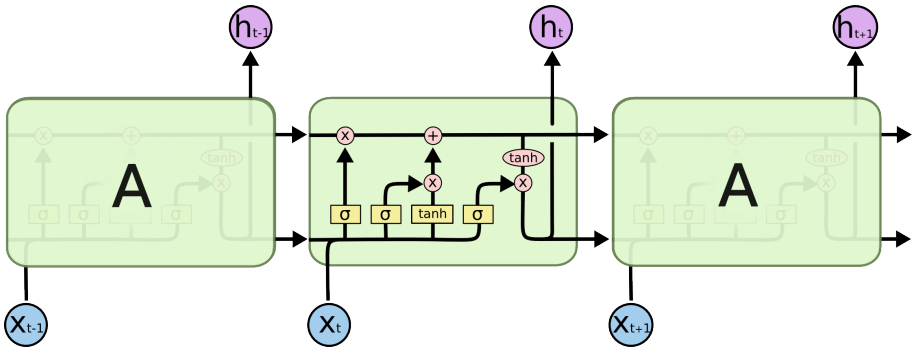


Figure 3.5: A LSTM unit, depicting its respective gates and flow of information

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs

The LSTM gates [26] [27] can be defined as follows, by Equations (3.6), (3.7), (3.8), (3.9), and (3.10):

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{3.6}$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{3.7}$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{3.8}$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \tag{3.9}$$
$$h_t = o_t \circ \sigma_h(c_t) \tag{3.10}$$

where the initial values are $c_0 = 0$ and $h_0 = 0$ and the operator $\circ$ denotes the Hadamard product (element-wise product). The subscript $t$ indexes the time step.

$x_t \in \mathbb{R}^d$: input vector to the LSTM unit

$f_t \in \mathbb{R}^h$: forget gate's activation vector

$i_t \in \mathbb{R}^h$: input gate's activation vector

$o_t \in \mathbb{R}^h$: output gate's activation vector

$h_t \in \mathbb{R}^h$: hidden state vector also known as output vector of the LSTM unit

$c_t \in \mathbb{R}^h$: cell state vector

$W \in \mathbb{R}^{h \times d}$, $U \in \mathbb{R}^{h \times h}$ and $b \in \mathbb{R}^h$: weight matrices and bias vector parameters which need to be learned during training

where the superscripts $d$ and $h$ refer to the number of input features and number of hidden units, respectively.

$\sigma_g$: sigmoid function

$\sigma_c$ and $\sigma_h$: hyperbolic tangent functions

The Gated Recurrent Unit (GRU) is a simplified and more efficient version of the LSTM proposed by [29]. The GRU merges the forget and input gates into an update gate. It also merges the cell state and hidden state. Some authors argue that the LSTM and the GRU are similar in performance to one-another [30] while others find that the GRU results in bigger errors [31].

[28] show that none of the variants of LSTM network can improve upon the standard LSTM architecture significantly. One explanation is the explicit internal memory and the greater gate-centric control in updating the LSTM [32]. In the context of financial time series, LSTMs also tend to outperform simple networks in terms of predicting price movements and maximizing Sharpe ratio [33].

As an improvement to the LSTM, [34] recommend "adding a bias of 1 to the forget gate of every LSTM in every application", as doing so significantly improves the performance of the LSTM on tasks where it falls behind the GRU.

### 3.7.5 Cost function

Before back-propagation starts, and the cost is used to update the parameters of the network, we must first be able to calculate the cost. The function that, given our predictions and our labels computes the cost, is called the cost function.

One commonly used cost function used for classification problems is cross entropy, also known as log loss. This can be measured as described in Equation(3.11):

$$H = -\sum_{c=1}^{C} y_{o,c} \log(p_{o,c}) \tag{3.11}$$

in which:

$$C = \text{number of classes} \tag{3.12}$$

$$o = \text{observation} \tag{3.13}$$

$$p = \text{predicted probability observation o is of class c} \tag{3.14}$$

$$y = \text{binary indicator (0 or 1) if class label c is the correct classification for observation o.} \tag{3.15}$$

For the special case of binary classification, the previous function can be described in Equation (3.16):

$$H = -y\log(p) - (1-y)\log(1-p) \tag{3.16}$$

### 3.7.6 Optimization algorithm

Gradient descent is by far the most popular way to optimize neural networks, with various implementations. Gradient descent is a way to minimize a neural network's cost function. This section describes the gradients which were tested during this dissertation. Only implementations of mini-batch gradients descent were considered, as they tend to avoid converging to local optima.

[35] argue that RMSprop, a mini-batch version of rprop [36], is a good choice for recurrent neural networks. Adam builds on RMSprop, by adding bias-correction and momentum. [37] show Adam slightly outperforming RMSprop towards the end of optimization as gradients become sparser. Some argue that the best overall choice of the gradient is Adam [38].

Another possibility is to use the more recent Nadam [39]. Nadam is identical to Adam, by building on RMSprop, by instead of Classical Momentum, it adds Nesterov Momentum.

## 3.8 Regularization techniques

Overfitting means fitting the model too closely to the training set. Overfitting leads to poor generalizations in out-of-sample tests. This section is dedicated to regularization techniques, i.e. mechanisms to reduce overfitting.

### 3.8.1 L2

Applying L2 regularization tends to shrink the extreme parameters, e.g. neuron weights, to a common level. L2 regularization adds a weight penalty term to the cost function, as described in Equation (3.17)

$$\text{Cost} = \text{Cost Function} + \lambda_2 \sum_{i=0}^{N} W_i^2 \tag{3.17}$$

### 3.8.2 L1

Applying L1 regularization tends to shrink the less important parameters, e.g. neuron weights, to zero. L1 regularization adds a weight penalty term to the cost function, as described in Equation (3.18)

$$\text{Cost} = \text{Cost Function} + \lambda_1 \sum_{i=0}^{N} |W_i| \tag{3.18}$$

The use of L1 regularization is a good option in the case of large feature space, as it tends to give more importance to a small group of features. However, a compromise between L1 and L2 regularization may be a preferred to simply L1 regularization, as it may behave erratically when the features are highly correlated or when handling high dimensional data [40].

### 3.8.3 Dropout

Dropout is a technique invented by [41] by preventing complex co-adaptations on the training set. The authors describe dropout as a "very efficient way of performing model averaging with neural networks". Dropout refers to the temporary partial drop of a layer in a neural network. Each neuron in the layer has a defined probability of being randomly omitted.

Regular dropout is typically applied at the inputs or outputs of a layer of neurons. Recurrent dropout, on the other hand, is applied between timesteps [42]. The comparison can be visually seen in Figure 3.6.

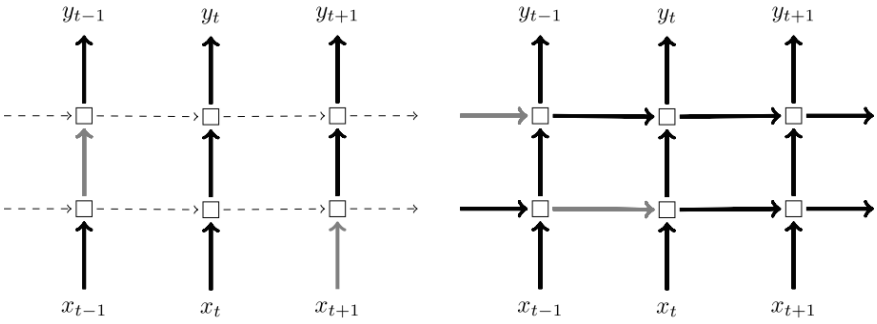


Figure 3.6: Regular dropout on the left. Recurrent dropout on the right

### 3.8.4 Early stopping

Early stopping consists of stopping the training when the error on the validation set stops improving after a certain number of iterations. Training past this point tends to improve the model's fit to the training data at the expense of increasing the generalization error.

Despite some calling this technique a "beautiful free lunch" [43], other researchers argue that generalization can be improved even when there is no apparent validation error change [44].

# Chapter 4

# Methodology

The main goal is to design a classifier, capable of accurately labelling out-of-sample instances, i.e stocks 1 year ahead, into either 1 or 0, depending on the direction of price movement predicted. The secondary goal is to build consistently profitable portfolios of stocks through the use of this classifier. This approach is different from previous studies of prediction of stocks' prices and price movements due to its use of fundamentals coupled with Machine Learning techniques. Specifically, the current literature is lacking on the topic of RNNs applied to stock price prediction, using fundamentals.

The problem is framed as a classification, and not as a regression. This is done purposefully, and with the intent of finding the classification probabilities given by the classifier. These probabilities then become useful when ranking stocks, and selecting the top 100 stocks most likely to outperform the standard.

Moreover, this study seeks to answer which features, built from fundamentals and market data, are most significant in terms of predicting the direction of price movements, and which leaning algorithms result in more accurate predictions.

Markets are traded using strategies developed by humans. These strategies may or may not be automated, but they are typically systematic, i.e. they form a pattern in behavior which can be captured and simulated by a machine. Even when there is human psychology at play, the irrational decisions on behalf of traders and portfolio managers could form individual patterns. Furthermore, if the market is seen as the aggregate of its participants, making transactions for the same thing, then the market can also be thought of as obeying a pattern, which is merely the sum of all patterns from its participants. If an investor is able to understand this pattern or, in the very least, a part of it, then he might be able to profit off of it. This study attempts to extract this pattern using an LSTM network. This architecture is chosen with the assumption that it's a better learning algorithm, capable of capturing dependencies in temporal datasets.

## 4.1 Data preparation

The fundamentals were extracted from Thomson Reuters Eikon [45]. This information consists of data from publicly-traded traded in the NASDAQ Exchange and in the New York Stock Exchange (NYSE), from 1983 to 2017. Because the number of dead companies in the database was suspiciously reduced, and very few dead companies were properly documented with financial statements, only active companies were extracted. Note that this procedure introduces survivorship bias in the sample.

For each company, the following data was fetched: Current Assets, Total Assets, Current Liabilities, Total Liabilities, Total Debt, Total Equity, Revenue, Gross Profit, Operating Income, EBIT, EBITDA, Net Income, Cash Flow, Market Cap, Price Close (year).

Adjusted close prices were extracted from Yahoo Finance for the same period, and used to calculate log-returns. Adjusted prices are the closing prices after adjustments for all applicable splits and dividend distributions [46]. The length of the time periods used to calculate the log-returns is 1 year. This period starts on the first day of April of year $t$ and ends on the first day of April of year $t+1$. The reason for this choice of the time period, i.e. from April-to-April and not December-to-December, is that it may take up to 90 days for publicly traded United States companies to release their 10-K financial statements [5]. Because of this, investors can only be sure to have access to this financial information by April. Attempting to use the previous months as the starting point for predictions could lead to look-ahead bias, as some of the information could have not yet been released, and therefore not yet be reflected into stock prices [3]. The 1st April cut-off is a heuristic, which cuts off most companies, but a handful of companies could still further delay the release of their financial statements.

Furthermore, prices of the Standard & Poor's 500 index were also extracted from Yahoo Finance.

### 4.1.1 Fundamentals ratios

The fundamentals extracted from Eikon had multiple issues, namely missing values, different scales and wrong values (e.g. Revenue being 0 while Net Income was positive). The instances with missing values and wrong values were dropped.

The problem with different scales is that we are analyzing different companies, belonging to different sectors and industries. Comparing assets of a company belonging to the financial industry, with the assets of a company that belongs to the manufacturing industry is probably a bad idea. To correct the problem of different scales, we use the fundamentals as inputs and combine them linearly, obtaining fundamental ratios. The following ratios were computed: Book to Market, Return on Equity, Leverage ratio, Debt to Equity, Liabilities to Equity, Long-term Liabilities to Equity, Earning Power, Net profit margin, Debt ratio, Price Earnings ratio, Working Capital to Fixed Assets, Return on Assets, Cashflow to Earnings, Cashflow to Equity, Current ratio, Gross profit margin, Asset Turnover, Earnings to Current Liabilities, Current Assets to Total Liabilities, Current Liabilities to Total Assets. Most of these ratios are Debt ratios, Liquidity

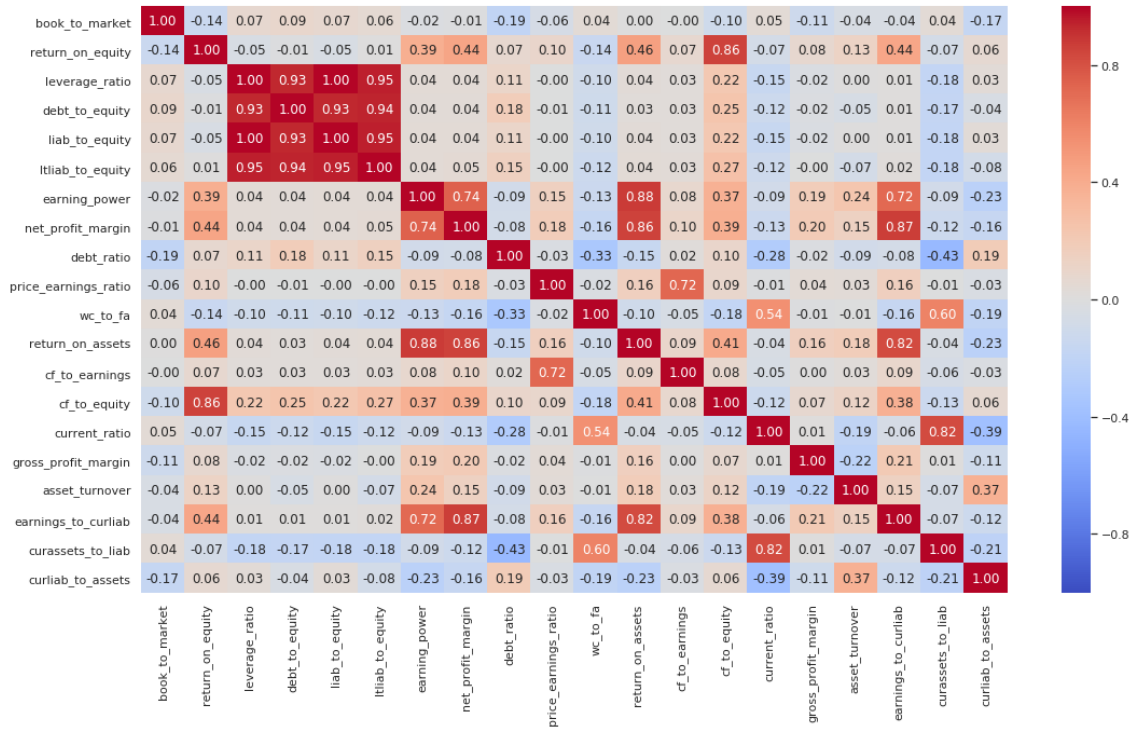


Figure 4.1: Correlation matrix of fundamental ratios

ratios or Profitability ratios. These ratios partially solve the problem of differing scales, as they can be used to compare different companies, even from different industries. However, the resultant dataset of ratios was contaminated with missing values and infinite values, due to zero divisions. Missing values, which occur in the case of a 0 by 0 division, were replaced by 0. Infinite values and extreme values, due to null denominators and very small denominators respectively, were bounded to 10*IQR (Inter-Quartile Range) from the median. Values that fell outside of this range were replaced by 10*IQR if above this range, or by -10*IQR if below this range.

As seen in Figure 4.1, some of these features are highly correlated. Training a model using a dataset with highly correlated features is typically undesirable.

### 4.1.2 Returns statistics

Calculated from the daily close prices, daily log-returns were analyzed and aggregated yearly. For each company and year, the following statistics were extracted: mean, standard deviation, skewness, kurtosis. These were used as features.

### 4.1.3 Labels

To construct our labels, the direction of price movements, yearly log-returns were used.

Because the returns are not normally distributed Figure 4.2, if we were to label instances solely on the future returns being positive or negative, we would end up with many more instances with

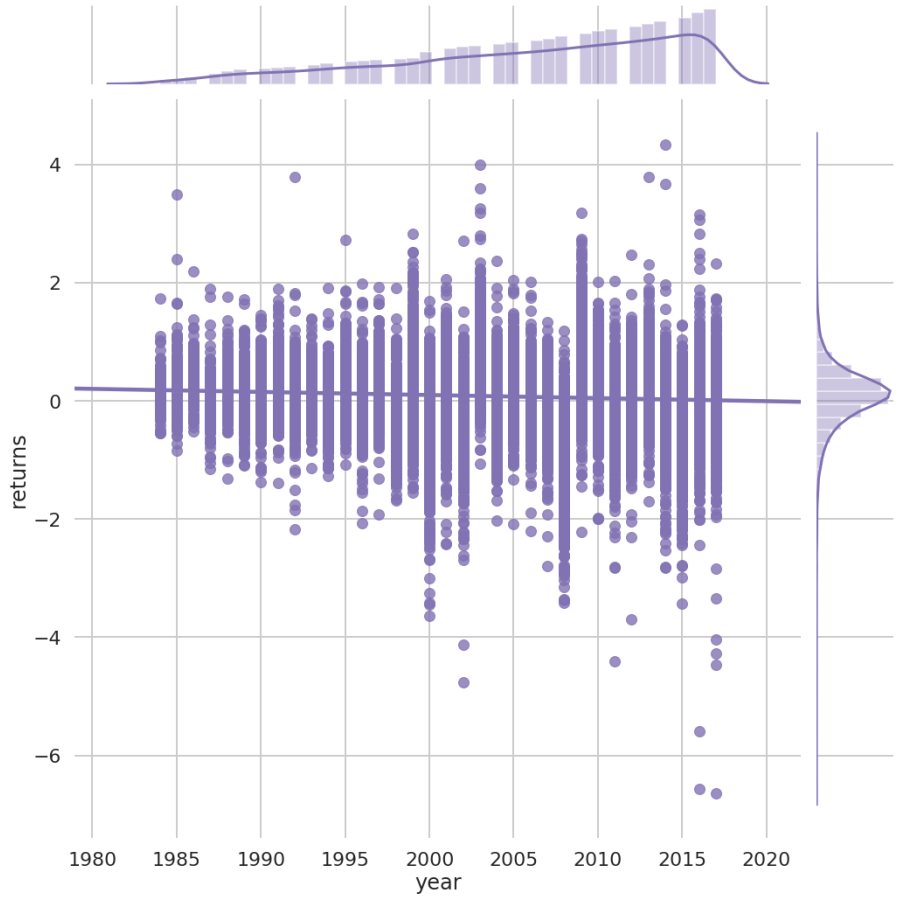


Figure 4.2: Distributions of daily returns per years.

positive returns. Instead, we first calculate the average log-return per year. Secondly, we use that average to find whether a given company over-performed or under-performed. An instance whose returns over-performed the average return for that year will be labeled as 1, and 0 otherwise. This way, per year, we will have approximately 50% actual positives and approximately 50% actual negatives, and therefore a balanced sample.

### 4.1.4 Resultant dataset

The resultant 25 features, composed of fundamentals ratios and returns statistics were then transformed by applying PCA whitening. This procedure yielded 20 orthogonal features while retaining 99% of the variance in the original data.

Every instance is indexed by the pair [company, year]. When learning a model, data from several companies will be fed to the learning algorithm at the same time, without regard for the company industry or sector.

By pre-processing the original features, and adjusting the features to the labels, some instances were eliminated. Only instances between the year 1984 and year 2017 were kept, totaling 35013

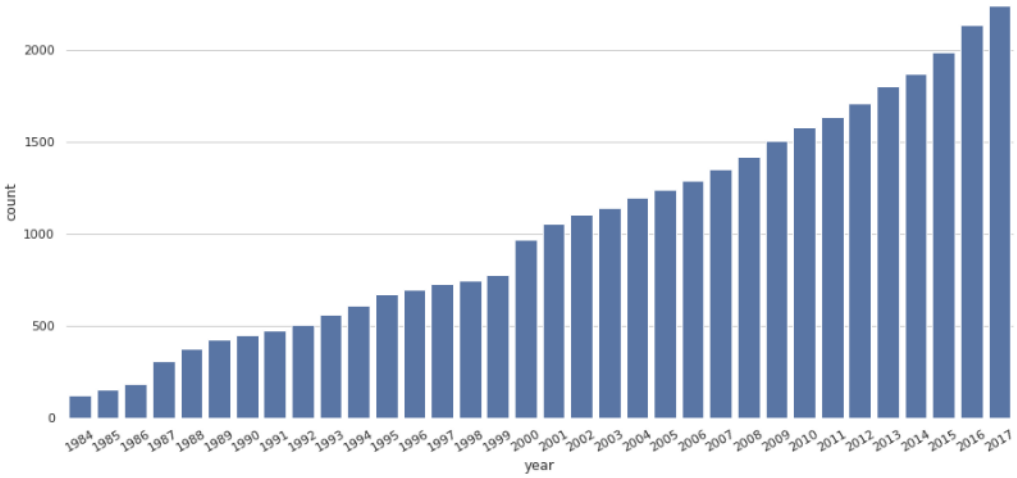


Figure 4.3: Distribution of instances per year. The horizontal axis shows the years, and the vertical axis shows the amount of companies available in the dataset for that same year. The distribution indicates selection bias towards recent year.

labeled instances. The last 2 years, comprising of 4373 instances are exclusively used for out-of-sample predictions.

Overall, the amount of companies per year is greatly skewed, as seen in Figure 4.3. This is mostly explained by the survivorship bias present in the extracted sample.

## 4.2 Baseline Classifiers

Logistic Regression was tested using the implementation of the Python machine learning library Scikit-learn [47]. This implementation includes an option for l1 regularization, which was used during the training of the model.

To train, validation and test the model, the method of the sliding window was used, with a minimum window size of 5 years and a maximum window size of 20 years, as depicted in Figure 4.4. Starting at the top, we train the model in the first 5 years of data and validate the model on the 6th year. Next, we use the first 6 years of data and validate the model on the 7th. We repeat this process until reaching the end of the dataset. Finally, the metrics of the model are calculated excluding the last 2 years, which are left exclusively for out-of-sampling testing. In total, this method processes a development set of 30 years, in which 25 years are used to validate the model and optimize hyperparameters.

## 4.3 LSTM Classifier

The neural network architecture chosen has 3 hidden layers: 2 LSTM layers and 1 fully-connected layer. The output layer consists of a single neuron with a sigmoid activation function, with output between 0 and 1.

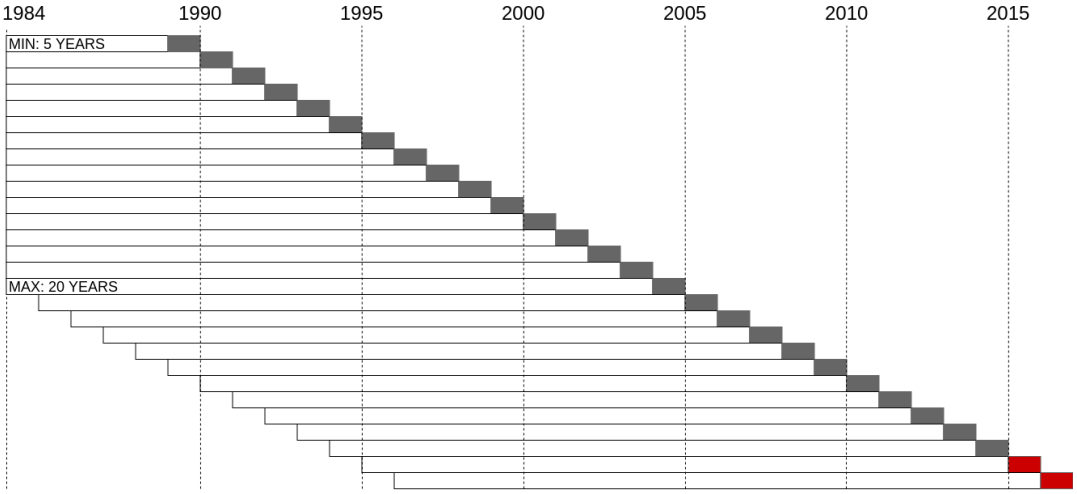


Figure 4.4: Sliding window method. The white bars represent the training instances, the gray squares represent the validation instances, and the red squares represent the out-of-sample testing instances. Every training set, the model is retrained and re-evaluated on the validation set.

LSTMs can either be stateful or stateless. In the case of a stateful LSTM, the model requires the instances to be sorted according to their original sequence. If we were doing this for just 1 stock, we would simply introduce an instance to the LSTM, train it, and then repeat the process with the next instance. However, since we're not trying to predict the direction of the price of one single stock, but many stocks, we would have to constantly be changing or interrupting the state, which defeats the purpose of using a stateful LSTM in the first place. The obvious alternative is to use a stateless LSTM. However, doing so requires us to feed the model several timesteps, every time.

The dataset used for the Logistic Regression only had 1 timestep per instance, e.g. the instance [AAPL,2005] only had features related from 2004-04-01 to 2005-04-01. On another hand, the LSTM requires several timesteps, e.g. 5 timesteps for the instance [AAPL,2005] would result in a matrix, with dimensions 5x20 (timesteps x features) consisting of yearly features from 2000-04-01 to 2005-04-01. In some cases, because the companies were fairly recent, no data was available for previous timesteps. Because of such cases, an extra feature was created. In the cases where the data for a given timestep would be missing, the extra feature would be set to 0; and 1 otherwise. The number of timesteps was set constant to 10.

Similarly to the Logistic Regression model, the method of the sliding window was used with a minimum window size of 5 years. However, because neural networks are computationally expensive and notoriously challenging to train, the maximum window size was reduced to 10 years. There is no overlap between the different validation or test periods [48].

Different architectures were implemented and tested using Keras [35] and Tensorflow [49], in Python 3.

### 4.3.1 Initial configuration

The architecture of the LSTM Classifier uses two distinct types of layer: one, fully-connected layer; and two, an LSTM layer. The number of layers and the layer size are set as hyperparameters and optimized later.

The number of epochs used is 300. Early stopping is used. If after 20 epochs the binary cross-entropy shows no improvement, the training is halted and skips to the next training-validation window. This is especially useful during hyperparameters optimization, since it speeds up training, therefore needing less CPU time. The activation function is another hyperparameter whose best value is to be discovered later. The possible range of activation functions is TanH, ReLU, LeakyReLU, PReLU, ELU, SoftSign, Softplus. The batch size used was 64 for all layers. The optimizer selected was Nadam, learning rate used was 0.0002, and the learning decay was 0.0004.

Although this was the initial configuration of hyperparameters, other possibilities were tested, including different early stopping thresholds, different batch sizes, batch normalization.

## 4.4 Evaluation methods

Training logs, metrics, and plots are saved to Google Cloud Storage during and after the training iterations. These can be easily accessible through a Tensorboard interface. Tensorboard allows for an interactive view and comparison of several models at the same time. In order to easily build these, even for models which don't use Tensorflow, we used tensorboardX [50].

### 4.4.1 Countering survivorship bias

One critic that could be done to this study is that it is important to include dead companies when sampling companies' information for purposes of building investment strategies. Failing to do so may cause survivorship bias. This study is one of the limitations of this study, and future works should take this into consideration when looking at these results.

However, to mitigate this problem, we'll be evaluating our investment strategy against a benchmark. This benchmark is a naive strategy that buys every company available in our sample and holds them for 1 year. 1 year later it sells the companies and buys again any companies available.

### 4.4.2 Classifier evaluation

At the end of each iteration of training and validating, classification metrics were calculated and saved, for later plotting and analysis The model can then be evaluated in terms of its Accuracy, and be better analyzed by viewing the plot of its Confusion matrix.

### 4.4.3 Portfolio evaluation

At the end of each iteration of training and validation, the top 100 companies most likely to over-perform the average return, i.e. the top 100 companies with the highest classification probability,

were selected to form an investment portfolio. Every stock in this portfolio is equally weighted. The portfolio is then held for a year and it is only again changed at the end of the next iteration. This portfolio can then be evaluated and visualized in terms of its performance (cumulative returns), maximum drawdown, average return (annualized), sharpe ratio (annualized), calmar ratio, and distribution of daily returns.

## 4.5 Hyperparameters optimization

Optimization of hyperparameters is an essential part of learning a black-box model such as deep neural networks. However, this is a complex task to achieve manually. Fortunately, several strategies exist with this purpose such as Random search, Grid Search with log-scales, Evolutionary algorithms, Bayesian optimization, etc. One state of the art technique is Google Vizier [51] and is commercially available through the AI Platform Hypertune service. Although it displays several hyperparameters optimization techniques, it defaults to Bayesian optimization, more specifically, Batched Gaussian Process Bandits.

Google's recommendation is to run at least 10 times more trials than there are hyperparameters. Furthermore, Hypertune allows some trials to be parallelized, but at the cost of losing some effectiveness in hyperparameter search. Typically, in the experiments done in this study, the number of parallel trials was set either to 4 or 8. Each trial runs on its independent machine (4 vCPUs and 15 GB RAM) and takes approximately 3h to fully train.

The metric selected to be maximized during Hyperparameter optimization is Accuracy.

## 4.6 Feature Importance

We want to understand feature importance and establish a ranking of features by importance. As a proxy for feature importance, we measure the Shapley values, as suggested in [52].

This procedure was repeated for every validation set. The Shapley values were then aggregated into a dataframe. Because of the application of PCA whitening to the features, the only possible Sharpley Explainer available for use was the KernelExplainer.

# Chapter 5

# Experiments

## 5.1 Comparison of datasets

7 variant datasets were tested, built using financial ratios, financial ratios year-over-year changes, fundamentals year-over-year changes, or a combination of the previous. The datasets tested can be described as follows:

R: Fundamental Ratios - results from the linear combination of fundamentals
C: Fundamentals Changes - relative to the year prior
RC: Fundamental Ratios Changes - relative to the year prior
R+C: Concatenation of features from the datasets R and C
R+RC: Concatenation of features from the datasets R and RC
C+RC: Concatenation of features from the datasets C and RC
R+C+RC: Concatenation of features from the datasets R, C and RC

| Dataset | Accuracy | Calmar Ratio | Mean Return |
|---|---|---|---|
| R | 54.4% | 0.086 | 15.4% |
| C | 53.0% | 0.056 | 9.5% |
| RC | 52.2% | 0.044 | 7.5% |
| R+C | 53.5% | 0.079 | 13.3% |
| R+RC | 53.3% | 0.081 | 13.9% |
| C+RC | 53.0% | 0.054 | 9.3% |
| R+C+RC | 53.7% | 0.082 | 13.9% |

Table 5.1: Comparison of datasets variants, measured after learning the Logistic Regression Classifier using each of the datasets. For each dataset, the metrics were computed on each of the validation sets, and then aggregated using the mean.

| Algorithm | Log-loss (val) | Accuracy (val) | Log-loss (test) | Accuracy (test) |
|---|---|---|---|---|
| KNN | 16.01 | 53.65% | 15.23 | 55.89% |
| NB | 16.27 | 53.48% | 15.35 | 55.56% |
| LR | 15.76 | 54.38% | 15.21 | 55.97% |
| BLR | 15.83 | 54.17% | 14.84 | 57.02% |
| RF | 15.84 | 54.13% | 15.68 | 54.61% |

Table 5.2: Comparison of K-Nearest Neighbors (KNN), Naive Bayes (NB), Logistic Regression (LR), Bagging Logistic Regression (BLR), and Random Forests (RF). For each validation set and testing set, the Log-loss and the Accuracy were measured, and respectively aggregated using the mean.

The dataset which achieved the best performance was R, as shown in Table 5.1. As such, the R dataset was the only dataset used for the remaining model evaluations.

## 5.2 Comparison of Classifiers

### 5.2.1 Baseline Classifiers

For the training and evaluation of the Baseline Classifiers, a sliding window of 20 years was used. Each training sample consisted of approximately 10,000 instances.

In the case where the available number of instances was lacking, which was recurrent for the earliest years, the bootstrap method was used to sample more instances. For each training set, an equal amount of samples per year was guaranteed. In doing so, we are able reduce the selection bias.

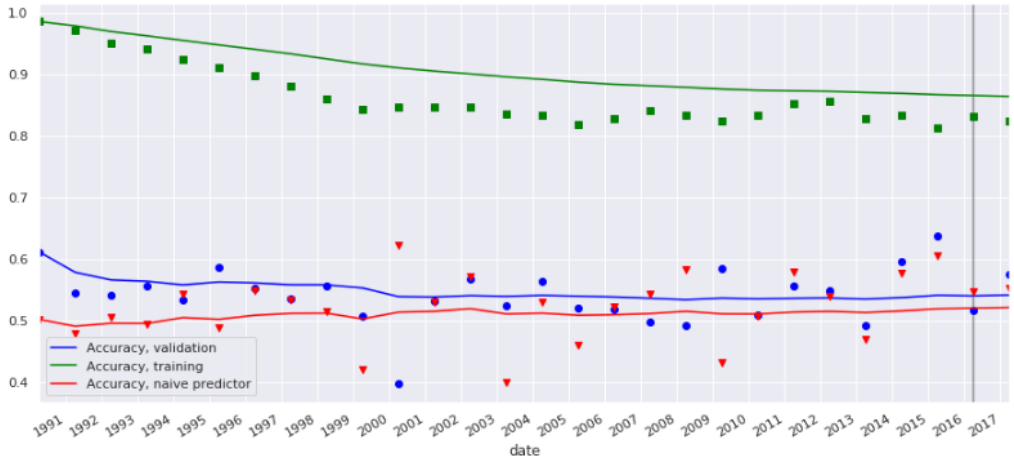


Figure 5.1: Random Forests Classifier: accuracy on the training set, accuracy on the validation set, and accuracy on the naive predictor (classifies all labels as 1). The lines represent the expanding mean value for each scatter plot. The gray line marks the split between the validation set and the out-of-sample testing set.

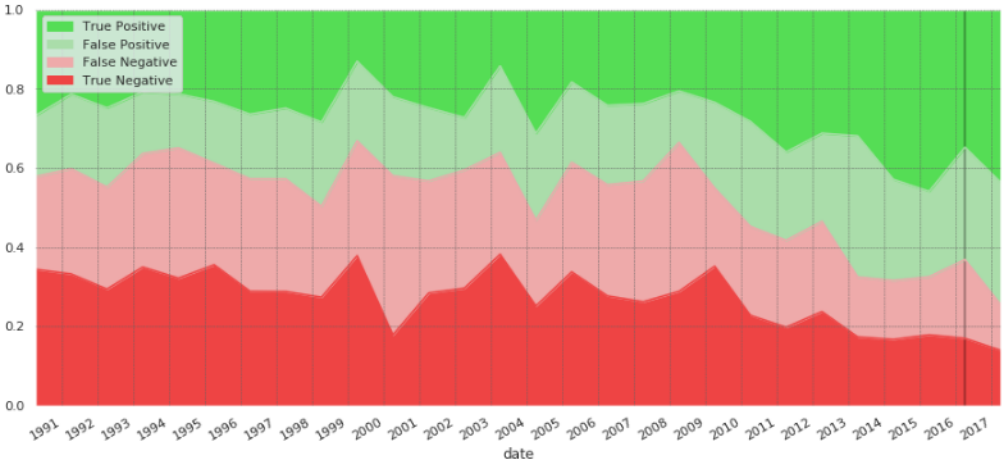


Figure 5.2: Confusion plot for the Random Forests Classifier. The gray line marks the split between the validation set and the out-of-sample testing set.

All Baseline Classifiers were implemented using scikit-learn. The best results for the Naive Bayes Classifier were found using the Gaussian Naive Bayes implementation. The best results for the KNN Classifier were found using hyperparameters parameters: n_neighbours=500, and leaf_size=500. The best results for the Logistic Regression Classifier were found L1 regularization. The best results for the Bagging Logistic Regression Classifier were found using L1 regularization and hyperparameters parameters: bag_estimators=1000, bag_samples=0.95, and bag_features=0.65. The best results for the Random Forests Classifier were found using hyperparameters parameters: n_estimators=1000, max_features=0.13, min_samples_split=0.016, criterion='gini', min_samples_leaf=1, and bootstrap=False. The scikit-learn implementation combines decision trees by averaging their probabilistic prediction, instead of letting each decision tree vote for a single class.

All Baseline Classifiers achieved similar Log-loss and Accuracy measurements, on the average validating set, as seen in Table 5.2. On the average testing set, both Log-loss and Accuracy measurements improved. All Baseline Classifiers improved an approximately equal amount, except for Bagging Logistic Regression which improved the most, and Random Forests which improved by a negligible amount. The lack of improvement in Accuracy and Log-loss by the Random Forests Classifier could be due to over-optimizing hyperparameters and overfitting the validation set.

Similar to what is seen in Figure 5.1 and in Figure 5.2 for the Random Forests Classifier, all baseline models performed poorly during the financial crisis of 2007-2009. This could suggest that the models are unable to predict and correct for sudden macroeconomic changes over a 1-year time horizon.

### 5.2.2 LSTM Classifier

The neural network architecture was formed by 3 LSTM layers of 64 neurons, followed by 2 fully-connected feed-forward layers of 128 neurons.

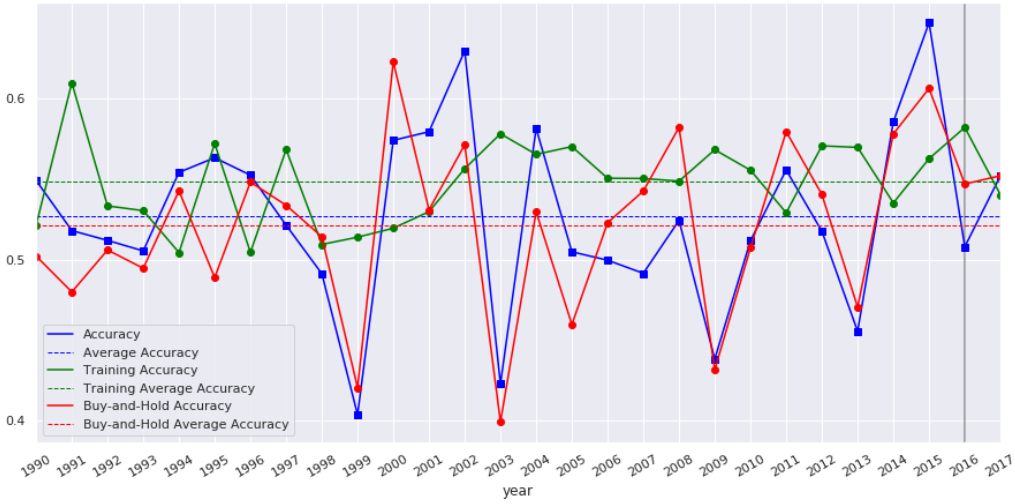


Figure 5.3: LSTM Classifier: accuracy on the training set, accuracy on the validation set, and accuracy on the naive predictor (classifies all labels as 1). The dotted lines represent the mean values. The gray line marks the split between the validation set and the out-of-sample testing set.

The optimizer chosen was Nadam with a learning rate of 0.0001, and a schedule decay of 0.00004. Batch normalization was tested, using several learning rates, and resulted in a worse performance. Therefore, batch normalization was disabled.

The best results were found using a batch size of 64, and the maximum number of epochs of 300. If, after 20 epochs, no improvement in performance was found, early stopping was activated.

In terms of regularization, L1 was set to 0.00180, L1-to-L2 ratio was set to 2.38981, dropout was set to 0.00278, and recurrent dropout was set to 0.00567.

The activation functions TanH, ReLU, Leaky ReLU, ELU, PReLU, SoftSign, and Softplus were tested. TanH achieved the best score.

On the validation sets, the LSTM neural network achieved a mean Accuracy of 54.05% and a mean Log-loss of 15.87. On the testing sets, the LSTM neural network achieved a mean Accuracy of 52.18% and a mean Log-loss of 16.67. Figure 5.3 shows the Accuracy measurements for the various tested periods.

As was the case with the previously studied models, as seen in Figure 5.4, the LSTM classifier performed similarly during the financial crisis of 2007-2009. This could suggest exposition to sudden macroeconomic changes.

As a consequence of the optimization necessary to the learning process of the neural network architecture, over 500 hyperparameter optimization trials were run. Ultimately, the small improvement in accuracy could possibly be explained by overfitting the validation set during hyperparameter optimization.

If using a GPU to speeding the learning process, the learning time is reduced to 3 hours. On the other hand, this is probably not the best approach as it is approximately 5 times more costly, and therefore not the best solution when on a tight budget, or under time constraints.

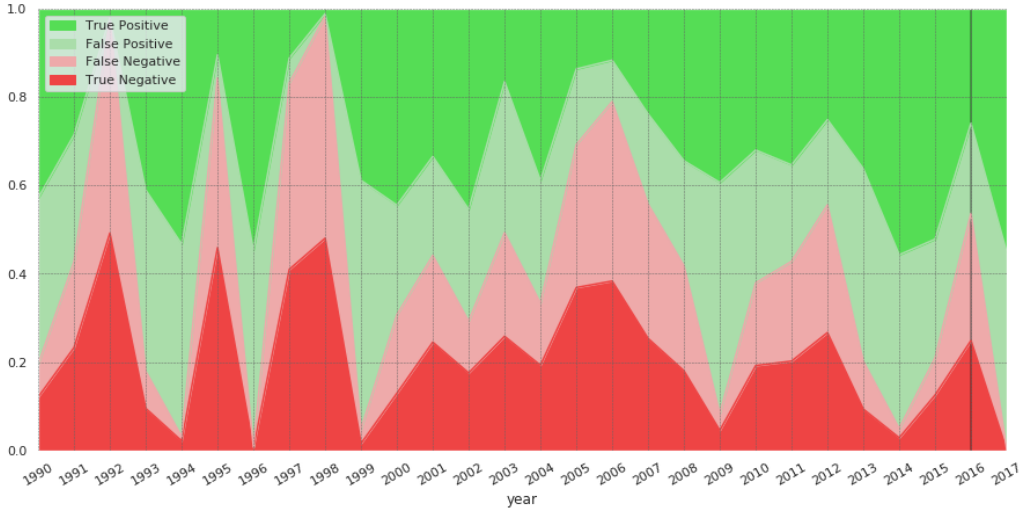


Figure 5.4: Confusion plot for the LSTM Classifier. The gray line marks the split between the validation set and the out-of-sample testing set.

Due to these limitations, although the possibility of using a Bagging LSTM Classifier was considered, training and evaluating such a model would require substantially more time than was available for this study.

### 5.2.3 Assessment

All 6 studied classifiers achieve similar accuracy for the periods tested. Although the accuracy is not exceptionally distant from 50%, even for small values of accuracy likely better than the probability of a coin toss, it could be possible to have enough edge in order to implement a statistical arbitrage investment strategy with an expected value that is positive.

Given the arguments presented above, despite the LSTM Classifier showing improvement over Baseline Classifiers, the LSTM neural network is greater in complexity, is harder to train and maintain, and is more prone to overfitting.

While any of the Baseline Classifiers took under 5 minutes to learn a model, and be evaluated, the LSTM neural network approach takes approximately 5 hours to accomplish such task. Due to the resources required to properly train the LSTM Classifier, we suggest using the Random Forests Classifier instead.

## 5.3 Feature Importance

With the goal of understanding featuring importance and ranking features, the Shapley values were analysed for the Logistic Regression Classifier, as seen in Figure 5.5, and the Naive Bayes Classifier, as seen in Figure 5.6.

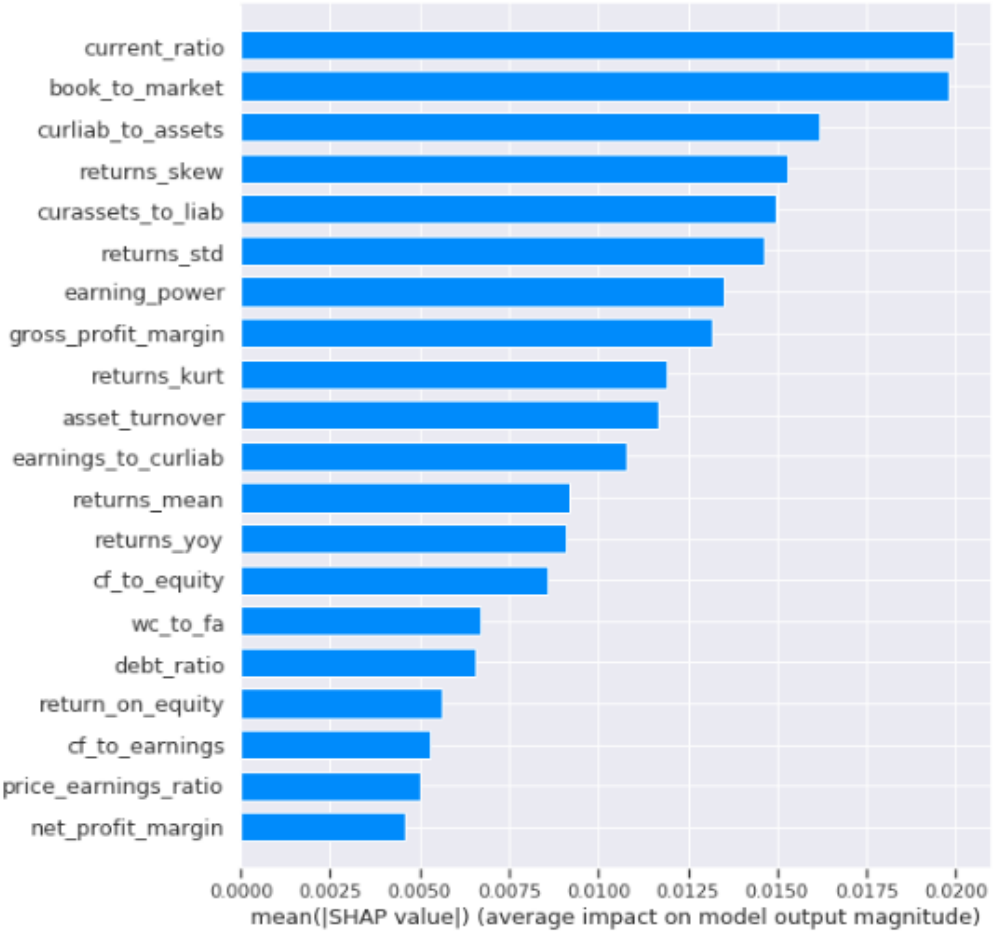


Figure 5.5: Shapley values per feature tested, using the Logistic Regression Classifier. The horizontal axis measures the Shapley values, i.e. the average impact each feature had on the model's probabilities of classification.

Further testing revealed an increased performance of some Classifiers by removing a set of particular features. For the Naive Bayes Classifier, the removal of features Debt Ratio and Leverage Ratio showed and increase of 5% in Accuracy. Conversely, the KNN Classifier showed no improvement upon the removal of any feature. No experiments regarding Shapley values were done for the LSTM Classifier, due to it's exceedingly large demand for computational time.

## 5.4 Feasibility of Investment Strategy

### 5.4.1 Year-over-Year return

The mean Year-over-Year returns for the overall validation period are: 12,17% for the KNN Classifier, 14,36% for the Naive Bayes Classifier, 15,39% for the Logistic Regression Classifier, 15,60% for the Bagging Logistic Regression Classifier, 17,57% for the Random Forests Classifier, as seen

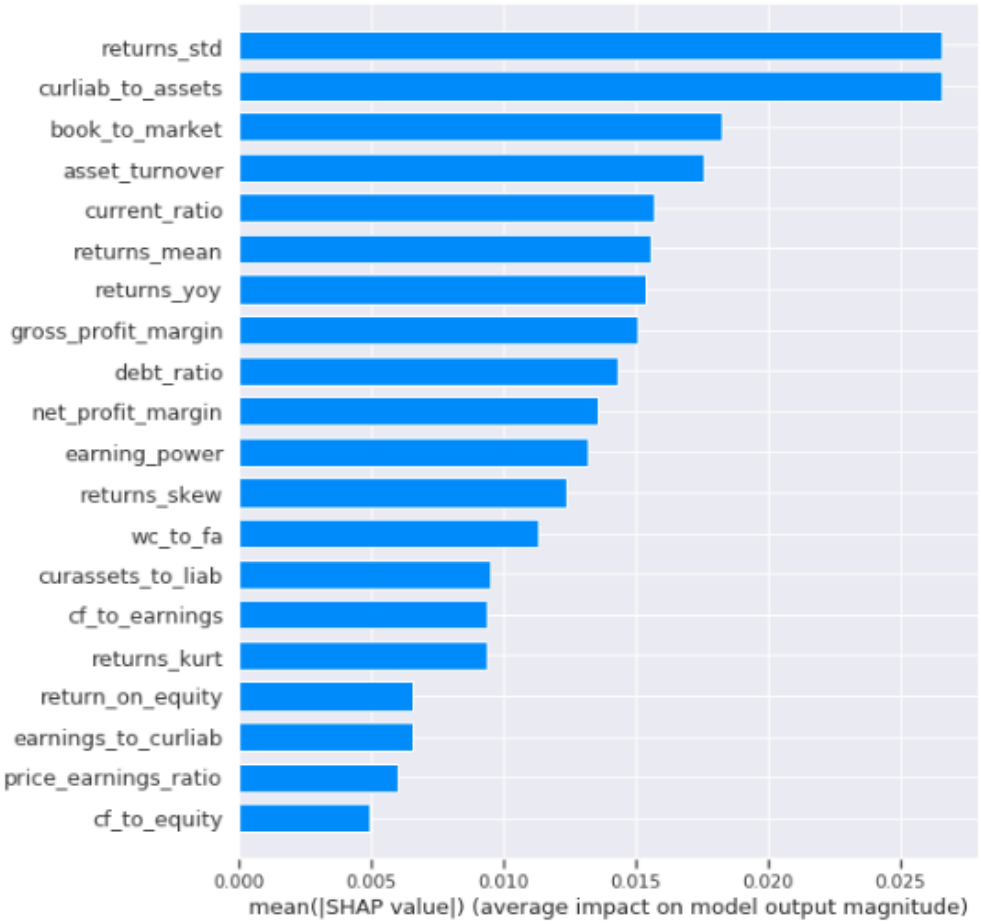


Figure 5.6: Shapley values per feature tested, using the Naive Bayes Classifier. The horizontal axis measures the Shapley values, i.e. the average impact each feature had on the model's probabilities of classification.

in Figure 5.7, and 15,80% for the LSTM Classifier, as seen in Figure 5.8. Note that these values don't account for survivorship bias.

### 5.4.2 Survivorship bias

In order to account for the survivorship bias present in the dataset, we firstly calculate the difference in the returns of the naive strategy, and the returns of the market, using the S&P 500 as a proxy for the overall market returns. Secondly, we use this difference in returns to subtract to the Classifiers' portfolios returns. Moreover, the resultant returns were cumulatively summed and ploted in Figure 5.9 and in Figure 5.10, labeled as "Model Corrected".

To clarify, this is a crude estimate of the survivorship bias, and a crude adjustement. We cannot be certain of the survivorship bias rate calculated in this sample. Changes to the update and maintenance policy of the Thomson Reuters database, stock market crashes, among other factors cannot be completely controlled for.


Figure 5.7: Year-over-Year return for the Random Forests Classifier. The dotted line represents the mean value for the entire validation period.

Note that none of the previous values account for inflation.

### 5.4.3 Returns

Among the Baseline Classifiers, the portfolio built using the Random Forests Classifier achieved the best mean returns and cumulative returns, while the portfolio built using the KNN Classifier achieved the worse mean returns and cumulative returns.

Caution is advised in that, despite having an able classifier, we are not guaranteed to be able to convert its predictions into a profitable investment strategy. A harsh selection and weighting of stocks within the portfolio could yield poor overall portfolio performance, despite having good stock picks.

Another possible explanation for the Classifiers' results is that the stocks predicted to outperform are also associated with higher risk. While this is a relevant concern, it is not the goal of this study to understand how to reduce investment risk.

Consequently, we make use of a naive portfolio selector which selects the top 100 stocks most likely to yield overperforming returns, as indicated by the Classifier.

Both the Random Forests Classifier and the LSTM Classifier model display similar results in terms of cumulative returns, as seen in Figure 5.9 and in Figure 5.10. However, during the testing period, the LSTM underperforms the Logistic Regression model. A possible explanation could be the overfitting to the validation set. Both models show a decreased performance from the year 2000 to the year 2003 and from the year 2007 to the year 2009. These periods are coincident with financial crises.

It can be concluded that, even after adjusting for sampling bias, the learned models' predictions are useful in building a profitable investment strategy, as evidenced by the results in Figure 5.9 and in Figure 5.10.

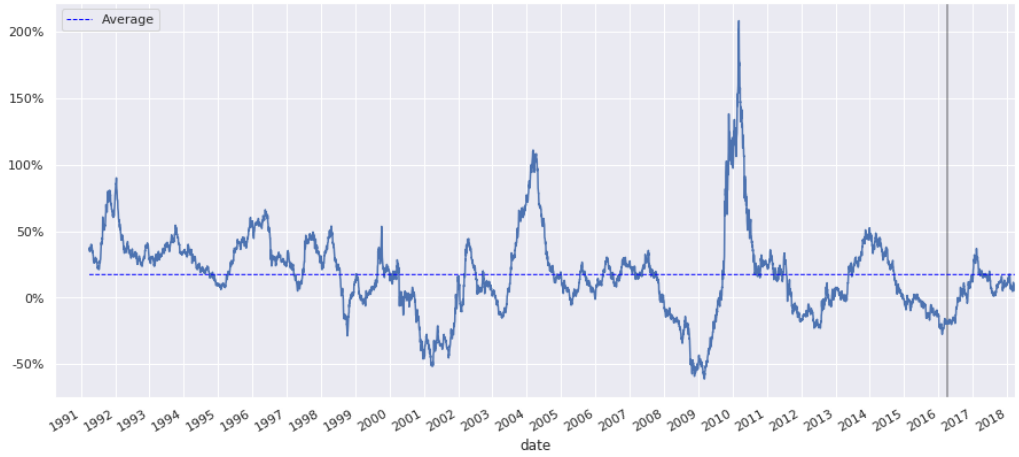


Figure 5.8: Year-over-Year return for the LSTM Classifier. The dotted line represents the mean value for the entire validation period.

### 5.4.4 Transaction costs and Inflation

Transaction costs were not simulated during the training, validation or testing of the model. We define transaction costs as the round-trip costs of buying and selling a stock, which includes the bid-ask spread as well as broker fees. We do not account for taxes. Thus, to account for these costs, based on information from [53], we overestimate the average historical transaction cost per trade for the studied period to be around 2%. Note that since the abolition of fixed-rate commissions in 1975, transaction costs have greatly decreased. Nowadays, the cost arising from broker fees is typically fixed per transaction, e.g. 5$. Hence, if we were to use this investment strategy today, we could expect transaction costs less than 1% relative to the amount invested, as long as the invested amount was moderately high, e.g. 100,000$.

Despite transaction costs, the portfolios built from the Classifiers achieve overall positive returns for the period studied.

Moreover, due to the direct comparison of the portfolios' returns to the S&P 500 returns not adjusted to inflation, no adjustment was done to the portfolio returns account for inflation.

### 5.4.5 Drawdown

The Maximum Drawdown is measured in as the relative change in cumulative returns from the highest point to the subsequent lowest point.

Typically, the Drawdown is measured since the year's start or some other arbitrary periodic point in time. This inevitably results in measurement of Drawdown substantially less than the one depicted, .e.g: if the performance of the investment were to fall for 3 years straight, each one with a Drawdown of 20% relative to year start, the lowest point measured would still be 20% for the third year, as opposed to 60%. The measurement of Drawdown using the year's start is not ideal, especially when analyzing Drawdown considering the possibility of leveraging the

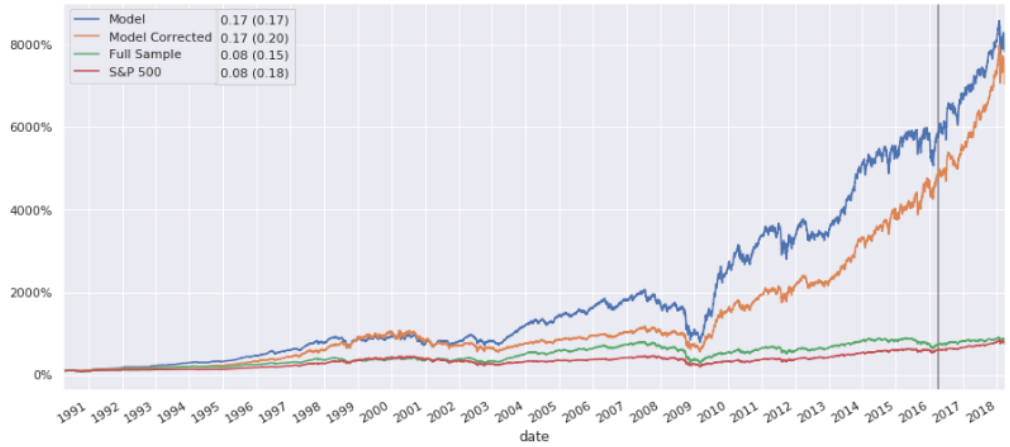


Figure 5.9: Cumulative returns for the portfolio built using the Random Forests Classifier's top 100 predictions. The "Model Corrected" shows the cumulative returns of the portfolio after adjusting for sampling bias. The gray line marks the split between the validation set and the out-of-sample testing set.

investment. Leveraging an investment is only feasible when the Drawdown is reduced. Leveraging an investment with high amounts of expected Drawdown could result in substantial loses to the investor.

As seen in Figure 5.11 and Figure 5.12, we see a confirmation of the decreased performance from the year 2000 to the year 2003 and from the year 2007 to the year 2009. Overall, the Maximum Drawdown for every portfolio built reached its most extreme between the year 2007 and 2009, where it ranged between 60% and 70%.

The Maxmum Drawdown measurements are indicative that the portfolio risk is not optimized. This metric could be reduced by using a separate model dedicated to portfolio weights optimization.

## 5.5 Discussion

The most relevant improvements in performance were the result of improving preprocessing and feature engineering prior to learning the models. Therefore, we argue that improving featuring engineering is most likely time better spent than attempting to improve the Baseline Classifiers' choices of hyperparameters, or the architecture of the LSTM Classifier.

Contrasting with similar research on forecasting of stock prices[54] [55], we find that DL is not the best choice for a learning algorithm. The Random Forests Classifier gives better mean returns. Moreover, the remaining Baseline Classifiers give reasonably similar results to the LSTM Classifier, with less risk of overfitting, and at a fraction of the cost and time required. Consequently, a Baseline Classifier such as the Logistic Regression, or the Random Forests, should be preferred to the LSTM Classifier.


Figure 5.10: Cumulative returns for the portfolio built using the LSTM Classifier's top 100 predictions. The gray line marks the split between the validation set and the out-of-sample testing set.

As supported by [56], the sensitivity to sudden macroeconomic market could be reduced by increasing the holding period, i.e. making predictions for 10 years ahead instead of just 1 year ahead. The reason is that predictions that are made just 1-year head may be strongly affected by the business cycle, whereas predictions made 10 years ahead should not.

One criticism of the methodology is the limited amount of data for the hold-out set. Ideally, we should have at least 10 years for testing, covering at least one bull-market (uptrending market) and one bear-market (downtrending market). However, because of the overall reduced dataset, by reserving the most recent 10 years for testing, we would lose a substantial portion of the data usable for training and validating.

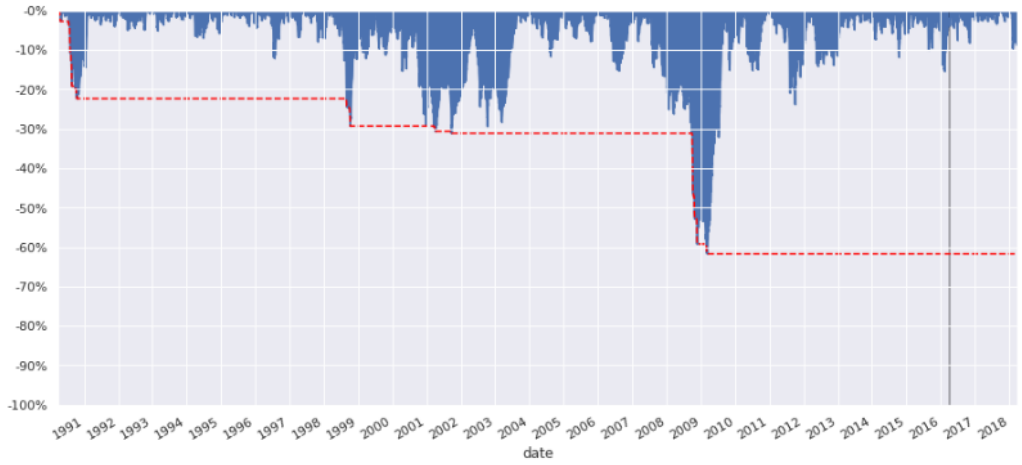


Figure 5.11: Maximum Drawdown for the portfolio built using the Logistic Regression Classifier's top 100 predictions. The gray line marks the split between the validation set and the out-of-sample testing set.

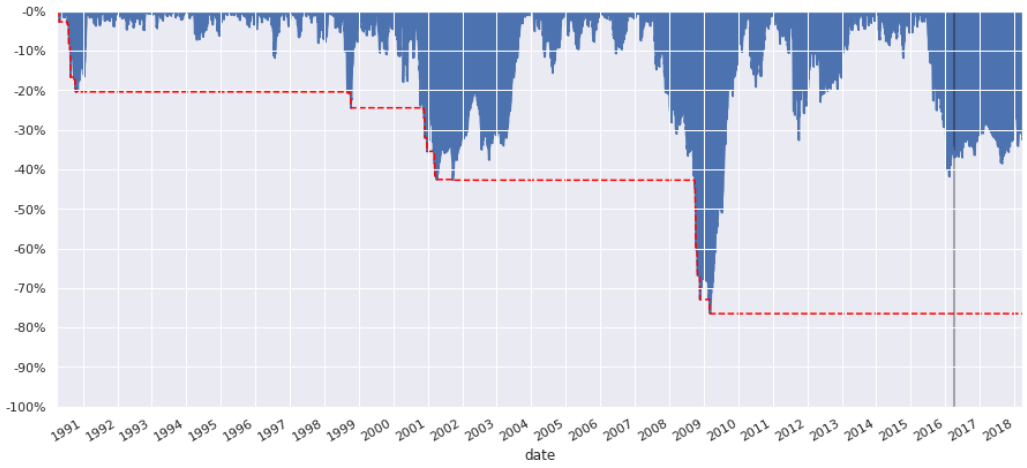


Figure 5.12: Maximum Drawdown for the portfolio built using the LSTM Classifier's top 100 predictions. The gray line marks the split between the validation set and the out-of-sample testing set.

# Chapter 6

# Conclusions

The main goal of learning a classification model capable of predicting where a stock was either overperforming or underperforming relative to the sample average was achieved.

Several learning algorithms were tested, using yearly features derived from both company fundamentals and market data. The algorithms K-Nearest Neighbors, Naive Bayes, Logistic Regression, Bagging Logistic Regression, and Random Forests were used as Baselines to compare to the LSTM neural network. The Logistic Regression Classifier and the Bagging Logistic Regression Classifier achieved the best Accuracy on the validation sets and out-of-sample sets. Despite the disproportionate time used for learning parameters and optimizing hyperparameters, the LSTM Classifier did not surpass the Baseline Classifiers in Accuracy. Moreover, the decreased Accuracy measured during the out-of-sample tests suggests that the LSTM Classifier overfitted the validation sets.

Ultimately, the top 100 predictions from the classifiers were used to build a portfolio of 100 stocks, equally weighted. Despite achieving profitability, the risk associated with the investment strategy makes the investment not suitable for the general investor. Moreover, the portfolio built using the LSTM Classifier's predictions underperformed all remaining models during the tests performed on the out-of-sample period. This could be explained by the LSTM Classifier's elevated capacity for overfitting.

One problem found is that this portfolios built were not immune to stock market crashes, as seen during the years between 2007 and 2009, achieving a measurements of Maximum Drawdown ranging between -60% and -70%. However, the portfolios built were not diversified nor optimized in terms of risk, position weighting, and covariance. This risk could be minimized by, e.g. counterbalancing this investment with other assets negatively correlated, or allowing the Shorting of stocks predicted to underperform. Proper risk management could reduce the Maximum Drawdown in the investment strategy, making it more appropriate for investors.

## 6.1 Further Work

The classifiers learned are the first step toward building a complete investment model. The predictions could be used as input to a second model tasked with optimizing the portfolio's stocks weights. Moreover, instead of selecting the 100 stocks with the highest classification score from the sample and attributing equal weights to each stock, the model could actively select the best stocks and learn which weights each stock should have, according to their respective features, as a part of the whole portfolio. The ideal model should achieve a portfolio composition that reduces Maximum Drawdown, even in the eventual stock market crashes.

In terms of data, the study could greatly improve by using financial data from all available companies from NYSE and NASDAQ for the time range selected, including dead companies. However, such datasets are usually expensive and are not guaranteed to not have missing or erroneous data. The data collected could also be quarterly data, noted with the date of release. By having the date of release of the financial statements, we could be more certain of not incurring in look-ahead bias, when forming predictions for the year ahead.

To further decrease the risk and increase model performance, the investment horizon could be increased, using a holding period of 10 years, instead of 1.

# References


[1] T. Fischer and C. Krauss, "Deep learning with long short-term memory networks for financial market predictions," *European Journal of Operational Research*, vol. 270, pp. 654–669, Oct. 2018.

[2] R. Maurya, "Fundamental and Technical analysis," July 2016.

[3] M. L. d. Prado, *Advances in Financial Machine Learning*. New Jersey: Wiley, 1 edition ed., Feb. 2018.

[4] B. G. Malkiel and E. F. Fama, "Efficient Capital Markets: A Review of Theory and Empirical Work*," *The Journal of Finance*, vol. 25, pp. 383–417, May 1970.

[5] U. SEC, "SEC.gov | Form 10-Q," 2011.

[6] D. A. Hirshleifer, "Behavioral Finance," SSRN Scholarly Paper ID 2480892, Social Science Research Network, Rochester, NY, Aug. 2014.

[7] H. Jacobs, "What explains the dynamics of 100 anomalies?," *Journal of Banking & Finance*, vol. 57, pp. 65–85, Aug. 2015.

[8] A. W. Lo, "The Adaptive Markets Hypothesis," *The Journal of Portfolio Management*, vol. 30, pp. 15–29, Jan. 2004. 01179.

[9] R. Hudson and A. Gregoriou, "Calculating and Comparing Security Returns is Harder than you Think: A Comparison between Logarithmic and Simple Returns," SSRN Scholarly Paper ID 1549328, Social Science Research Network, Rochester, NY, Feb. 2010.

[10] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, Aug. 1996.

[11] Tin Kam Ho, "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, pp. 832–844, Aug. 1998.

[12] K. P. F.R.S, "LIII. On lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, pp. 559–572, Nov. 1901. 00000.

[13] H. Hassan, A. Negm, M. Zahran, and O. Saavedra, "Assessment of artificial neural network for bathymetry estimation using high resolution satellite imagery in shallow lakes: Case study el burullus lake," 04 2015.

[14] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, (USA), pp. 807–814, Omnipress, 2010. 06090 event-place: Haifa, Israel.

[15] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (G. Gordon, D. Dunson, and M. Dudík, eds.), vol. 15 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 315–323, PMLR, Apr. 2011. 03205.

[16] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp," in *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, (London, UK, UK), pp. 9–50, Springer-Verlag, 1998. 02631.

[17] T. Gao and Y. Chai, "Improving stock closing price prediction using recurrent neural network and technical indicators," *Neural Computation*, vol. 30, no. 10, pp. 2833–2854, 2018.

[18] J. B. Heaton, N. G. Polson, and J. H. Witte, "Deep Learning in Finance," *arXiv:1602.06561 [cs]*, Feb. 2016.

[19] S.-S. Kim, "Time-delay recurrent neural network for temporal correlations and prediction," *Neurocomputing*, vol. 20, no. 1-3, pp. 253–263, 1998.

[20] E. Saad, D. Prokhorov, and D. Wunsch II, "Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks," *IEEE Transactions on Neural Networks*, vol. 9, no. 6, pp. 1456–1470, 1998.

[21] Y.-K. Kwon and B.-R. Moon, "Daily Stock Prediction Using Neuro-genetic Hybrids," in *Genetic and Evolutionary Computation — GECCO 2003* (E. Cantú-Paz, J. A. Foster, K. Deb, L. D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowsland, N. Jonoska, and J. Miller, eds.), Lecture Notes in Computer Science, pp. 2203–2214, Springer Berlin Heidelberg, 2003.

[22] N. Khoa, K. Sakakibara, and I. Nishikawa, "Stock price forecasting using back propagation neural networks with time and profit based adjusted weight factors," pp. 5484–5488, 2006.

[23] D. L. Minh, A. Sadeghi-Niaraki, H. D. Huy, K. Min, and H. Moon, "Deep Learning Approach for Short-Term Stock Trends Prediction Based on Two-Stream Gated Recurrent Unit Network," *IEEE Access*, vol. 6, pp. 55392–55404, 2018.

[24] H. Y. Kim and C. H. Won, "Forecasting the volatility of stock price index: A hybrid model integrating LSTM with multiple GARCH-type models," *Expert Systems with Applications*, vol. 103, pp. 25–37, Aug. 2018.

[25] Y. Bengio, "Learning Deep Architectures for AI," *Foundations and Trends® in Machine Learning*, vol. 2, pp. 1–127, Nov. 2009.

[26] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997. 15890.

[27] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *Neural Computation*, vol. 12, pp. 2451–2471, Oct. 2000. 01467.

[28] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, pp. 2222–2232, Oct. 2017.

[29] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *arXiv:1406.1078 [cs, stat]*, June 2014.

[30] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," *arXiv:1412.3555 [cs]*, Dec. 2014.

[31] A. J. P. Samarawickrama and T. G. I. Fernando, "A recurrent neural network approach in predicting daily stock prices an application to the Sri Lankan stock market," in *2017 IEEE International Conference on Industrial and Information Systems (ICIIS)*, pp. 1–6, Dec. 2017.

[32] C. C. Aggarwal, *Neural networks and deep learning: a textbook*. Cham: Springer, 2018.

[33] G. Tegnér, *Recurrent neural networks for financial asset forecasting*. PhD thesis, 2018.

[34] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An Empirical Exploration of Recurrent Network Architectures," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 2342–2350, JMLR.org, 2015. 00657 event-place: Lille, France.

[35] F. Chollet and others, *Keras*. 2015. 02469.

[36] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[37] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Dec. 2014.

[38] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv:1609.04747 [cs]*, Sept. 2016.

[39] T. Dozat, "Incorporating Nesterov Momentum into Adam," Feb. 2016.

[40] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly Media, 1 edition ed., Apr. 2017.

[41] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv:1207.0580 [cs]*, July 2012.

[42] Y. Gal and Z. Ghahramani, "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks," *arXiv:1512.05287 [stat]*, Dec. 2015.

[43] G. Hinton, Y. Bengio, and Y. LeCun, "Lecture notes in deep learning," 2015.

[44] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: closing the generalization gap in large batch training of neural networks," *arXiv:1705.08741 [cs, stat]*, May 2017.

[45] T. R. Eikon, "Eikon api." Subscription Service. Accessed in October, 2018.

[46] Y. Finance, "What is the adjusted close? yahoo help - SLN28256," 2018.

[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[48] S. G. Makridakis, S. C. Wheelwright, and R. J. Hyndman, *Forecasting: Methods and Applications*. New York: Wiley, 3 edition ed., Dec. 1997.

[49] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[50] T.-W. Huang, "tensorboardx." https://github.com/lanpa/tensorboardX, 2019.

[51] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. E. Karro, and D. Sculley, eds., *Google Vizier: A Service for Black-Box Optimization*, 2017.

[52] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 4765–4774, Curran Associates, Inc., 2017.

[53] C. M. Jones, "A Century of Stock Market Liquidity and Trading Costs," SSRN Scholarly Paper ID 313681, Social Science Research Network, Rochester, NY, May 2002. 00594.

[54] S. Yadav and K. P. Sharma, "Statistical Analysis and Forecasting Models for Stock Market," in *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*, pp. 117–121, Dec. 2018. 00000.

[55] B. Lin, W. Chu, and C. Wang, "Application of Stock Analysis Using Deep Learning," in *2018 7th International Congress on Advanced Applied Informatics (IIAI-AAI)*, pp. 612–617, July 2018. 00000.

[56] M. Pedersen, "Strategies for Investing in the S&P 500," SSRN Scholarly Paper ID 2653676, Social Science Research Network, Rochester, NY, Jan. 2015. 00000.