A Machine Learning Approach to Stock Market Trading

Benjamin Nicholson - 1810569

2020/2021

Abstract

The use of machine learning (ML) in finance, and particularly within the stock markets, is seldom explored due to the fast-paced and often unpredictable nature of these problems. I attempt to look into using an ML algorithm to forecast likely stock price movements and then to apply these results to an automated trading algorithm. I implemented a random forest ML algorithm with the aid of stock price data and technical indicators for a multitude of existing stocks. This resulted in an estimator that could then be applied to a trading algorithm. Both the estimator and the trading algorithm led to successful results over the years I had tested them on, beating the baseline models I had set out. I learnt many useful things over the course of this project. Firstly, the importance of useful input data when executing an ML model in a financial scenario. There was a necessity for a vast and diverse amount of data, while also ensuring that this data was indeed adding insight into the learning process. Additionally, I found the significance of a sensible and strategic trading method, a key factor I came across is that even the best estimator will not produce sound and consistent results when applied incorrectly. Ultimately, despite the successful results, it is difficult to generalise the methods as completely safe and secure due to the smaller testing size caused by the limitations in available historic data.

Contents

1 Introduction

As made clear by the title, my project is on "A Machine Learning Approach to Stock Market Trading". Any publicly traded company will be available to purchase on a *stock market* and hence have a *stock price*, among which there is a vast world of profit, intelligence and strategy.

The bulk of my project (and hence the following report) is twofold:

1. How the stock market can be interpreted in order to act as a useful tool for a machine learning algorithm.

2. How I can then apply the results of this learnt model to be used effectively within an automated trading algorithm.

A machine learning (ML) algorithm is one that improves through experience of past data, to then provide a prediction on future data. For the purpose of this project, it will be important to find which kind of input data will help aid the machine learning process in producing worthwhile results, and also which machine learning algorithm would be suitable for the task's needs. The former poses an initial challenge of; how can I acquire the appropriate financial data? How much data is necessary to produce good results? Hundreds of types of information exist for a given stock so it's key to filter the useful information that can actually provide some insight into the workings of a stock's price.

Why use machine learning?

Any trading algorithm, or any investment at all for that matter, relies on the idea of numerical problems that need maximising (i.e in profit), the aim is to use the predictive powers of machine learning to forecast results and use these predictions to maximise the performance (profit) of these numerical problems.

The stock market, in particular, has key aspects which I have identified as useful traits for any machine learning algorithm:

- High volumes of data are available and processed every day.
- There is a demand for quantitative decisions to be made.
- We require high accuracy predictions to be made in order to be successful.

These reasons combined show how the integration of artificial intelligence (specifically machine learning in this case) in finance is a match made in heaven, an idea emphasised by Bernard Marr in an article on *bernardmarr.com*[1].

1.1 Motivation

As a Maths & Computer Science student, I have an interest in applied mathematics and particularly in finance. To start the project I had immediately begun thinking of machine learning tasks within finance and was drawn to the idea of some sought-after *"perfect, unbeatable trading strategy"*. My aspirations were not quite **that** high, but I was still compelled by the idea of these automated trading algorithms and why (or why not) they could work.

Due to the rising success and popularity of machine learning and its applications over the last few decades, many people have looked into its various uses within finance and often found difficulty when being face with reliability and robustness of results. Additionally, where success is found, due to the nature of the task, results are rarely disclosed in order to continue to exploit these methods for profit.

Overall, the challenge promises to be difficult but interesting. However, even with those difficulties, it can't be ignored that machine learning offers a fresh, forward-thinking approach that could ultimately allow investment companies to maximise their profits and attract new customers.

1.2 Objective

Ultimately the aim of this project is to explore the success of machine learning and attempt to apply it to the stock market, an area often avoided. This is **not** a *"get rich quick scheme"* and thus I will not be placing my personal success of the project on the results and profitability of the concluding methods. Instead, I aim to explore the stock market and reflect on the benefits and drawbacks of this kind of approach that I will face while attempting it.

2 Literature Review

While researching existing papers on similar projects, I was encouraged by reading a wide variety of information and advice available. As well as seeing positive results be made from this kind of approach.

This included *"A Cross-Sectional Machine Learning Approach for Hedge Fund Return Prediction and Selection"* [2], which explored the performance of machine learning algorithms and their ability to predict hedge fund returns. The paper looked into the ML algorithms; lasso, random forest, gradient boosting and deep neural network. And although the task at hand differed from what my objective was, it was still uplifting to see the positive results all the algorithms could achieve. It was particularly helpful in giving me ideas and advice on how I should be handling my data; namely the importance of gathering a large plethora of input data. It also warned of the existence of **survivorship bias**. This is the bias caused by ultimately ignoring funds that go bankrupt and completely fail since you are only testing funds that exist *now*, this leads to being unable to fully simulate the genuine circumstances of the real world. This is certainly something that I will need to appreciate as the same bias will exist with companies and their respective stocks.

One of the most clear and expansive sources I found was *"Machine Learning for Factor Investing"* [3], it focused on investment strategies and portfolio optimisation. The methods I found through this paper proved to be extremely helpful when tasked with forming my own methods. Moreover, I took inspiration from their use of multiple stocks that are assessed at a given moment. Where I had initially planned to focus on one stock at a time, with the idea to potentially increase this number, I soon found that the necessity for multiple stocks leads to a larger and more diverse dataset which ultimately leads to better results - this is something that I will discuss further, later in the report.

While looking further into examples more similar to the direction I was planning on going, I found the most well-known case study to be the company **Aidyia**, this is a fully AI-driven hedge fund that has already displayed plenty of success showing consistent results. However, as mentioned previously as one of the task's difficulties, it is difficult to get a lot of information since most of the companies workings remain private. Although in the article *"The Revolutionary Way Of Using Artificial Intelligence In Hedge Funds"* [1], it did still place emphasis on the company's requirement to frequently update the model in order to keep up with the ever-changing conditions of the market.

There also exists much more literature to back up these suggested findings, including *"Financial Time Series Forecasting with Deep Learning: A Systematic Literature Review"* [4] and *"A hybrid stock trading framework integrating technical analysis with machine learning techniques"* [5]. With the former having a larger focus on **deep learning** applications for this problem (a subset of machine learning that spans further than the scope of this project). And the latter utilising *technical indicators*, providing me with supporting evidence to potentially use this kind of data in my own

application. While I was already familiar with technical indicators, this paper really demonstrated their predictive potential. They will be defined and discussed in further detail in a later section.

3 A Machine Learning Approach

The first thing that needed doing was to identify what the problem at hand actually was. After some thought, I arrived at a summarised statement:

> Create a model that can forecast the direction/trend of a stock's price based on input data formed by the given stock.

3.1 Design

While researching various machine learning algorithms and existing applications, I came across an extremely helpful *roadmap* from *TowardsDatScience.com* [6], that outlines principles that can apply to **any** machine learning approach, made up of these necessary aims/milestones.

- State the question and determine the data.
- Acquire the dataset and correct missing data (if applicable).
- Prepare the data and split it into sets.
- Establish a baseline model. This is what the algorithms will compete against.
- Train the model on training data.
- Make predictions on the test data.
- Analyse and compare results.
- Make adjustments (if applicable) and interpret a final model.

3.1.1 Random Forest

The machine learning algorithm I will be choosing to implement is ***Random Forest***. A **decision tree** is a predictive model that implements a tree-like structure to split a set of input data by characteristics and have predicted outputs at each leaf node. A **random forest**, however, is a *collection*, or *ensemble*, of multiple decision trees, where the results of a prediction are either the average of all tree results (for regression) or the mode of the tree results (for classification).

> A ***regression*** ML algorithm is one which attempts to predict a numerical output.
> A ***classification*** ML algorithm is one which instead attempts to predict from a set of possible categories.

The figures that follow outline some simplistic examples to illustrate the difference between these types of algorithms.

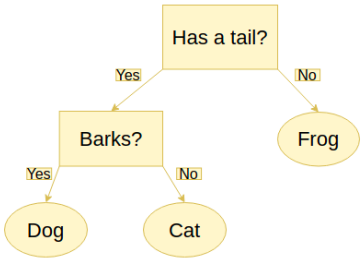

Figure 1: Decision tree classifier for animal prediction.

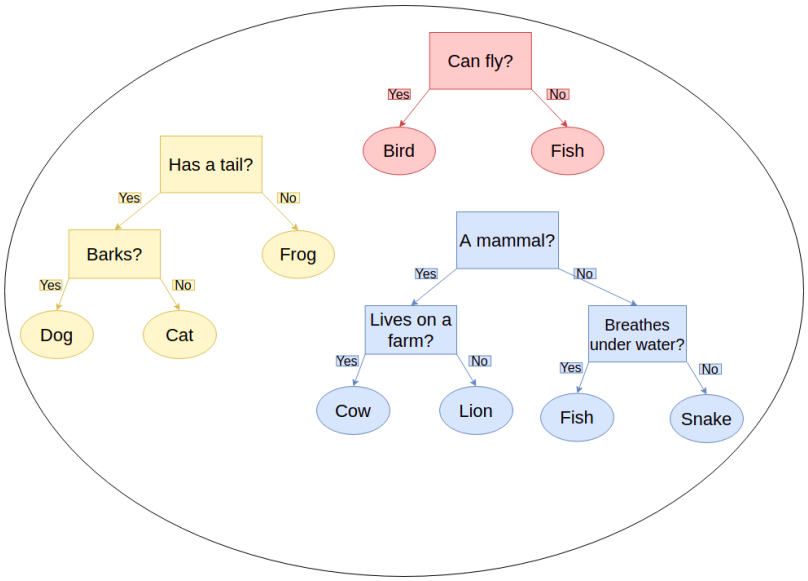

Figure 2: Random forest classifier for animal prediction.

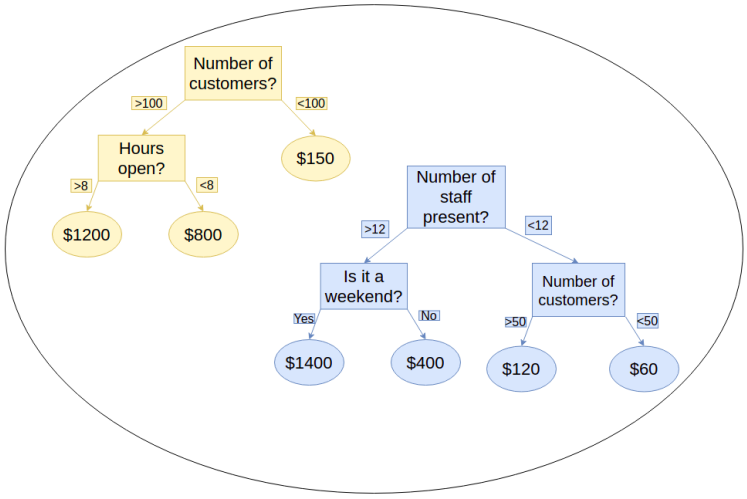

Figure 3: Random forest regressor for predicting a shops profits.

To understand the random forest, the methodology is as follows

Pseudocode

1. **Input:** training set, T, of size N with K input variables and integer M number of trees to be made.
2. Take a sample of T of size $n < N$, with replacement.
3. Randomly select $k < K$ input variables at each node. Take the input variable available that best splits the data at this node.
4. Create each tree as large as possible. (until termination criteria is met).
5. Repeat M times.
6. **Output:** random forest estimator.
7. Predict new data by aggregating predictions in the random forest.

While creating the decision tree we use the sample of data from T to continuously split the data at each **node** until we arrive at outputs at the **leaf nodes** - so for each split, we have a node. I mentioned in step *'3.'* that we must use an input variable that best splits the data. By this I mean asking the right question, for the right variable that leads to the most equal split of all the data while keeping similar classes of results together. The two main splitting criterion I will be looking at is ***Gini impurity*** and ***entropy*** for information gain.

Gini impurity is a measurement of the *likelihood* that a new instance of data is *incorrectly* labelled. The basic idea is to split the data based on whichever feature reduces this impurity.

Entropy is very similar to the Gini impurity, it also is a measurement of impurity and in particular how pure or impure a particular node is. Where high entropy (pure) would mean all data values at a node are of the same class (i.e dog) whereas a low entropy (impure) node would mean it contains many different possible classes (i.e cats, dogs and snakes). The aim is to split the data that maintains high entropy across all nodes.

The addition of using a random forest offers more accuracy while also reducing the variance of results, in comparison to what can be achieved simply by one decision tree. Below are some key overall advantages and disadvantages of the random forest algorithm.

Advantages:

- Proven to be accurate and reliable.
  - There are many examples of the success of random forests, among which was *"A cross sectional machine learning approach for hedge fund return prediction and selection"* [2], as mentioned in Section 2.
- Easy to deduce the importance of each input variable.
- Easy to understand and manipulate.
- Robust against outliers.

Disadvantages:

- Although the increase of decision trees will increase accuracy, it will also slow the algorithm down.
- The outcome estimator is *predictive* but not *descriptive*. So we cannot draw conclusions about the relationship between the variables.

3.1.2 Input Data

As previously mentioned, one of the key aspects of this section of the project is to decipher which kind of input data would prove to be useful for an ML algorithm of this nature. Although this was an early hurdle of the project, it did prove to be one of the most difficult.

Basic price data

My initial plan was to start small and slowly improve the dataset. While researching what kind of data existed and was available about the price of a stock, I soon found that the fundamental price data was **OHLC data**.

> **OHLC data** is the basic price data available for a stock and is simply an acronym for 'Open', 'High', 'Low', 'Close' which respectively represent the open price, the highest price of the day, the lowest price of the day, and the close price of a stock over a given day.
> This kind of data will typically also include the date and the *volume*, which is the amount of this stock that was traded in that trading day.

There were many potential sources for this OHLC data. The best I found were *AlphaVantage* [7] and *Yahoo finance API* [8]. While AlphaVantage was potentially a more extensive source and also offered many different types of data beyond the OHLC data, it did have capped usage for a free account. Where my plan was originally to only analyse the prices of one stock at a time, this cap was not an issue. However, as I began to realise the importance of a larger and more diverse dataset, this did become a restriction and so I ultimately used Yahoo finance.

This data is then stored as a Python data frame and can be easily visualised using graphing methods available with *'matplotlib'*, both shown below using the 'Google' stock.

date	1. open	2. high	3. low	4. close	5. volume
2019-12-31	1335.79	1340.66	1332.13	1339.39	976061.0
2019-12-30	1356.81	1357.00	1337.84	1339.71	1000592.0
2019-12-27	1364.00	1367.07	1353.00	1354.64	1160873.0
2019-12-26	1346.55	1363.20	1345.51	1362.47	1183800.0
2019-12-24	1350.21	1352.01	1344.16	1344.43	673410.0

Figure 4: Google OHLC data

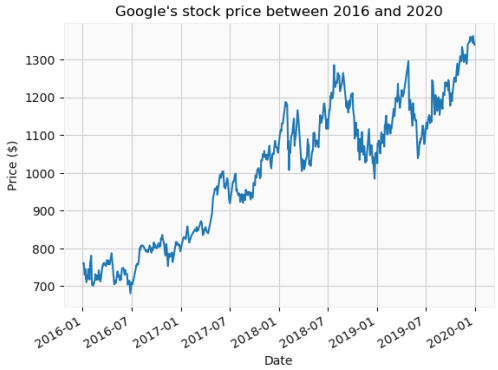

Figure 5: A plot of Google's price data

3.1.3 Technical Indicators and Improved Dataset

I quickly found that this type of data alone would not suffice for use in a successful ML estimator. I would need to both *improve* and *expand* my dataset.

There are two major schools of thought for stock market analysis, ***technical analysis*** and ***fundamental analysis***. Technical analysis focuses on past data, charts and technical indicators to make estimations on the future state of the market and use this to make trading decisions. Since I was already using past price and volume data, it made sense for me to continue along with this rationale and thus include *technical indicators* in my data input. This is an idea that has also been supported by my research [5].

> **Technical Indicator:** Calculations based on price, volume or open interest of a stock, used by traders in order to predict future trends, price reversals, or entry/exit points - definition from *Investopedia.com* [9].

In total, I have incorporated 12 technical indicators to be added to the dataset. They are: 10-day price change (%), 5-day price change (%), Chaikin oscillator, average directional movement index (ADX), Bollinger bands, exponential moving average (EMA), simple moving average (SMA), moving average convergence/divergence (MACD), on-balance volume, relative strength index (RSI), stochastic oscillator, Aroon indicator.

Example - RSI

The relative strength index, or RSI, is a number that measures the magnitude of the recent price changes in relation to the past prices over a certain period. It takes a value in the interval $[0, 100]$, where an RSI$>$ 100 indicates that the given stock is *overvalued*, and an RSI$<$ 100 indicates that the stock is *undervalued*.

The calculations for computing RSI are

$$RSI = 100 - \frac{100}{1 + \frac{(\text{previous average gain} \times 13) \ \times \ \text{current gain}}{-((\text{previous average loss} \times 13) \ + \ \text{current loss})}} \tag{1}$$

where for 'previous average gain/loss' and 'current gain/loss' we have ,

if $\text{Price}_{now} > \text{Price}_{previous}$

$$Gain = \text{Price}_{now} - \text{Price}_{previous} \tag{2}$$

$$Loss = 0 \tag{3}$$

otherwise

$$Gain = 0 \tag{4}$$

$$Loss = \text{Price}_{previous} - \text{Price}_{now}. \tag{5}$$

And we use a lookback window of 14 days to calculate the average. This 14 days is selected as a standard parameter with this technical indicator and will be a useful factor when later deciding the output data.

The remaining technical indicators will have descriptions and calculation definitions in the Appendices section 7.

Originally I was only analysing one stock at a time. The limitations of this included having a 'non-diverse' dataset and not having sufficient data points to apply to a machine learning algorithm which, in its nature, requires a vast access of data. A solution to both of these limitations was to increase the number of stocks I would be analysing in my random forest estimator. The most obvious place to look was to focus on the largest stocks in the world, that are most frequently traded. This would be a natural choice since these would be the same stocks that would be considered for trading in most existing trading strategies. With help from the article, *"Implementation of Technical Indicators into a Machine Learning framework for Quantitative Trading"* [10], I discovered the use of utilising the ***S&P 100***, this is a *stock market index* of some of the 100 most popular and well-established stocks traded in the United States. A *stock market index* is a number/price that is computed by a set of stocks, its aim is to display sector balance whilst also offering a general signal of the state of the markets. I also used the article [10] to learn of the Python package *'beautiful soup'* which helps utilise scraping of web pages, this way I can collect the *tickers* for the 100 stocks that comprise the S&P 100 and then call their respective stock data using Yahoo finance (as previously mentioned). This means it would be easy to update the random forest algorithm if the stocks that make up the S&P 100 were to be updated.

> A **ticker** is a short reference code that is an indicator for a company that is traded on the stock market - e.g. Apple = *'AAPL'*.

3.1.4 Datasplit

How much data?

When attempting to determine how much data I use for my overall dataset, it was worth asking myself a few questions. How will my data be stored? How fast will the algorithm run? How will I be pre-processing the data? It is known for random forests, and most ML algorithms for that matter, that while an increase of data points will improve results, it will also slow down the execution time of the algorithm.

It is also worth noting that the shape of the market has changed a lot over the last few years and so taking information too much into the past will lead to data that does not fairly reflect today's market. Therefore it was better, as planned, to add more data on additional stocks than to go further back in time. In this case, it is the idea that we prefer a *breadth* of knowledge of the current market than a *depth* of knowledge on one stock.

How will it be split?

In any machine learning process we must establish a training and testing dataset, most likely by a split of the overall dataset. Since we are working with time series data, it is better to split the data by years rather than completely randomly. We have two concerns to battle when splitting the data, with smaller sized training data we have a greater variance in the estimator's parameters, meaning results will vary greatly between models. With small testing data, however, the performance of the estimator will have greater variance depending on what it is testing.

Ultimately, I have decided to take the stock price data, and their respective technical indicator

calculations, for the 100 stocks in the S&P 100 between the years 2010 and 2019 inclusive. This results in an approximate 250,000x29 sized data frame. I will use 2010-2018 as the training data for the random forest algorithm, which will then be tested on the 2019 data - this results in a 9:1 training-testing split. I will then use the 2019 data as the year to test the automated trading algorithm. This is a reasonable decision to make as this data is *'unseen data'* for the algorithm, meaning it was not present during the training decisions.

3.1.5 Output data

A random forest algorithm can take one of two forms: a **regressor** (where it is predicting a number) or a **classifier** (where it is predicting from a class). Where I had originally chosen a regressor, I was predicting the given stocks price for the *end* of the next trading day. This method was frankly over-ambitious and nonsensical, producing poor results.

One problem with this idea was that the ability to predict a price was too unrealistic. It was also not necessary as the potential to just forecast trend/direction could still prove extremely useful. The conclusion I got from this was the importance of doing a smaller task well, than a large task poorly.

The other key problem was the choice to predict for the *next* day. This did not give the market enough time to react to what the random forest was attempting to understand. The essence of the stock market is too unpredictable and volatile in nature to make 'next day' predictions with accuracy.

Changing to a classifier, I would now predict the change of a stock's price (UP or DOWN) in 10 days time. This gives enough time for the market to make changes with enough confidence to make a prediction on it. Additionally, most of the technical indicators work on the idea of a *time frame* - i.e the 14 day lookback period for calculating RSI in section 3.1.3. This meant I could tune these indicators to work for a 10-20 day period, meaning the selection of 10-day trend prediction for my output could coincide with the technical indicators.

Output data: Prediction for a stock's price change in 10 days. Output will either be UP or DOWN.

3.1.6 Baseline Models

When pursuing a machine learning problem it is worth taking time to construct a *baseline model*. This model is necessary in order to compare future ML models to help decipher whether the implementation of our model is even producing worthwhile results. I came up with a couple of simplistic models for both a regression and classification problem:

- **Regression:** Constant 1-day change,

$$\text{Tomorrow's price} = \text{Today's price} + (\text{Today's price} - \text{Yesterday's price}). \tag{6}$$

- **Regression:** No change,

$$\text{Tomorrow's price} = \text{Today's price}. \tag{7}$$

- **Classification:** Same price direction change as previous,

$$\text{Tomorrow's change} = \text{Today's change}. \tag{8}$$

- **Classification:** Always predicting UP.

Although these models are very basic, they do provide a lower bound for the algorithm to aim to beat. This is the purpose of any baseline model.

3.2 Hyperparameters and Optimisation

> **Hyperparameters** are the pre-determined parameters for a given ML algorithms whose values control an aspect of the learning process. They to be tuned in order to optimise results.

A random forest contains many hyperparameters. I will go through the four I have identified as key hyperparameters of the algorithm that need tuning and then discuss the process I took in order to optimise these values.

3.2.1 n_estimators

The number of decision tree estimators that will make up the overall *forest*. Too few trees will lead to inaccurate results, while too many will increase the time it takes to execute the algorithm.

3.2.2 max_features

This is the maximum number of features (or input variables) that will be considered at each split of a decision tree. Again, too few will lead to inaccurate results, and too many here will lead to *overfitting*. This is where the algorithm will perform extremely well on the training (seen) data but poorly when faced with the testing (unseen) data. Ultimately it has fit too closely to one kind of data.

3.2.3 max_depth

This is the maximum depth of each decision tree. Similarly to the previous two, too much depth will slow the program down, and too little depth will lead to overfitting.

3.2.4 Criterion

This is the splitting criterion that the algorithm uses, as described in the Random Forest section 3.1.1. I will look at which of the two discussed, *Gini* or *entropy*, will lead to better results.

3.2.5 Validation Curves and Grid Search

I will utilise a **grid search** (available on the *'scikit-learn'* package). This is an exhaustive method of finding the best possible ML estimator by creating an instance of the ML algorithm by going through every combination of the hyperparameters chosen. For this, I need to decide which hyperparameters to optimise (this has already been done) and which values of these hyperparameters to enter into the grid search.

For **max_features** we will look at two computational functions $\sqrt{n\_features}$ and $log_2(n\_features)$ where $n\_features$ is the number of features or input variables per data point.

For **criterion** we will be looking at either, as discussed, Gini or entropy for information gain. Although similar, these two calculations may produce differing results (which could prove vital).

For the remaining two hyperparameters it was harder to find a start point at exactly which values should be looked into. I decided to make use of ***validation curves***. These are graphs that plot both the *training score* and *test score* of the random forest over a wide variety of possible entries for a given hyperparameter. A *score* is a number between 0 and 1 that represents the performance of the algorithm on its respective data set. The aim of this method is to inspect the plotted results and choose a value of the hyperparameter where the training and testing scores most closely converge. The idea is to then make use of this value and focus on integers around it to use in the grid search.

max_depth

The max_depth validation curve was done for the values: *max_depth*=10, 20, 50 and 100. The results plotted the following graph,

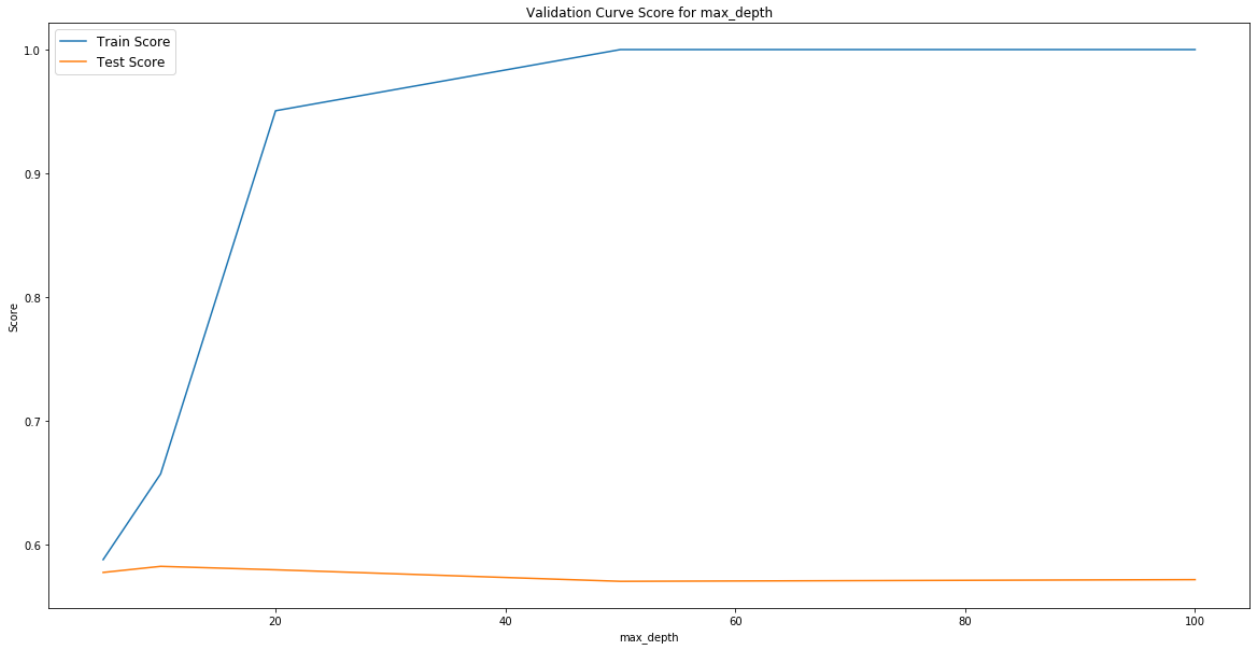

Figure 6: A plot of estimator score against the maximum depth of the decision trees for training and testing data.

From the plot of the 'Train Score' you can see that the values plateau at around the 50 max_depth mark. The 'Test Score' seems to decrease very slowly and stay below the 0.6 mark, but also plateauing at max_depth=50. Therefore for the grid search I will use values of max_depth at 40, 50, 60 and *None*. Where *None* indicates that there is no restriction on the depth of the tree (this method is exhaustive).

n_estimators The n_estimators validation curve was done for the values: *n_estimators*=50, 100, 200, 500. The results are plotted as,

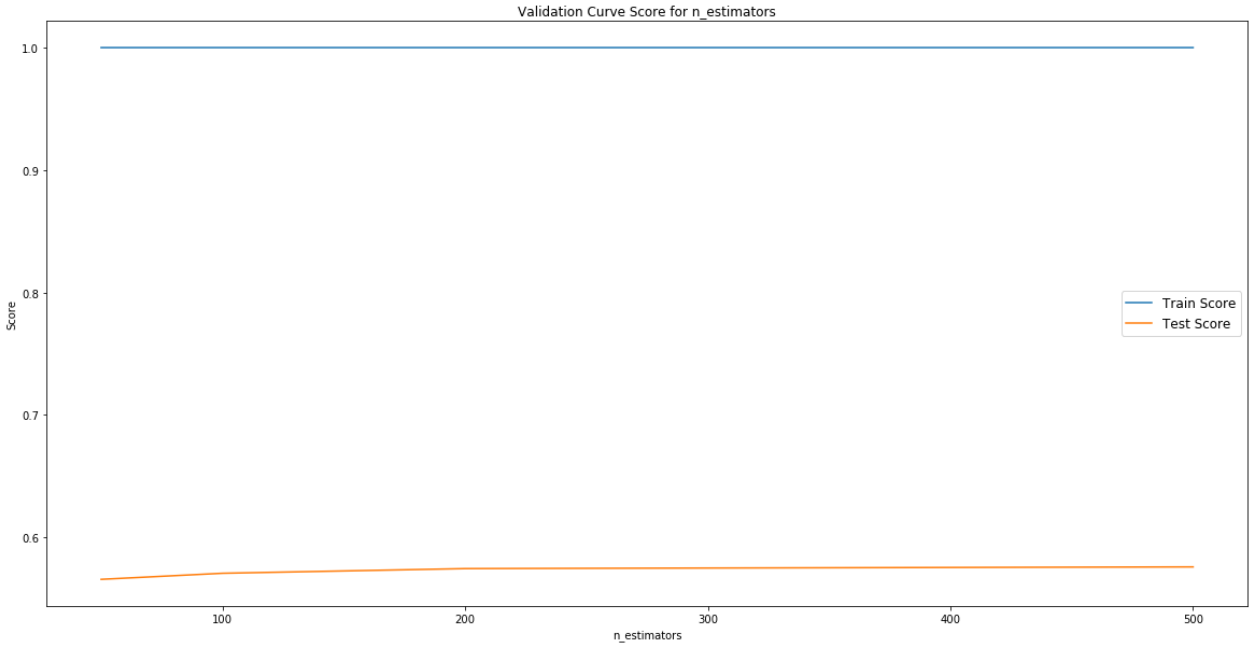

Figure 7: A plot of estimator score against the number of decision tree estimators in the random forest for training and testing data.

The results here are less clear since the Score is perfect for all values in the training data. However, on the testing data, since the score only increase slightly between 100 and 200 estimators (yet the runtime will double) I will choose values about 100. For the grid search I will use values of n_estimators at 80, 100 and 120.

Grid Search Results

To create the grid search I need the input data (described in sections 3.1.2 and 3.1.3) and the hyperparameter values (previously determined). I run the grid search to make (and test) random forest classifiers for each possible model.

From the results I have ranked and displayed the top 10 performing random forests.

kernel	rank_test_score	mean_test_score	std_test_score
entropy_None_sqrt_120	1	0.569933	0.006190
gini_60_sqrt_120	2	0.568789	0.005331
entropy_50_sqrt_120	3	0.568619	0.005801
entropy_40_log2_120	4	0.568558	0.004696
entropy_40_log2_100	5	0.568546	0.004980
gini_40_sqrt_120	6	0.568540	0.005444
gini_60_log2_120	7	0.568333	0.005155
entropy_50_log2_120	8	0.568150	0.004176
entropy_60_sqrt_120	9	0.567950	0.004236
entropy_None_log2_120	10	0.567919	0.005121

Figure 8: Table of random forest algorithms showing; parameters (kernel), rank, mean score and standard deviation.

And so the best combination of hyperparameters (and hence the one I shall be using for the automated trading algorithm) is a random forest with the parameters: *Criterion*=entropy, *max_depth*=None, *max_features*=sqrt (square root of the number of features), and *n_estimators*=120. And from the 'mean_test_score' column in the table I can expect a prediction accuracy of about 57%.

3.3 Results

From the 'Design' section of making a random forest estimator we have seen an initial, simplistic model using only price data from one type of stock (Google) and attempting to predict the next day's price. Understandably this was a model that needed adjusting and we ultimately arrived at a more thorough form of input data, including technical indicators and spanning for over 100 stocks. We will now see the results of both types of models and what this means moving forwards.

3.3.1 Original Model

- **Input data:** OHLC data for Google between 2010 and 2019 (inclusive).
- **Output data:** Next day's close price.

We perform a ***K-fold cross-validation*** on this random forest model.

> **K-fold cross-validation** is an evaluation method for ML models in which you randomly take $1/K$ of the input data out for testing, train on the rest of the data and repeat n times. You arrive at a *mean* and *standard deviation* over all n results.

The *scoring* method here was to measure price error (how incorrect a prediction was). We achieved the results 'mean error'=9.37 and 'standard deviation'=0.84. Meaning, on average, each guess was \$9.37 off the actual value.

Taking the two regression based baseline models *'Constant 1-day change'* and *'No change'* we arrive at a table of results,

Model	Mean error($)
Random Forest	9.37
Constant change	12.14
No change	8.36

Table 1: Results

Showing that the *'No change'* baseline model was the better of the two regression baselines, and also that the random forest failed to beat the baseline model. This not only shows that the random forest process was badly done, but also that the problem setup was incorrect since the best method was to do nothing.

We can also look at a plot of the input variables and their respective *'importances'*. Where a variable's importance is an overall measure of how much a given variable was used in the decision making.

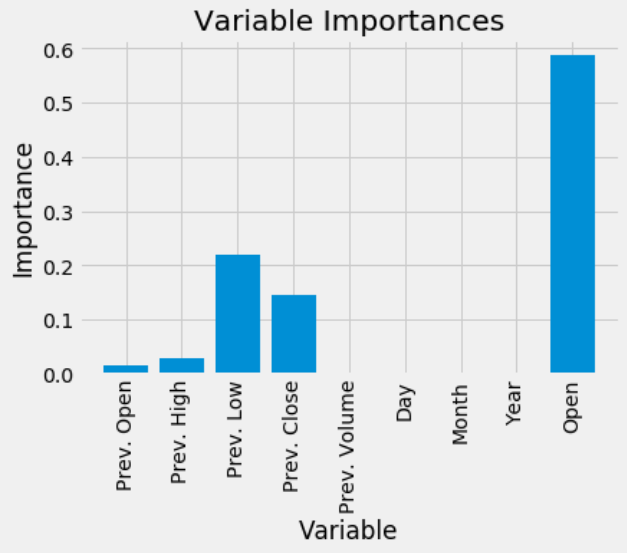

Figure 9: Bar chart showing the *importance* of each input variable.

From this we can see that most input variables were not even used at all, with 'open' being by far the most useful variable.

3.3.2 Improved Model

- **Input data:** Price data and technical indicators for 100 stocks that make up the S&P 100, between 2010 and 2019 (inclusive).
- **Output data:** Prediction of the movement of the price in 10 days time.

This time we only test on the 2019 data (using 2010-2018 for training). We get a test result of 'prediction accuracy'=57.5%. We can also look at the confusion matrix of the results (where a 1 represents 'UP' and a 0 represents 'DOWN').

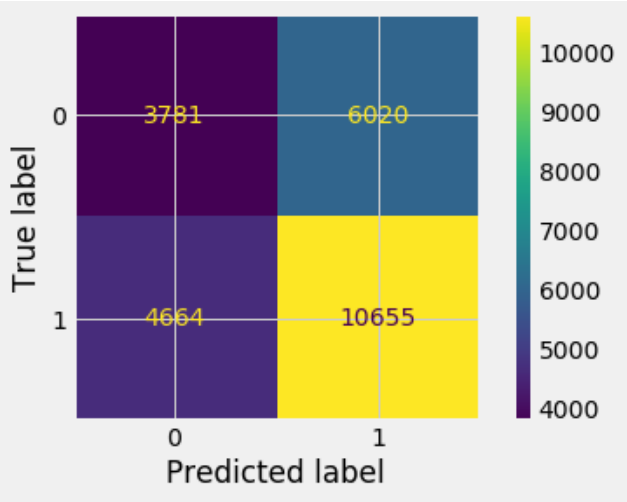

Figure 10: Bar chart showing the *importance* of each input variable.

Showing that it was extremely successful when predicting UP movements, 63.9% accurate, but unsuccessful while attempting to predict DOWN movements, 44.8% accurate. This ultimately shows that the estimator has more difficulty when being tasked with predicting downward trends. This fact can be used when applying it to an automated trading method - i.e. only attempt to predict upward trends.

Taking the two classification baseline models *'Same direction as previous day'* and *'Always UP'* we arrive at a table of results results,

Model	Prediction accuracy (%)
Random Forest	57.5
RF (only UP attempts)	63.9
Same direction	48.6
Always up	51.5

Table 2: Results

Always predicting up was the better of the two baselines. But the random forest estimator successfully beat these baseline models, with the UP success showing comfortably the best approach.

If we now look at an input variable importance plot we find that essentially all input variables have a say in the prediction process and no variable clearly stands out as a 'sole contributor'. This is the sign of a better, more useful, problem setup.

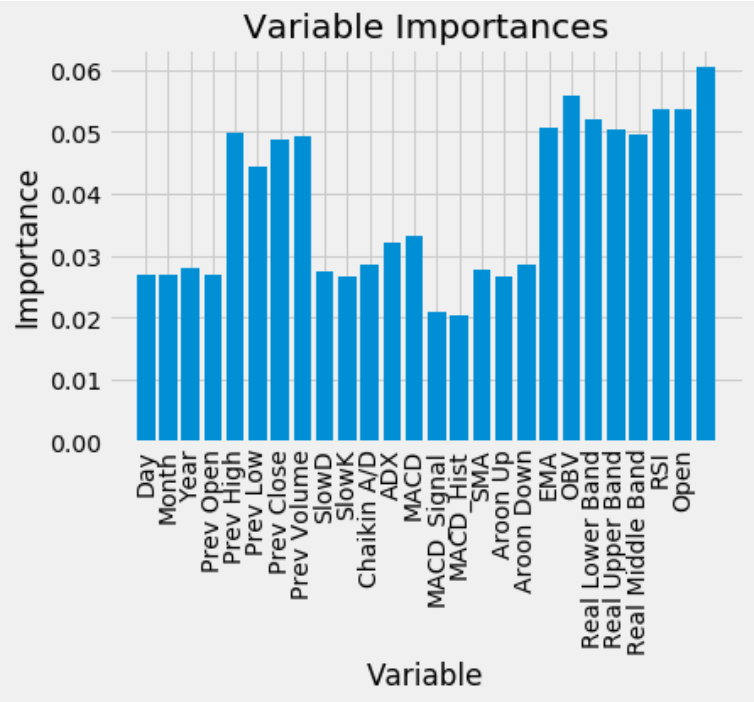

Figure 11: Bar chart showing the *importance* of each input variable.

With the success of these test results, we can now store the best estimator produced by the grid search used to produce these results, and apply it into an automated trading algorithm - as explored in the next section.

4 Automated Trading Algorithm

The next idea is to apply the estimator we have just trained to an automated trading algorithm. One of the key details here is the importance of a useful, sensible trading method. Even with the best estimator, applied incorrectly while ignoring basic rules of finance will not lead to profitable or consistent results.

4.1 Design

At this stage, the objective was to devise a simplistic *initial* algorithm, with the subsequent task of adding new complexities to this algorithm by using basic trading rules and strategies.

The *'Trading: Back-test'* section in *"Implementation of Technical Indicators into a Machine Learning framework for Quantitative Trading"* [10] offered a good basis to build an algorithm on. The way they had tested their machine learnt trader was to recall the 10 best stocks, based on their probabilities of an 'up' movement, purchase them and sell them 10 days later.

4.1.1 Trading Algorithm: Version 1

The algorithm I began with can be described by the following steps:

1. **Input:** Random forest estimator r, price and technical indicator data over a time frame T, amount of initial money/equity X.

2. Use r to predict which 10 stocks are most likely to go up in the next 10 days.
3. Buy as many of these 10 stocks, without going over X.
4. Hold for 10 days.
5. Sell **all** stocks at the new price and update X by total loss/profit.
6. Repeat over the whole duration of T.
7. **Output:** X

Or more explicitly as a flowchart:

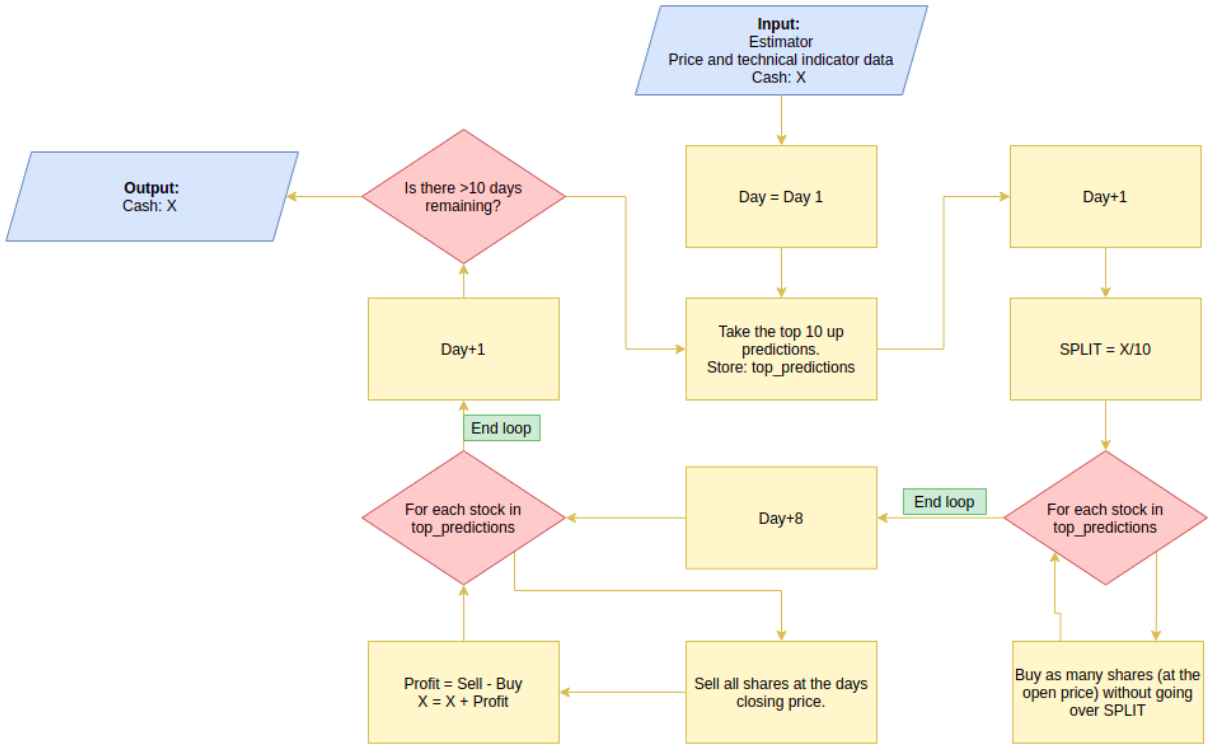

Figure 12: Trading algorithm - Version 1 flowchart.

Although this method was most likely 'over-simplistic', it did offer a good basis for a more developed trading algorithm. It manages to incorporate a basic trading rule of maintaining a *diverse* portfolio of stocks, while also utilising the machine learnt algorithm from the previous section.

4.1.2 Trading Algorithm: Version 2

What we ultimately arrive at is an improved trading algorithm. Using the fundamental ideas developed by 'Version 1' with adjustments and additional functionality based on general rules of finance and trading, and also on advice and research found online.

Storing Trades

The first addition is to store trades in a data frame. This both improves efficiency and flexibility for the algorithm process (where before there was no sort of *'memory'* of the trades). It also acts as a useful resource as once the trading process has been completed it is stored for later perusal. During which, further analysis can be taken to study which stocks were the best performing, which trading decisions won and lost money, and essentially allows further detail on how the ML algorithm or the trading process can be improved in the future.

The data that will be stored on each trade are: ticker, open date, open price, quantity, trade price (open price×quantity), close date, close price, profit, expiry, status.

The *'expiry'* column will track how long the stock has until it is sold - this is initially set to 10 days. And *'status'* is of the form 'open', 'closed', or 'pending' which is where the stock has been selected to purchase, but the purchase has not been executed yet.

These two columns prove useful during the execution of the algorithm.

Stop-Loss

This is a basic rule of trading. It is the idea of setting a price at which you will close the trade (and stop your losses) if the stock's price falls below this value. It is essentially to protect yourself from a sudden crash in the market, where you may not have the funds/patience to wait for the price to recover or simply don't believe it will.

The choice of where to place your stop-loss is an important one. Too close to the original price and it may be triggered after just a small dip before going upwards. Too far from the original price and execution of a stop-loss may become devastating as the loss becomes larger the further you go. I have chosen a stop-loss of 95% of the original price (i.e 5% price drop), however, this is ultimately a value, like hyperparameters, that needs to be tuned and optimised.

The opposite school of thought to a stop-loss is a *'take profit'*, where you close the trade once you reach a price **above** the start price. However, I have chosen to ignore this feature and instead close the trade once 10 days have passed.

Note: it is not unrealistic to implement a stop-loss in this algorithm since it is a feature that is available on most trading platforms.

Money Management - Kelly Criterion

Another key, yet simple, rule of trading is **money management**. This is the budgeting, saving and general oversight on how your capital is spent. Money management is key, where poor money management can lead to debt and financial strain [11]. It can also spoil potentially profitable results from an otherwise sound strategy.

In my Version 1 algorithm, all the equity (cash) was split 10 ways. This, although easy to apply, was quite a risky strategy as a crash in the market, which would hinder all 10 stocks I owned, would have an impact on 100% of my equity.

This leads to the introduction of **Kelly criterion**, developed by John Kelly [12]. It is a money management system that offers a calculated **K%** which indicates how much of your total equity you should place on each trade. The process to make these calculation is as follows,

$$K\% = W - \frac{1 - W}{R} \tag{9}$$

where $K\%$ = Kelly percentage, W = win probability and R = win/loss ratio.

$$W = \frac{\text{win trades}}{\text{win trades} + \text{loss trades}} \tag{10}$$

$$R = \frac{\text{Average gain on wins}}{\text{Average loss on losses}} \tag{11}$$

Regardless of calculations, I have capped $K\% < 10\%$ due to increased risk at higher values. If we weren't to impose a cap, a string of successful results could falsely guide us into a relaxed $K\%$ causing too much equity to be staked on a single trade.

I have implemented a *dynamic Kelly criterion* where after 40 trades have been made, a $K\%$ is calculated and then continuously updated after each trade. Before this 40 day point, I cautiously enforce $K\% = 5\%$.

Confidence Minimum

In the previous algorithm, I was always collecting the top 10 predicted stocks to move upwards at each iteration. To help aid the robustness of the algorithm, I am now collecting *at most* the top n stocks (n isn't always 10 since stocks are now being bought and sold at different times and prices due to new changes) and filtering where the confidence of an 'UP' movement $\geq 60\%$. This confidence is provided by the random forest estimator.

This change means that if the market is in decline, or at a state which may indicate decline, the trading algorithm is not **forced** to buy n stocks, only those it has deemed confident enough to buy.

As with the stop-loss, this 60% value is one that will need optimising.

With these changes, we can then test the algorithm by running it alongside the 2019 data we used while testing the random forest algorithm.

4.2 Results

- **Input data:** Random forest estimator, price and technical indicator data used to test RF estimator, cash $X = \$1,000,000$.
- **Output data:** Return on investment, $\left(\frac{\text{CASH}_{\text{end}}}{\text{CASH}_{\text{start}}} - 1\right) \times 100\%$.

The selection of $X = \$1,000,000$ was an arbitrary selection of a large round number, but also with the aim to pick an amount that allows the algorithm to buy a variety of all stocks multiple times.

The results of both versions 1 and 2 of the trading algorithms can be seen in the following table:

Algorithm Version	Return on Investment (%)
1	16.3
2	25.6

Table 3: Automated trading algorithm results on 2019 data.

Showing that the 2nd version performed better, as hoped. This 25.6% return on investment states that for our hypothetical \$1,000,000 investment into the algorithm, we would have gained \$256,000 over the year 2019.

As well as performing better on this 2019 data, we would also expect version 2 to be less risky and more robust against a variety of data due to its inclusion of more rigorous trading methods. We can test this by re-running the algorithm on 2010 data (here I am just selecting the first year of data I had available). The results were:

Algorithm Version	Return on Investment (%)
1	88.2
2	128.9

Table 4: Automated trading algorithm results on 2010 data.

Here the returns are much higher. This is because, unlike the 2019 data which was *unseen*, this was *seen* data, meaning it was present during the learning process. Therefore these results must be taken lightly when using it to evaluate the overall algorithm. Despite this, however, it still acts as a good comparison between algorithms showing the second version's ability to produce higher returns on different data, backing up our claims that it is the better algorithm.

We can also utilise the saved data frame when running the version 2 algorithm on the 2019 data. For example, the first entry is as follows,

ticker	open date	open price	quantity	trade price	close date	close price	profit	expiry	status
CSCO	03/01	42.30	1182	49998.60	16/01	43.96	1962.12	0	closed

This can be interpreted as; On the 3rd of January the algorithm executed a trade of 1182 shares of CSCO (Cisco Systems) at $42.40/share resulting in a $49,989.60 trade, it was then closed on the 16th of January at $43.96/share resulting in a $1,962.12 profit. Screenshots of the actual document collected from this data frame can be found in the Appendices.

Some further things of interest from the data frame:

- There was 266 total trade made by the automated algorithm, resulting in slightly over one trade made per working day (US).
- The largest winning trade was BIIB (Biotech), which closed on 16th October resulting in $32,855.47 profit
- Interestingly, the largest losing trade was also BIIB, closing on 18th March resulting in $32,432.72 loss. This basically cancels out the profit gained later in the year.
- There were 10 trades left in an open position. They had not been closed before the year had run out.

We can also gain a rundown of results from an array that was used to continuously update the $K\%$. It shows 164 win trades resulting in $590,206.12 profit, and 93 loss trades resulting in -$334,201.28 loss.

It is clear from these results that it is impossible (and unnecessary) to avoid losing trades. However, when you can make the wins outweigh the losses, and minimise these losses where they do occur, you can produce profitable results.

4.3 Analysis

One thing you may have noticed from these results is the lack of baseline models to compare our results to. This is due to the increased difficulty in selecting an appropriate baseline model for this type of problem because a trading fund/strategy can take many different attributes. For example, if we only focus on returns we could look at *'risk-free investment rate'*, this is a national average

of risk-free investment yearly returns (this includes bank account interest rates, treasury yields, government bonds). For 2019 the risk-free investment rate was 2.1% (according to *"statista.com"* [13]), being much lower than the returns achieved by our algorithm. Alternatively, we could look at an investment by focusing on just one of the stocks that we were analysing. For example, if we were to only buy Apple stocks (AAPL) at the start of the year and closed it at the end of the year we would have made a 92.1% return, much higher than our algorithm.

However, this return would vary greatly between our selection of stocks, with some possibly giving negative results. This leads us onto another attribute, **risk**. As with risk-free investments you can (as the name suggests) gain this interest without any (or with negligible) risk, so that 2.1% return is essentially guaranteed. This idea of risk is then slightly increased for our algorithm since the stock markets can move up **or** down. However, we are at least *diversifying* our strategy by having different amounts of our equity (cash) in different stocks, whereas the method caused by only purchasing one stock can be seen as an *'all eggs in one basket'* approach, being the riskiest method. These all follow the idea of **'risk = reward'**, the more you risk, the more you can gain and also lose.

The following are some additional attributes a *portfolio* can have. Here portfolio is just a selection of stocks, either managed by a fund manager or an individual - or an algorithm in this case.

- **Standard deviation:** The volatility of a portfolio, measured by

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}, \tag{12}$$

  where x_i is the return (%) on day i, $\bar{x}$ is the average return of each day over n, and n is the number of trading days we are looking at. This will typically be done over at least one year's worth of data.

- **Sharpe ratio:** This is a measurement calculated by

$$S.R. = \frac{R_p - R_f}{\sigma}, \tag{13}$$

  where R_p is the return of the portfolio, R_f is the risk-free rate (discussed previously) and σ is the standard deviation of the portfolio as just described. It is a measure of the portfolio's performance in relation to the excess risk that was taken by it. It works on the assumption that investment returns are normally distributed.

- **Alpha (α):** A term used to describe a portfolio's ability to 'beat the market'.

- **Beta (β):** Works similarly to α and measures the *volatility* of a portfolio in comparison to the rest of the market (as apposed to performance).

These attributes would act as good further analysis for our algorithm. However, these would all require more testing data for the algorithm in order to have a good variety of results over various market environments. This lack of testing data is one of the key drawbacks of this problem setup and will be discussed further in the Evaluation section 5.

5 Evaluation

5.1 Pros

Fundamentally, the clear positive to take from this project is the methods' abilities to produce successful results while (where applicable) beating the baseline models I had devised. One of the original aims was to develop a completely automated trading algorithm with no human input. While there were areas I had tweaked throughout the process, what we finally end up at is a self-sufficient algorithm that is unaided. Although further tweaks would have to be made if this algorithm was to be run in real-time.

An additional 'pro' of the algorithm is the ease to update it for different stocks or different dates. The algorithm will update itself to accommodate whichever stocks make up the S&P 100 at the given time. However, this process would involve a 're-learning' stage where a new random forest would need to be made which could lead to a very long machine learning process. This brings us on to the 'cons'.

5.2 Cons

As implied, one main disadvantage of these methods is that it contained multiple aspects of code which had long execution times. This includes the pre-processing and calculations of all technical indicators for a large number of stocks, and also the grid search for finding and training the best random forest algorithm. There may exist some areas of the code which can be further optimised in order to gain minimum runtime, although this is something that would need to be explored deeper. Moreover, it is expected that any machine learning algorithm, especially those handling vast amounts of data, take a while to run due to the exhaustive and meticulous nature of the process - this is something that ultimately cannot be avoided.

As mentioned briefly in the Analysis section 4.3, although a year does offer a reasonably large amount of data to test our algorithm on, it can be seen as offering only a small window into the past of the stock market. Thus, to improve the validity of the methods I would need to test on multiple years. The difficulty with this is that the years 2010-2018 were seen data used in the machine learning process, meaning they are ineligible for fair testing. This leaves only 2020 as a full year to test, which was obviously quite an anomaly for the stock market, although it could be argued that this is good preparation in order to test an algorithm when faced with turbulent conditions. Clearly, the stock market has existed long before 2010, however, the shape of the market has changed vastly and so testing on data before my 2010-2018 training data would involve providing data that is not a fair reflection on what has been learnt by our estimators - a separate estimator would have to be produced for this purpose.

There is also the issue caused by **backtesting** (testing a strategy on existing data). I have already discussed 'survivorship bias' in the Literature Review section 2, but I must also consider the difficulty of generalising results of a model that has only been executed on *existing data*. To make a completely natural study I would need to edit the methods to work in real-time.

5.3 Final Thoughts

One of the key elements I have taken from this project is the importance of useful input data. It is vital to understand what drives the price of an asset and use this to your advantage when approaching the stock market. I appreciate there is a complete side of stock market analysis I have not even touched upon. This includes company success, trader sentiment and even global news/economy.

I would also like to note that, with 2019 being a key year for my testing of the algorithm, 2019 overall was successful for the stock market hinting that any application of a trading strategy would be favourable. On the other hand, in the past 30 years, the general US stock market has gone up 23 out of 30 times (according to *"Yahoo finance"* [14]) suggesting that it may not be unreasonable to test on successful data.

I will end the evaluation by making note of aspects which (if I had more time) I would be interested in applying to my current methods in order to improve results, analysis and applicability.

- Repeat the algorithm over 2020 data. This would provide multiple new aspects:
    1. Test the robustness of the algorithm over a new data source.
    2. Provide enough data to perform further analysis to measure attributes of the algorithm described in the Analysis section 4.3.
    3. Test the algorithm against a market in recession.
- Add the possibility to ***short*** stocks. Shorting is the action of selling a share you don't own with the contractual obligation to buy the share at a later date. This is used when an investor believes the price of a stock will go down and would prove to be useful when handling 2020 data (or any market in recession). Shorting is available on most markets.
- Apply transaction fees. This is a key part of any trading process, one that has been completely ignored in my models. The addition of transaction fees would add authenticity to the problem.
- Apply the algorithm in an 'ongoing' nature, utilising data from the present day to see how I could evolve this model to work as an active trader.

6 Conclusion

Overall I have thoroughly enjoyed this project and found it both challenging and stimulating. I am extremely pleased to have achieved positive results for both the machine learning and automated trading methods. I am also happy and encouraged by everything I have learnt in both computer science (with a real-world machine learning application and large data management) and in finance (mostly with the use of technical analysis). Despite this, I believe the results I have produced should be taken with a pinch of salt since there is only a small sample of results. I consider this as the main drawback of my project, caused by time limitations and also limitations of available data and past stock history - as discussed in the Evaluation section 5. Additionally, I think that the fundamental design of the project is based on an oversimplification of what is in fact a very complex construct, to focus solely on the technical side of a stock is currently very *'tunnel vision'*, although still proved to be an interesting concept for the purpose of this project. To conclude, I have found it fascinating to see the results that I could produce, as well as others I have come across during my research, which begs the question of how much further will the use of machine learning in the stock market go in the next 5 to 10 years? Will the ability of precise, instant, computational and emotionless methods provided by artificial intelligence overtake the need for people in suits at Wall Street?

7 Appendices

GitLab Link

GitLab link containing project repository:

`https://git-teaching.cs.bham.ac.uk/mod-ug-proj-2020/bxn769`

See the 'README.md' file for how the code works and how to re-run any aspects of code, if applicable.

Technical Indicators

10 Day Change
The 10 day price change %,

$$10\_change = \left(\frac{\text{Price}_{\text{now}}}{\text{Price}_{\text{now}-10}} - 1 \right) \times 100\% \tag{14}$$

5 Day Change
The 5 day price change %,

$$5\_change = \left(\frac{\text{Price}_{\text{now}}}{\text{Price}_{\text{now}-5}} - 1 \right) \times 100\% \tag{15}$$

Simple Moving Average
A continuous average taken over the price's average of the last n days. Used to indicate if the price is too high or too low in comparison to past prices.

$$SMA = \frac{P_1 + P_2 + \cdots + P_n}{n} \tag{16}$$

where P_i is the price on day i, and n is the number of days.

Exponential Moving Average
Similar to SMA, but places weights on more recent prices.

$$EMA = \text{Price}_{\text{today}} \times \text{multiplier} + EMA_{\text{yesterday}} \times (1 - \text{multiplier}) \tag{17}$$

Bollinger Bands
Bollinger bands are 3 lines on the price chart showing the simple moving average (SMA) and an upper and lower bound. The price is expected to remain between these bands.

$$Bol_{Upper} = SMA + 2 \times \sigma \tag{18}$$

$$Bol_{Lower} = SMA - 2 \times \sigma \tag{19}$$

where σ is the standard deviation of the price over the last n days.

Aroon Indicator
The Aroon indicator shows two lines, up and down. When the up line is above the down line, it indicates an upward price trend. When the down line is above the up line, it indicates a downward trend.

$$A\_UP = \frac{25 - \text{Periods since 25 period high}}{25} \times 100 \tag{20}$$

$$A\_DOWN = \frac{25 - \text{Periods since 25 period low}}{25} \times 100 \tag{21}$$

MACD

The 26-period EMA subtracted by the 12-period EMA. It creates signals when it crosses through a signal line.

$$MACD = EMA(12) - EMA(26) \tag{22}$$

OBV Indicator

On-balance volume (OBV) indicates the momentum of a stocks price using volume changes. Volume is the amount of a given stock traded on a trading day.

$$OBV = OBV_{\text{prev}} + \begin{cases} \text{volume}, & \text{if price} > \text{price}_{\text{prev}} \\ 0, & \text{if price} = \text{price}_{\text{prev}} \\ -\text{volume}, & \text{if price} < \text{price}_{\text{prev}} \end{cases} \tag{23}$$

Stochastic Indicator

Stochastic indicator (or oscillator) K, is a measure of a calculation about a mean price, used to indicate when a stock is overvalued or undervalued.

$$K = \left(\frac{C - 14 \text{ day price low}}{14 \text{ day price high} - 14 \text{ day price low}} \right) \times 100 \tag{24}$$

where C is the most recent closing price.

ADX Indicator

ADX is used to determine the strength of a given trend. The stronger a trend is, the less risky it is to trade on this trade.

Chaikin Oscillator

The Chaikin oscillator applied MACD to a signal line. A movement above the signal line is signs of an upward trend.

$$CO = (\text{3-day } EMA \text{ of } ADL) - (\text{10-day } EMA \text{ of } ADL) \tag{25}$$

where

$$ADL = M(\text{Period} - 1) + M(\text{Period}) \tag{26}$$

where M is a multiplier and Period is either 3 or 10.

Note: definitions and calculations taken from `https://www.investopedia.com/`

Validation Curve Code

```
'''
Validation curve
'''
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import validation_curve
from sklearn.model_selection import TimeSeriesSplit

#Create validation curve for the Random Forest Classifier
rf = RandomForestClassifier()
train_scoreNum, test_scoreNum = validation_curve(rf,
                                X = data, y = labels,
                                param_name = 'n_estimators',
                                param_range = [50,100,200,500],
                                cv = TimeSeriesSplit(n_splits = 3),
                                verbose=3)

train_scores_mean = np.mean(train_scoreNum, axis=1)
train_scores_std = np.std(train_scoreNum, axis=1)
test_scores_mean = np.mean(test_scoreNum, axis=1)
test_scores_std = np.std(test_scoreNum, axis=1)

plt.figure(figsize = (20,10))
plt.plot([50,100,200,500],train_scores_mean)
plt.plot([50,100,200,500],test_scores_mean)
plt.legend(['Train Score','Test Score'], fontsize = 'large')
plt.xlabel('n_estimators')
plt.ylabel('Score')
plt.title('Validation Curve Score for n_estimators', fontsize = 'large')
```

Figure 13: Python code for producing a validation curve plot.

Algorithm Results Document

Results showing the first 40 and last 40 data entries in our results data frame for the trading algorithm: version 2 on 2019 data.

	Ticker	Open_date	Open_price	Quantity	Trade_price	Close_date	Close_price	Profit	Expiry	Status
0	CSCO	2019-01-03 00:00:00	42.2999992370606	1182	49998.5990982056	2019-01-16 00:00:00	43.9599990844727	1962.11981964111	0	closed
1	MMM	2019-01-03 00:00:00	188.279998779297	265	49894.1996765137	2019-01-16 00:00:00	189.479995727539	317.999191284172	0	closed
2	MSFT	2019-01-03 00:00:00	100.099998474121	499	49949.8992385864	2019-01-16 00:00:00	105.379997253418	2634.71939086913	0	closed
3	MA	2019-01-03 00:00:00	187.5	266	49875	2019-01-16 00:00:00	197.770004272461	2731.8211364746	0	closed
4	LIN	2019-01-03 00:00:00	154.059997558594	324	49915.4392089844	2019-01-16 00:00:00	157.240005493164	1030.32257080078	0	closed
5	DD	2019-01-03 00:00:00	77.1412200927734	648	49987.5106201172	2019-01-16 00:00:00	78.8491439819336	1106.73468017578	0	closed
6	JNJ	2019-01-03 00:00:00	128.139999389648	390	49974.5997619629	2019-01-16 00:00:00	128.039993286133	-39.0023803711083	0	closed
7	MCD	2019-01-03 00:00:00	175.449996948242	284	49827.7991333008	2019-01-16 00:00:00	179.350006103516	1107.60260009766	0	closed
8	VZ	2019-01-03 00:00:00	56.2999992370605	888	49994.3993225098	2019-01-16 00:00:00	57.0800018310547	692.642303466797	0	closed
9	ADBE	2019-01-03 00:00:00	220.880004882813	226	49918.8811035156	2019-01-16 00:00:00	241.949996948242	4761.81820678711	0	closed
10	AIG	2019-01-17 00:00:00	42.8499984741211	1185	50777.2481918335	2019-01-31 00:00:00	43.2299995422363	450.301265716553	0	closed
11	BK	2019-01-17 00:00:00	49.939998626709	1017	50788.978603363	2019-01-31 00:00:00	52.3199996948242	2420.4610862732	0	closed
12	ADBE	2019-01-17 00:00:00	240.130004882812	211	50667.4310302734	2019-01-31 00:00:00	247.820007324219	1622.59051513672	0	closed
13	CL	2019-01-17 00:00:00	62.1199989318848	818	50814.1591262817	2019-01-31 00:00:00	64.6800003051758	2094.08112335204	0	closed
14	MMM	2019-01-17 00:00:00	188.559997558594	269	50722.6393432617	2019-01-31 00:00:00	200.300003051758	3158.06147766114	0	closed
15	GS	2019-01-17 00:00:00	195.949996948242	259	50751.0492095947	2019-01-31 00:00:00	198.009994506836	533.539367675789	0	closed
16	AAPL	2019-01-17 00:00:00	38.5499992370605	1318	50808.8989944458	2019-01-31 00:00:00	41.6100006103516	4033.08180999756	0	closed
17	AMT	2019-01-17 00:00:00	163.779998779297	310	50771.799621582	2019-01-31 00:00:00	172.839996337891	2808.59924316406	0	closed
18	CHTR	2019-01-17 00:00:00	292.320007324219	173	50571.3612670898	2019-01-31 00:00:00	331.049987792969	6700.28662109375	0	closed
19	EXC	2019-01-17 00:00:00	45.8300018310547	1108	50779.6420288086	2019-01-31 00:00:00	47.7599983215332	2138.4361114502	0	closed
20	VZ	2019-02-01 00:00:00	55.2099990844727	943	52063.0291366577	2019-02-14 00:00:00	54.0299987792969	-1112.74028778075	0	closed
21	EXC	2019-02-01 00:00:00	47.6699981689453	1093	52103.3079986572	2019-02-14 00:00:00	47.8600006103516	207.672668457031	0	closed
22	MRK	2019-02-01 00:00:00	75.9700012207031	685	52039.4508361816	2019-02-14 00:00:00	78.9400024414063	2034.45083618164	0	closed
23	MDLZ	2019-02-01 00:00:00	45.8600006103516	1136	52096.9606933594	2019-02-14 00:00:00	47.3199996948242	1658.55895996094	0	closed
24	HON	2019-02-01 00:00:00	148.699996948242	350	52044.9989318848	2019-02-14 00:00:00	150.520004272461	637.002563476563	0	closed
25	COP	2019-02-01 00:00:00	68.3399963378906	762	52075.0772094727	2019-02-14 00:00:00	69.0999984741211	579.121627807632	0	closed
26	CRM	2019-02-01 00:00:00	152.399993896484	341	51968.3979187012	2019-02-14 00:00:00	159.470001220703	2410.87249755859	0	closed
27	CL	2019-02-01 00:00:00	64.9000015258789	802	52049.8012237549	2019-02-14 00:00:00	65.4499969482422	441.096328735366	0	closed
28	AMZN	2019-02-01 00:00:00	1638.88000488281	31	50805.2801513672	2019-02-14 00:00:00	1622.65002441406	-503.12939453125	0	closed
29	LLY	2019-02-01 00:00:00	120.319999694824	433	52098.5598678589	2019-02-14 00:00:00	120.75	186.190132141113	0	closed
30	TMUS	2019-02-15 00:00:00	70.6500015258789	742	52422.3011322022	2019-03-01 00:00:00	72.3399963378906	1253.9761505127	0	closed
31	LOW	2019-02-15 00:00:00	102.73999786377	510	52397.3989105225	2019-03-01 00:00:00	103.959999084473	622.200622558601	0	closed
32	GILD	2019-02-15 00:00:00	66.7099990844727	786	52434.0592803955	2019-03-01 00:00:00	66.0100021362305	-550.197601318359	0	closed
33	CAT	2019-02-15 00:00:00	134.100006103516	391	52433.1023864746	2019-03-01 00:00:00	137.470001220703	1317.66809082031	0	closed
34	TGT	2019-02-15 00:00:00	72.5599975585938	722	52388.3182373047	2019-03-01 00:00:00	72.9400024414062	274.363525390625	0	closed
35	PFE	2019-02-15 00:00:00	40.2087287902832	1304	52432.1823425293	2019-03-01 00:00:00	41.1385192871094	1212.44680786133	0	closed
36	COP	2019-02-15 00:00:00	69.9400024414063	749	52385.0618286133	2019-03-01 00:00:00	68.9300003051758	-756.491600036621	0	closed
37	PYPL	2019-02-15 00:00:00	95	552	52440	2019-03-01 00:00:00	98.8000030517578	2097.60168457031	0	closed
38	AMT	2019-02-15 00:00:00	177.279998779297	295	52297.5996398926	2019-03-01 00:00:00	177.919998168945	188.799819946289	0	closed
39	CVS	2019-02-15 00:00:00	68.2600021362305	768	52423.681640625	2019-02-20 00:00:00	64.2200012207031	-3102.72070312501	0	closed
40	BLK	2019-02-21 00:00:00	436.600006103516	239	104347.40145874	2019-03-06 00:00:00	429.859985351563	-1610.8649597168	0	closed

Figure 14: First 40 data entries of our trading algorithm.

227	GE	2019-11-06 00:00:00	10.9300003051758	10376	113409.683166504	2019-11-19 00:00:00	11.5	5914.31683349611	0	closed
228	MET	2019-11-06 00:00:00	48.3600006103516	2345	113404.201431274	2019-11-19 00:00:00	49.1500015258789	1852.55214691161	0	closed
229	COST	2019-11-06 00:00:00	299.970001220703	378	113388.660461426	2019-11-19 00:00:00	302.239990234375	858.055847167969	0	closed
230	HD	2019-11-07 00:00:00	234	498	116532	2019-11-20 00:00:00	220.899993896484	-6523.80303955077	0	closed
231	AMT	2019-11-07 00:00:00	206.710006713867	564	116584.443786621	2019-11-20 00:00:00	217.899993896484	6311.15277099611	0	closed
232	CVS	2019-11-07 00:00:00	71.9400024414063	1621	116614.74395752	2019-11-20 00:00:00	74.9199981689453	4830.57307434082	0	closed
233	FB	2019-11-12 00:00:00	190	613	116470	2019-11-25 00:00:00	199.789993286133	6001.26588439941	0	closed
234	ABT	2019-11-13 00:00:00	83.9599990844727	1402	117711.918716431	2019-11-26 00:00:00	85.4199981689453	2046.91871643066	0	closed
235	FB	2019-11-13 00:00:00	194.699996948242	604	117598.798156738	2019-11-26 00:00:00	198.970001220703	2579.08258056641	0	closed
236	HD	2019-11-14 00:00:00	235	502	117970	2019-11-20 00:00:00	220.899993896484	-7078.20306396483	0	closed
237	BKNG	2019-11-20 00:00:00	1846.06005859375	64	118147.84375	2019-12-04 00:00:00	1921.53002929688	4830.078125	0	closed
238	ADBE	2019-11-20 00:00:00	299.390014648437	397	118857.83581543	2019-12-04 00:00:00	302.510009765625	1238.63806152344	0	closed
239	V	2019-11-20 00:00:00	182.309997558594	652	118866.118408203	2019-12-04 00:00:00	180.600006103516	-1114.91442871094	0	closed
240	TGT	2019-11-21 00:00:00	126.580001831055	938	118732.041717529	2019-12-05 00:00:00	124.680000305176	-1782.20143127441	0	closed
241	JNJ	2019-11-21 00:00:00	135.940002441406	873	118675.622131348	2019-12-05 00:00:00	139.559997558594	3160.25573730475	0	closed
242	BKNG	2019-11-21 00:00:00	1851.33996582031	64	118485.7578125	2019-12-05 00:00:00	1904.21997070313	3384.3203125	0	closed
243	AMZN	2019-11-21 00:00:00	1743	68	118524	2019-12-05 00:00:00	1740.47998046875	-171.361328125	0	closed
244	ADBE	2019-11-26 00:00:00	305	391	119255	2019-12-10 00:00:00	304.170013427734	-324.524749755859	0	closed
245	MCD	2019-11-27 00:00:00	194.259994506836	616	119664.156616211	2019-12-11 00:00:00	194.720001220703	283.364135742173	0	closed
246	MDLZ	2019-11-27 00:00:00	52.7299995422363	2272	119802.558959961	2019-12-11 00:00:00	53.9300003051758	2726.40173339844	0	closed
247	PEP	2019-12-05 00:00:00	136.190002441406	883	120255.772155762	2019-12-18 00:00:00	135.970001220703	-194.26107788083	0	closed
248	GOOG	2019-12-05 00:00:00	1327	90	119430	2019-12-18 00:00:00	1351.91003417969	2241.90307617187	0	closed
249	JNJ	2019-12-05 00:00:00	139.350006103516	863	120259.055267334	2019-12-18 00:00:00	143.190002441406	3313.91683959961	0	closed
250	LLY	2019-12-06 00:00:00	119.970001220703	1006	120689.821228027	2019-12-19 00:00:00	130.850006103516	10945.2849121094	0	closed
251	NKE	2019-12-06 00:00:00	96.5599975585938	1250	120699.996948242	2019-12-19 00:00:00	101.150001525879	5737.50495910643	0	closed
252	BK	2019-12-06 00:00:00	49.8300018310547	2424	120787.924438477	2019-12-19 00:00:00	50.8499984741211	2472.47186279297	0	closed
253	DHR	2019-12-06 00:00:00	147.710006713867	817	120679.07548523	2019-12-19 00:00:00	150.440002441406	2230.40650939941	0	closed
254	CSCO	2019-12-11 00:00:00	44.3600006103516	2722	120747.921661377	2019-12-24 00:00:00	47.7799987792969	9309.23501586916	0	closed
255	CSCO	2019-12-12 00:00:00	44.5499992370606	2717	121042.347927094	2019-12-26 00:00:00	47.8499984741211	8966.09792709351	0	closed
256	HON	2019-12-12 00:00:00	176.339996337891	686	120969.237487793	2019-12-26 00:00:00	176.880004882813	370.445861816406	0	closed
257	CSCO	2019-12-19 00:00:00	47.1500015258789	2578	121552.703933716				1	open
258	F	2019-12-19 00:00:00	9.55000019073486	12732	121590.602428436				1	open
259	CL	2019-12-19 00:00:00	67.2699966430664	1807	121556.883934021				1	open
260	SPG	2019-12-20 00:00:00	146.270004272461	845	123598.15361023				2	open
261	MDLZ	2019-12-20 00:00:00	55.1100006103516	2245	123721.951370239				2	open
262	CRM	2019-12-20 00:00:00	164.729995727539	751	123712.226791382				2	open
263	CSCO	2019-12-20 00:00:00	48.25	2564	123713				2	open
264	USB	2019-12-26 00:00:00	59.8600006103516	2082	124628.521270752				5	open
265	AMZN	2019-12-27 00:00:00	1882.92004394531	66	124272.722900391				6	open
266	FDX	2019-12-27 00:00:00	153.070007324219	820	125517.406005859				6	open

Figure 15: Last 40 data entries of our trading algorithm.

References

[1] *The Revolutionary Way Of Using Artificial Intelligence In Hedge Funds – The Case Of Aidyia*. en. https://bernardmarr.com/default.asp?contentID=1823.

[2] W. Wu et al. "A Cross-Sectional Machine Learning Approach for Hedge Fund Return Prediction and Selection". In: (2019). DOI: 10.2139/ssrn.3238466.

[3] Guillaume Coqueret and Tony Guida. *Machine Learning for Factor Investing: R Version*. en. CRC Press, Aug. 2020. ISBN: 978-1-00-017676-6.

[4] Omer Berat Sezer, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu. "Financial Time Series Forecasting with Deep Learning : A Systematic Literature Review: 2005-2019". In: *arXiv:1911.13288 [cs, q-fin, stat]* (Nov. 2019). arXiv: 1911.13288 [cs, q-fin, stat].

[5] Rajashree Dash and Pradipta Kishore Dash. "A Hybrid Stock Trading Framework Integrating Technical Analysis with Machine Learning Techniques". en. In: *The Journal of Finance and Data Science* 2.1 (Mar. 2016), pp. 42–57. ISSN: 2405-9188. DOI: 10.1016/j.jfds.2016.03.002.

[6] Will Koehrsen. *Random Forest in Python*. en. https://towardsdatascience.com/random-forest-in-python-24d0893d51c0. Jan. 2018.

[7] *Free Stock APIs in JSON & Excel — Alpha Vantage*. https://www.alphavantage.co/.

[8] *Yahoo Finance – Stock Market Live, Quotes, Business & Finance News*. en-GB. https://uk.finance.yahoo.com/.

[9] James Chen. *Technical Indicator Definition*. en. https://www.investopedia.com/terms/t/technicalindicator.asp.

[10] Modishubham. *Implementation of Technical Indicators into a Machine Learning Framework for Quantitative Trading*. en. https://towardsdatascience.com/implementation-of-technical-indicators-into-a-machine-learning-framework-for-quantitative-trading-44a05be8e06. Dec. 2020.

[11] James Chen. *What Is Money Management?* en. https://www.investopedia.com/terms/m/moneymanagement.asp.

[12] Will Kenton. *Understanding the Kelly Criterion*. en. https://www.investopedia.com/terms/k/kellycriterion.asp.

[13] *Risk Free Rate UK 2020*. en. https://www.statista.com/statistics/885750/average-risk-free-rate-united-kingdom/.

[14] *Dow Jones Industrial Average (^DJI) Historical Data - Yahoo Finance*. en-US. https://finance.yahoo.com/quote/%5EDJI/history/.