Capstone Project
Machine Learning Engineer Nanodegree

James Edwards, October 2017

# I. Definition

## Project Overview

For this capstone project, I intend to investigate the potential uses of machine learning in Investment and Trading. As I have worked in this industry for 15 years, I have some knowledge and expertise to apply to this domain. The traditional approach to portfolio management relies on a portfolio manager and/or research analysts to generate an investment thesis for a particular security or securities and build a portfolio which takes an appropriate level of risk in order to achieve a client's long-term performance objectives. This process is reviewed regularly and changes to the portfolio may take place if the investment thesis for a security changes. This traditional approach attempts to "beat the market" in most cases, although the rise of "passive" fund management, where the portfolio only attempts to match the performance of a given stock market index, have grown to significant prominence in recent years.

In this project, I would like to explore whether we can use machine learning techniques to predict the direction of securities for the US stock market.

## Problem Statement

The problem for this project is whether a machine learning model can deliver better performance than a simple "buy and hold" strategy (e.g. where we purchase a security, hold it for a specific length of time and then measure the percent change in price over that time period). As we are simply trying to predict whether a stock price will be higher or lower at some point in the future, we can treat this as a supervised learning problem and label our data. The labels will be "1" and "0" for higher and lower respectively. We will therefore label each *t* datapoint with the sign of the return for the datapoint at *t+1*.

The solution is for the learning algorithm to correctly identify whether a given stock price will be higher (+1) or lower (0) 20 days forward. We use 20 days in this instance to proxy a holding period of 1 month (or 4 weeks with a 5 day working week).

## Metrics

As we have labelled our training and testing data using "perfect foresight", we can compare the accuracy of our predictions to these labels. The difficulty is knowing how well our learner has performed. Would an accuracy score of over 50% be sufficient? In this instance we can benchmark our score versus just holding the security with no buys or sells after the initial purchase (e.g. each datapoint is labelled with a 1). In order for our learner to be making better predictions than the market, the learner accuracy score must therefore be greater than the benchmark accuracy score. Further, the resulting return from the learner strategy should also be greater than simply buying and holding the security for the period in question.

The relevant metrics for this project will comprise F1 scoring of the label prediction accuracy as well as the relevant financial performance of the predicted labels. The metrics we will use are defined as follows:

**F1 score for label prediction:**

We should aim to capture not only the true positive results, but also the number of false negatives and false positives (Type I and Type II errors). The F1 score formula is:

$$F_{\beta} = \frac{\left((1+\beta^2)\cdot true\ positive\right)}{\left((1+\beta^2)\cdot true\ positive + \beta^2 \cdot false\ negative + false\ positive\right)}$$

**Financial performance metrics**

It will be informative to measure the performance characteristics of the model versus the benchmark also. In this instance I propose using the following metrics:

**Return over the test period:**

$$r=(r_{t1}+1)\times(r_{t2}+1)\times(r_{t3}+1)\times(r_{t4}+1)...\times(r_{tn}+1)-1$$

**Volatility over the test period, scaled by the time period:**

$$\sigma=\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2}x\sqrt{N}$$

**Sharpe Ratio:**

The sharpe ratio is a measure of how efficient an achieved return is, that is it measures the return per unit of risk and is defined as:

$$S_a=\frac{R_a-R_f}{\sigma_a}$$

Where $R_a$ is the return of the security or model, $R_f$ is the risk-free return and $\sigma_a$ is the standard deviation of the security or model's excess return over the risk-free rate.

**Maximum drawdown:**

The maximum drawdown is the measure of the largest decline from a historical peak in return to trough. It is defined as:

$$MDD(T)=\max_{\tau\in(0,T)}\left[\max_{t\in(0,\tau)}X(t)-X(\tau)\right]$$

# II. Analysis

Using pandas and Yahoo! Finance data, we can download historical stock price data for various stocks on US stock exchanges. Lists of stock symbols for the three main stock exchanges can be found here:

Nasdaq: http://www.nasdaq.com/screening/companies-by-name.aspx?letter=0&exchange=nasdaq&render=download
NYSE: http://www.nasdaq.com/screening/companies-by-name.aspx?letter=0&exchange=nyse&render=download
AMEX: http://www.nasdaq.com/screening/companies-by-name.aspx?letter=0&exchange=amex&render=download

In order for the model to generalise well, we should identify stocks from each sector with sufficient price history. In an attempt to mitigate both survivorship bias and time period of the data, we will select stocks from each sector that have varying lengths of history e.g. one stock with the maximum history, one stock with the median history and one stock with the minimum amount of history, in this instance 750 days (representing about 3 years of daily price data).

Using the symbols from the above links and sector identifiers, we arrive at the below subset of stocks.

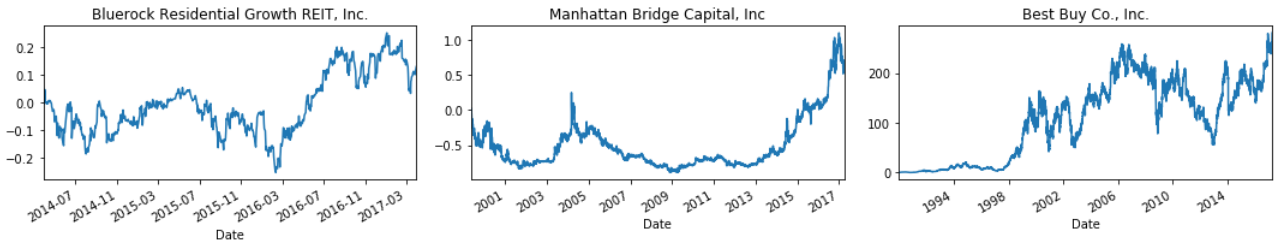
**Figure 1. Subset of US stocks to be used for training and testing**

| Symbol | Name | Sector | Industry | Number of days of data |
|---|---|---|---|---|
| AMRK | A-Mark Precious Metals, Inc. | Basic Industries | Other Specialty Stores | 758 |
| BAK | Braskem S.A. | Basic Industries | Major Chemicals | 4397 |
| ABX | Barrick Gold Corporation | Basic Industries | Precious Metals | 6867 |
| CBPX | Continental Building Products, Inc. | Capital Goods | Building Materials | 795 |
| STRT | Strattec Security Corporation | Capital Goods | Auto Parts:O.E.M. | 5408 |
| AA | Alcoa Corporation | Capital Goods | Metal Fabrications | 6867 |
| MDWD | MediWound Ltd. | Consumer Durables | Specialty Chemicals | 765 |
| VII | Vicon Industries, Inc. | Consumer Durables | Telecommunications Equipment | 5508 |
| AME | AMTEK, Inc. | Consumer Durables | Metal Fabrications | 6867 |
| KN | Knowles Corporation | Consumer Non-Durables | Consumer Electronics/Appliances | 788 |
| VCO | Vina Concha Y Toro | Consumer Non-Durables | Beverages (Production/Distribution) | 5335 |
| ADM | Archer-Daniels-Midland Company | Consumer Non-Durables | Packaged Foods | 6867 |
| BRG | Bluerock Residential Growth REIT, Inc. | Consumer Services | Real Estate Investment Trusts | 759 |
| LOAN | Manhattan Bridge Capital, Inc | Consumer Services | Real Estate Investment Trusts | 3871 |
| BBY | Best Buy Co., Inc. | Consumer Services | Consumer Electronics/Video Chains | 6867 |
| NADL | North Atlantic Drilling Ltd. | Energy | Oil & Gas Production | 800 |
| MMLP | Martin Midstream Partners L.P. | Energy | Oil Refining/Marketing | 3628 |
| APA | Apache Corporation | Energy | Oil & Gas Production | 6867 |
| CCCR | China Commercial Credit, Inc. | Finance | Major Banks | 756 |
| SAFT | Safety Insurance Group, Inc. | Finance | Property-Casualty Insurers | 3613 |
| AB | AllianceBernstein Holding L.P. | Finance | Investment Managers | 6867 |
| AGTC | Applied Genetic Technologies Corporation | Health Care | Biotechnology: Biological Products (No Diagnostic Substances) | 760 |
| MGLN | Magellan Health, Inc. | Health Care | Hospital/Nursing Management | 3299 |
| ABT | Abbott Laboratories | Health Care | Major Pharmaceuticals | 6867 |
| TNET | TriNet Group, Inc. | Miscellaneous | Business Services | 760 |
| NCTY | The9 Limited | Miscellaneous | Business Services | 3089 |
| BID | Sothebys | Miscellaneous | Business Services | 6867 |
| VZA | Verizon Communications Inc. | Public Utilities | Telecommunications Equipment | 781 |
| TSU | TIM Participacoes S.A. | Public Utilities | Telecommunications Equipment | 4623 |
| AEP | American Electric Power Company, Inc. | Public Utilities | Electric Utilities: Central | 6867 |
| RUBI | The Rubicon Project, Inc. | Technology | Computer Software: Programming, Data Processing | 756 |
| INS | Intelligent Systems Corporation | Technology | Computer Software: Prepackaged Software | 4408 |
| AAPL | Apple Inc. | Technology | Computer Manufacturing | 6867 |
| SALT | Scorpio Bulkers Inc. | Transportation | Marine Transportation | 831 |
| ATSG | Air Transport Services Group, Inc | Transportation | Air Freight/Delivery Services | 3448 |
| ALK | Alaska Air Group, Inc. | Transportation | Air Freight/Delivery Services | 6867 |

## Data Exploration and Visualisation

Using pandas we can source the historic price data from Yahoo! Finance and obtain the following charts plotting the performance of the Adjusted Close price history for a sample of the above stocks:

**Figure 2. Price performance plots**



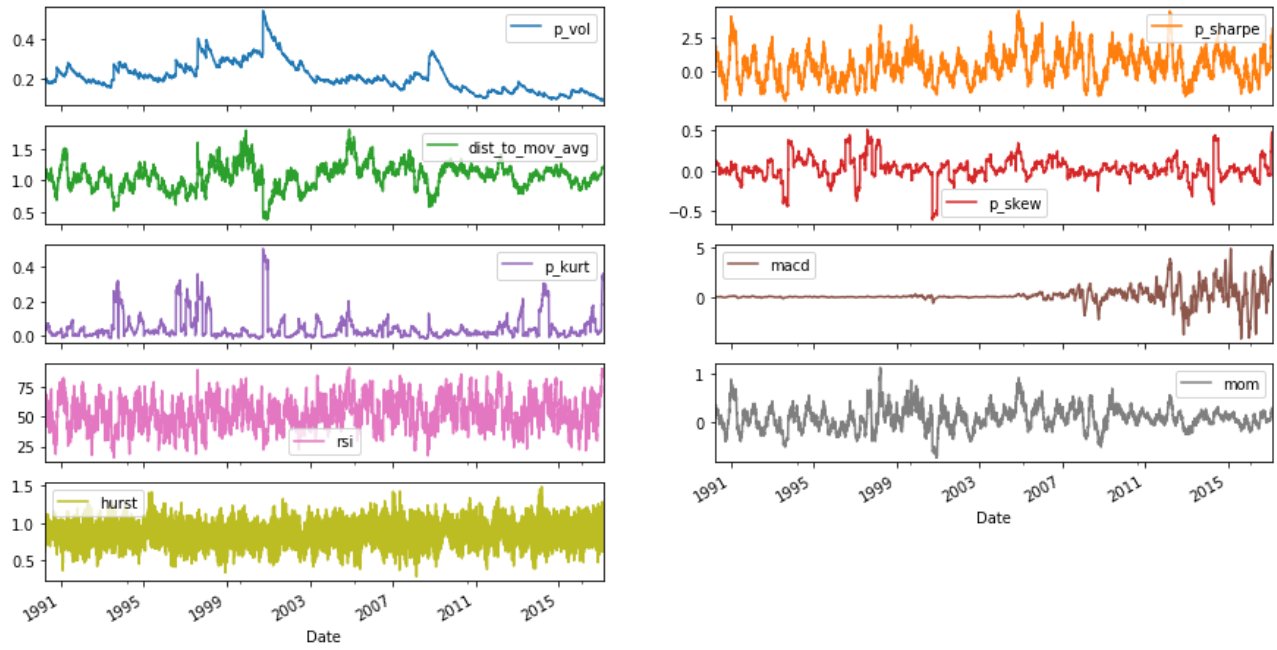

Using the historic price data, we will calculate a number of additional features for our learning algorithms:

1. N-day volatility
2. N-day sharpe ratio (or the volatility adjusted return over N days).
3. Ratio distance to N-day price moving average.
4. N-day skewness.
5. N-day kurtosis.
6. Moving Average Convergence Divergence technical indicator (“MACD”)
7. Relative Strength Index (“RSI”, a technical analysis indicator)
8. Momentum (“mom”, or simply the percent price change over N days)
9. Hurst exponent – a measure of how mean reverting a time series is.

Each feature will be calculated over a given timeperiod e.g. 5, 10, 20, 60 and 120 days.

An example of these features over 60 days for Apple, Inc. is shown below:

**Figure 3. Calculated features over 60 working days for Apple, Inc.**



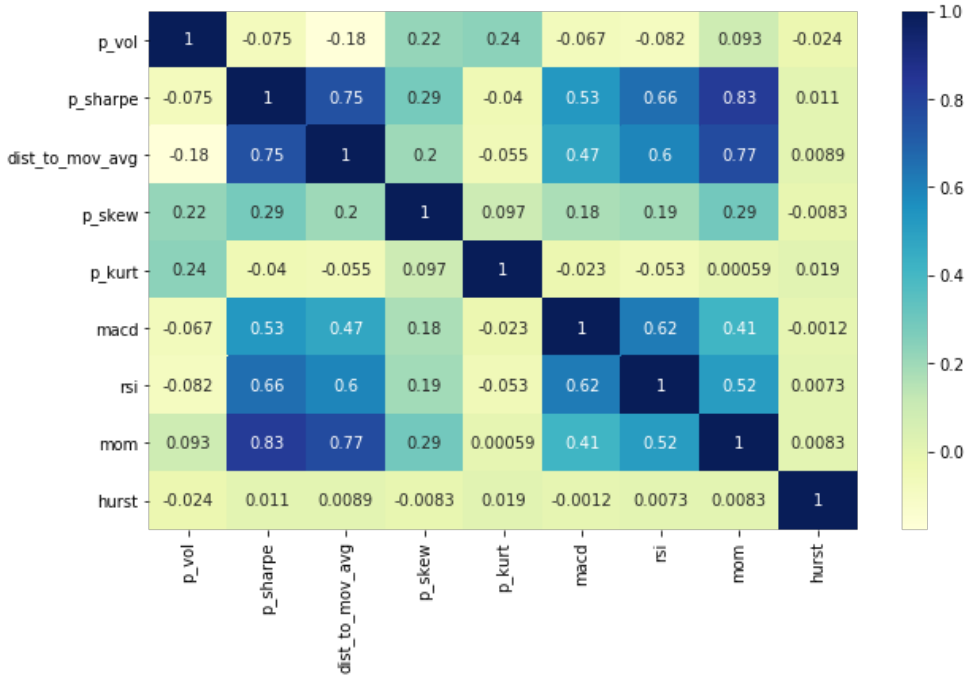

We can now create all these features across all the stocks we wish to learn from and explore the data. This results in a dataset of 139,460 datapoints, which should be sufficiently large a size to use for training and testing.

Firstly, how correlated are the individual features we have chosen?

The below correlation matrix shows high correlation in dark blue (e.g. 1.0 for perfect correlation) with lighter shades of yellow, green and lighter blue for lower correlations. We can see from both the values and the colours that there is a reasonably low correlation of all the features with volatility ("p_vol"). We can also see that sharpe ratios ("p_sharpe") have a reasonably high correlation with the technical indicators MACD, RSI and MOM. Features with high correlation with each other may turn out to be superfluous in the learning process as they may not indivudally capture truly orthogonal features of the underlying model.

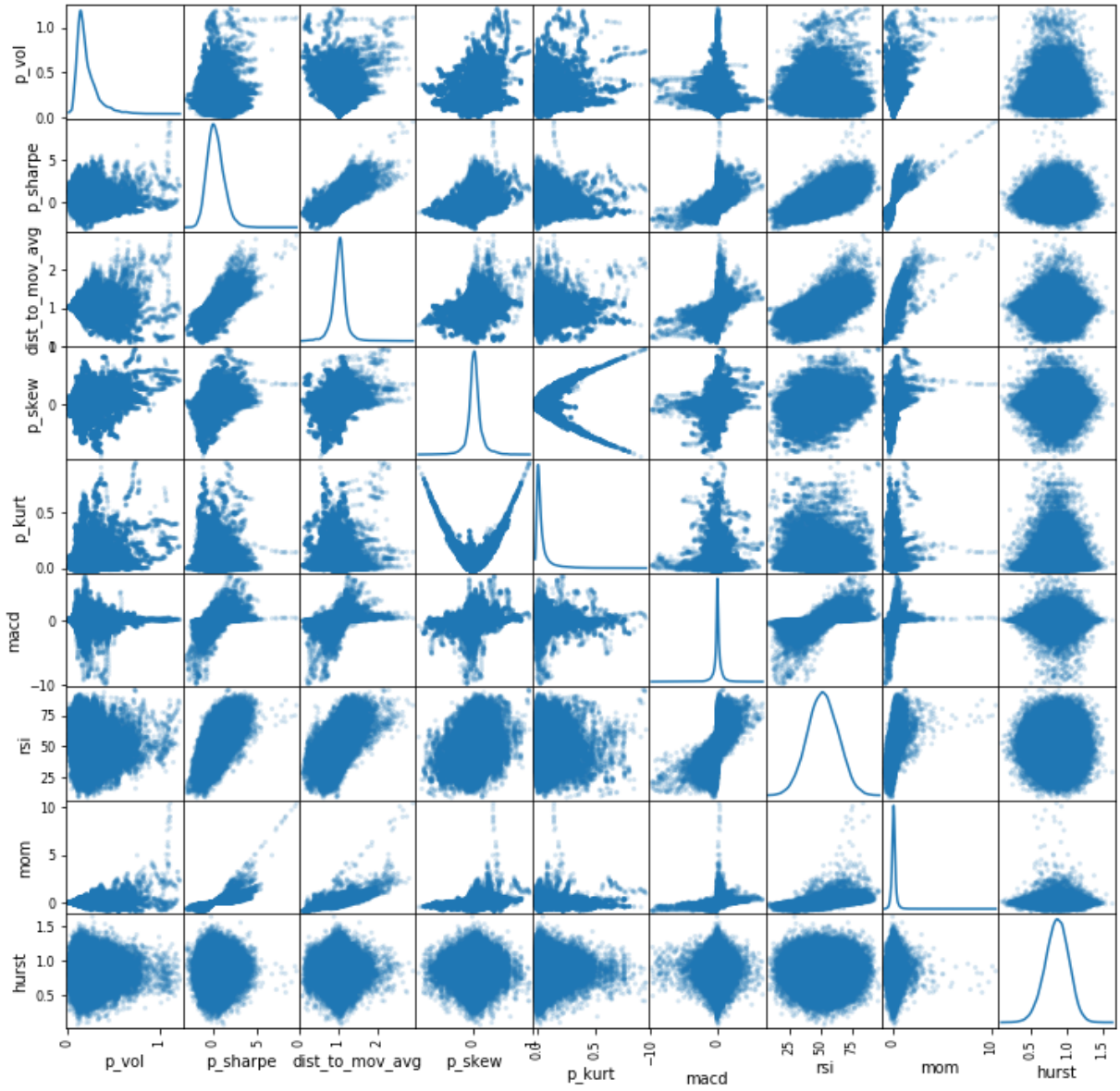
The Hurst exponent has zero correlation with any of the other features.

**Figure 4. Correlation matrix for features using a 60 day period.**



Using a scatter matrix and Kernel Density Estimate plot, we can visualise the correlation and distribution of each feature better.

**Figure 5. Scatter matrix and Kernel Density Estimate of features using 60 day period**



Looking at the diagonal first, we can see that skewness, RSI and the Hurst exponent appear to be roughly equally distributed around the central point. With the exception of the Moving Average Convergence Divergence ("MACD") indicator, the features appear to have positive skew.

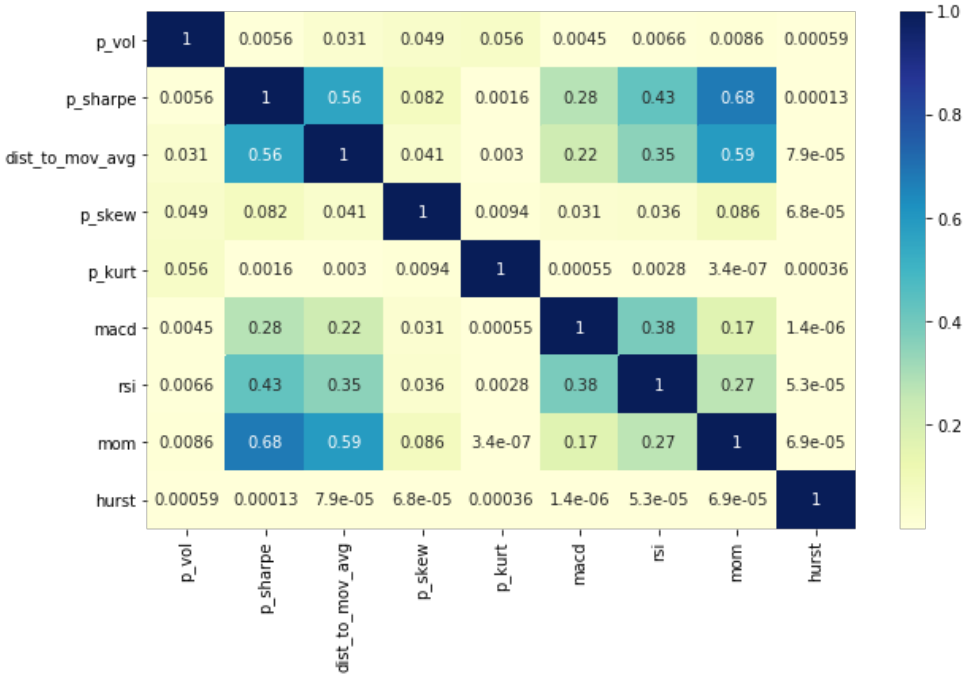
In the correlation table, the following features exhibit correlation close to or greater than 0.5:

Distance to moving average, Sharpe ratio, MACD, RSI, Momentum.

We can see from the scatter matrix that these features show some relationship: looking at Distance to moving average and sharpe ratio (dist_to_mov_avg, p_sharpe) we can see a positive slope. However the datapoints are spread around an imaginary line of best fit. We can make the same statement for the other variables. Looking at "mom" and "p_sharpe" a linear relationship appears to exist, however many datapoints would lie some distance from the line of best fit.

Using the square of the correlation coefficients produced earlier, we can see the r-squared value of the each feature with one another which should give us an indication of how well a line of best fit actually fits the data.

**Figure 6. Plot of r-squared values of features using 60 day holding period**



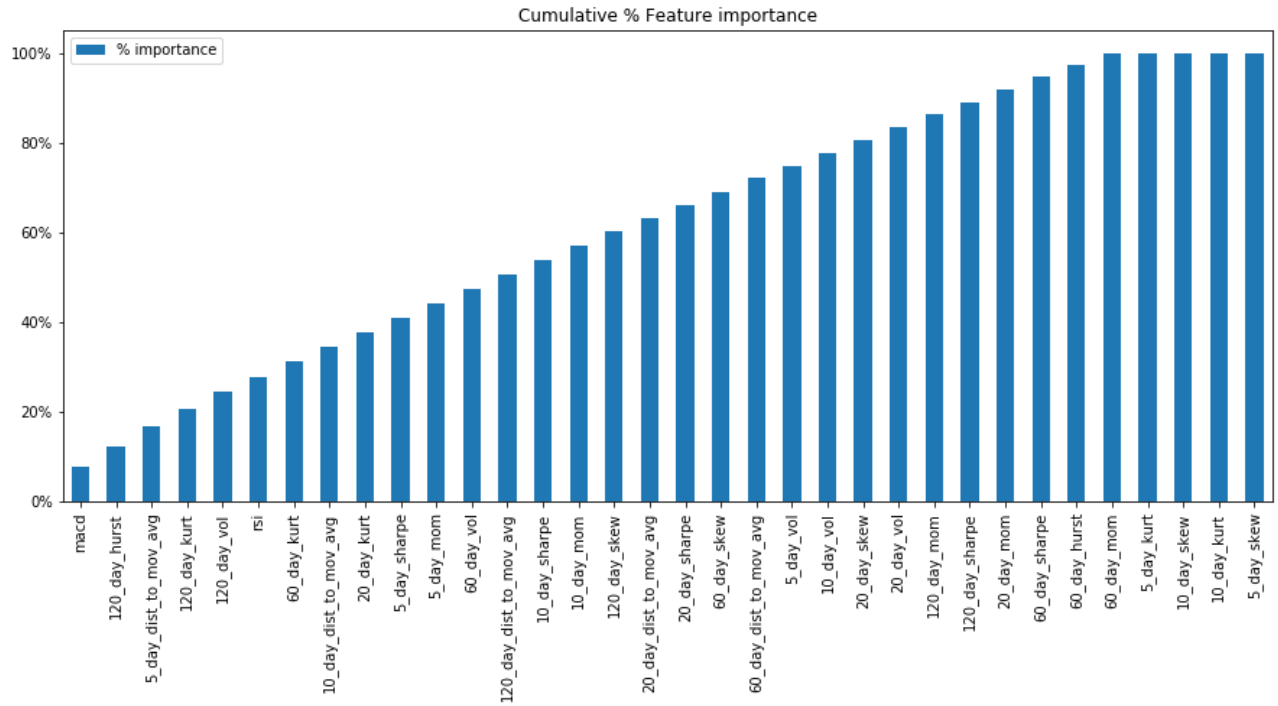

The r-squared value for "dist_to_mov_avg" and "p_sharpe" in this instance is 0.56, which suggests each feature explains about 56% of the variance of the other feature. For "mom" and "p_sharpe" this value is 0.68, which suggests that just over two-thirds of the variance of one is captured by the other.

This suggests we may be able to drop one or more features in the learning process.

**Feature Importance**

Figure 7 shows a cumulative plot of feature importance for all features and periods.

**Figure 7. Feature importance ordered by % importance (cumulative sum)**




MACD has the greatest feature importance at 7.78%. The average feature importance is 2.94% and the standard deviation of each importance is 1.41%. As the chart above demonstrates, the cumulative sum of importance increases in a reasonably linear way. In addition to the sttandard deviation we can infer that each features importance is quite tightly clustered around the mean.

The exceptions to this are 10-day Kurtosis, 5-day Skewness, 5-day Kurtosis and 10-day Skewness which have 0% feature importance each. These can be removed from the training set.

## Algorithms and techniques

Having labeled our data as described in the previous section, the solution to the problem is whether the learner can accurately predict the labels. We can then quantitatively score the learner versus this “perfect foresight” model. Using these predicted labels, we can then create a trading strategy for a particular security that buys or sells the security, given the model output 1 or 0. The performance of this strategy during a test and/or validation period can then be quantitatively compared to the actual performance of the security over the same time period.

By treating this as a supervised learning problem, in the first instance I would like to make use of support vector machines and decision trees to help classify the data.

There are many factors that may contribute to the future direction of a stock – it is not necessarily a question of simply separating these factors in a linear way. Support vector machines (“SVMs”) can help here as they can separate data in a non-linear way. Whilst SVMs can aid in a non-linear separation, the results can be hard to interpret. To aid this, decision trees can further help categorise our data in an easier to understand way.

A combination of these algorithms can be achieved using Scikit-learn’s VotingClassifier.

## Benchmark model

As we have labelled our training and testing data using “perfect foresight”, we can compare the accuracy of our predictions to these labels. The difficulty is knowing how well our learner has performed. Would an accuracy score of over 50% be sufficient? In this instance we can benchmark our score versus just holding the security with no buys or

sells after the initial purchase (e.g. each datapoint is labelled with a 1). In order for our learner to be making better predictions than the market, the learner accuracy score must therefore be greater than the benchmark accuracy score. Ideally, the model should also deliver a superior investment return, a higher sharpe ratio and a lower maximum drawdown than the benchamrk.

# III. Methodology

## Data Preprocessing

The data has been pre-processed according to the following steps:

1. Price data for holidays or other periods of stock exchange closure which occur on a weekday (Monday to Friday) are copied forward.

2. Data is visually checked for any obvious erroneous prices e.g. extreme price spikes. A stylised example is shown below.

**Figure 7. Stylised extreme price movement.**


3. Classification labels (0 or 1) are applied to the return of the stock 20 days forward: if positive, replace with 1, if negative replace with zero.

4. Once features have been calculated using price data, the features are normalised using Scikit-learn's StandardScaler to have mean zero and unit variance.

## Implementation

### Training

The following steps were followed to train our classifier:

1. For each stock in our set, calculate the features from the price data and label the data points with a 1 or 0 according to whether the stock went up (+1) in price or fell (0) 20 days forward. Each feature is calculated using a different time period. For the purposes of this exercise, the periods chosen are: 5 days, 10 days, 20

days, 60 days and 120 days. Each period roughly represents 1 week, 2 weeks, 1 month, 3 months and 6 months of business daily data.

2. Split this dataset into a training and a testing set. The test set represents the most recent 20% of all the data for each stock e.g. if the entire dataset for stock X is 36 month long, the training set is the first 28.8 months and the test set is the final 7.2 months of data. Reserve the test set for later use.
3. Once the individual training and testing datasets have been created combine the training datasets into one large dataset.
4. Normalise the training set and save the StandardScaler parameters for later use in testing.
5. Shuffle the training set.

This results in a training set with 132,003 data points and 34 features.

**Custom code requirements**

The following helper classes and/or functions were created to enable implementation:

**pytrader class:**

A custom class which performs the following main tasks:

1. calculation of features and labelling of training and testing data
2. a custom train_test_split function which splits the data into a train and test set, shuffles the training set but retains date ordering for the test set.
3. Wrapper functions for fitting and predicting
4. functions to create a "trading strategy" from the model predictions
5. utility functions to save and load the final model

**stats class:**

A custom class to calculate the relevant performance metrics from the trading strategy output of model predictions as well as functions to plot cumulative performance, maximum drawdown and other relevant metrics for visualisation.

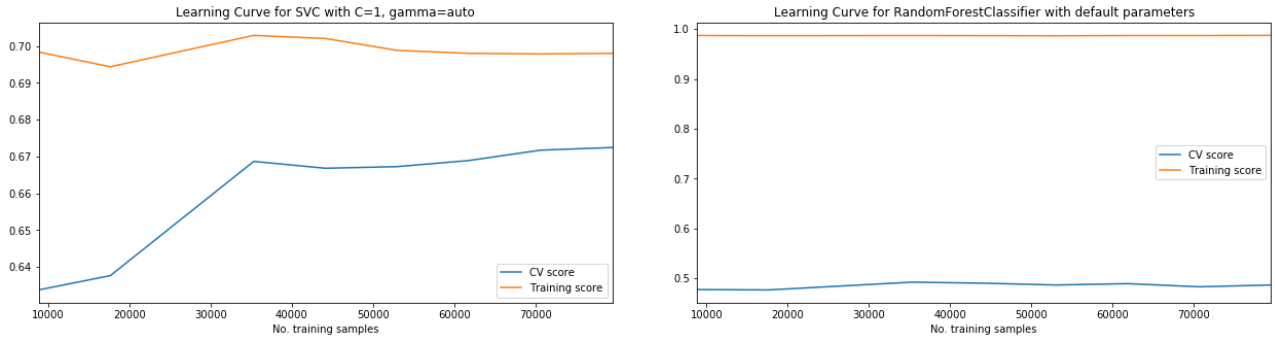
**Problems encountered**

To calculate the Hurst exponent I have made use of a 3rd party Python library ("nolds") to save on having to hand-code an implementation. For periods below 60 days, the Hurst exponent outputs nonsensical data or returns an error. In order to reach a conclusion of this project, I have decided to drop these 3 features (Hurst exponent over 5, 10 and 20 days) rather than spend time digging into 3rd party code and seek a solution. This approach may be justified as this feature gives a measure of how trending a time series may be. For short term periods, it would in all likelihood exhibit something close to a random walk (brownian motion, or "noise") and therefore not be useful in the learning process.

## Refinement

Learning curves for the individual, unrefined SVC and RandomForestClassifier models is shown in Figure 8.

**Figure 8. Learning curves for models with default parameters.**



We can make the following observations about each model:

- The training score for the SVC converges to around 70% with a low number of training samples and does not materially improve or worsen with the addition of more training samples. This suggests the model may be underfitting the data.
- The cross validation score for the SVC does improve with the addition of more training samples, however converges to a level of about 67% - quite close to the overall training score. This is also suggestive of a model which is underfitting and failing to capture the full underlying description of the data.
- The training score of the RandomForestClassifier converges to close to 100% immediately. It is generally accepted that Decision Trees can be prone to overfitting and this would appear to the case here.
- The cross validation score for the RandomForestClassifier converges to around 50% and fails to improve with the addition of more data. This supports the overfitting argument and demonstrates that the RandomForestClassifier is not generalising well to new data.

We may be able to improve the scores by tuning the parameters of each classifier. To help identify better parameters we will use Scikit-learn's GridSearch cross validation.

The following parameter space was set for the search:

**SVC using an RBF kernel:**

C parameter: 0.001 to 100, geometrically spaced.
Gamma parameter: 0.001 to 100, geometrically spaced.
Class weights: None, 'balanced', 0.5 for each label

The balance of classes in this training data is of particular importance. As stock prices have – in aggregate – tended to go up over time, we want to avoid a situation where the classifier simply predicts the most frequent class.

Having run the GridSearch using the above parameter space for each stock in our subset, I then averaged the values for C and gamma, and took the most frequent value for the class weights. The following parameters were identified as optimal:

C: 25.327714
gamma: 0.43075
Class weights: 0.5 for each class

**RandomForestClassifier:**
Number of estimators: [50,100,200,500]
Minimum samples per leaf: [2,10,50,100,200]
Maximum number of features: [None, 'auto', 'sqrt', 'log2'] (None means using all of the features available)
OOB score: [True, False]
Class weights: None, 'balanced', 0.5 for each label

Generally, the more estimators the better, although the classifier will generally reach a plateau at a certain number of these and not improve further. Increasing the minimum samples per leaf from the default of 2 may help with generalisation. The scikit-learn documentation cites empirically good values for using less than the number of maximum features for classification tasks.

Using the same approach with the SVM I used Gridsearch for each stock in the subset and then either averaged numerical values or took the most frequent in the case of categorical values:

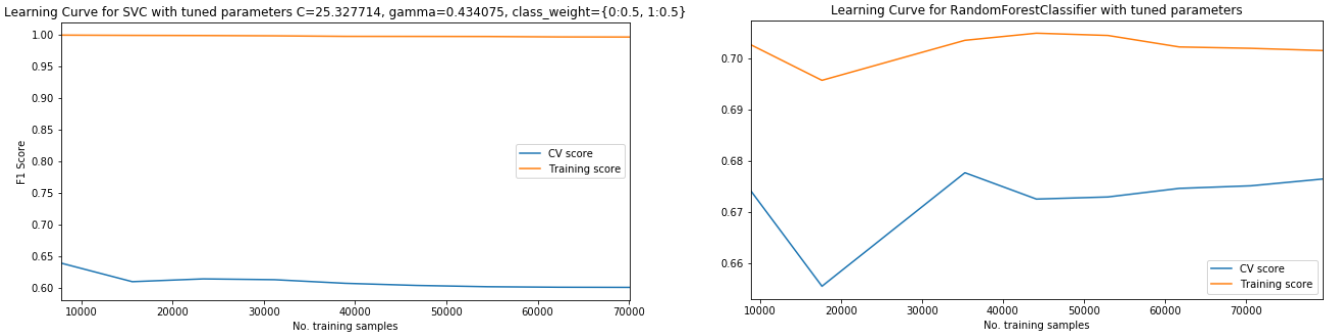
max_features: log2
min_samples_leaf: 200
n_estimators: 500
oob_score: True
class weights: 0.5 for each class

**Refined model output**

The parameter search identified the following as the best parameters for the SVC.
C: 25.327714
Gamma: 0.434075
Class weights: 1: 0.5, 0: 0.5

RandomForestClassifier:
max features: log2
min_samples_leaf: 200
n_estimators: 500

**Figure 9. Learning curves for models with tuned parameters.**




We can make the following observations:

- The RandomForestClassifier training score has reduced to around 70% from previously almost perfectly fitting the data. The cross validation scores have improved from around 50% to ~67% suggesting an improvement in generalisation to unseen data.
- The SVC training score has improved to close to 100% and the cross validation score remains around 67%.

Tuning the parameters of the models suggests they may generalise better than before, however are still suggestive of a model which is not fully capturing the true underlying relationship between the features and the labels.

# IV. Results

## Model Evaluation and Validation

Prior to combining the SVC and RandomForestClassifier into an overall ensemble, the train and testing output of the overall aggregate SVM and RandomForest  models is shown in Figure 10.

**Figure 10. SVC model scores less benchmark scores**

| | train_f1_diff | test_f1_diff | return_diff | sharpe_diff | drawdown_diff |
|---|---|---|---|---|---|
| **AMRK** | 0.3446 | -0.1220 | -0.0238 | -0.0348 | 0.1131 |
| **BAK** | 0.2844 | -0.1192 | 0.0241 | 0.0845 | 0.2654 |
| **ABX** | 0.3226 | -0.1236 | 0.0038 | -0.0039 | 0.1004 |
| **CBPX** | 0.2754 | -0.1386 | -0.1397 | -0.7757 | -0.0012 |
| **STRT** | 0.3036 | -0.1127 | -0.0442 | -0.1402 | 0.0442 |
| **AA** | 0.3008 | -0.1219 | -0.0030 | 0.0033 | 0.0439 |
| **MDWD** | 0.3111 | -0.0504 | -0.0184 | -0.1089 | 0.0099 |
| **VII** | 0.2658 | -0.0611 | 0.0139 | 0.0019 | -0.0319 |
| **AME** | 0.2733 | -0.1986 | -0.0238 | -0.1132 | -0.1584 |
| **KN** | 0.2375 | -0.0901 | 0.1289 | 1.7019 | 0.0512 |
| **VCO** | 0.2812 | -0.1591 | 0.0211 | 0.0876 | -0.0275 |
| **ADM** | 0.2774 | -0.2166 | -0.0313 | -0.1554 | -0.0758 |
| **BRG** | 0.2948 | -0.1300 | -0.0066 | -0.1240 | 0.0454 |
| **LOAN** | 0.2669 | -0.1368 | -0.0551 | -0.1076 | -0.0245 |
| **BBY** | 0.2965 | -0.0982 | 0.0012 | 0.0172 | 0.1136 |
| **NADL** | 0.3943 | -0.0141 | -0.0064 | -0.0100 | -0.0073 |
| **MMLP** | 0.3185 | -0.1335 | 0.0218 | 0.0488 | 0.2807 |
| **APA** | 0.3156 | -0.1514 | 0.0083 | 0.0171 | 0.0821 |
| **CCCR** | 0.3892 | -0.0072 | 0.0040 | 0.0059 | 0.0000 |
| **SAFT** | 0.3012 | -0.1192 | -0.0009 | 0.0531 | 0.0937 |
| **AB** | 0.2851 | -0.1573 | -0.0268 | -0.0858 | 0.0349 |
| **AGTC** | 0.3551 | -0.0196 | 0.0301 | 0.0518 | -0.0000 |
| **MGLN** | 0.2782 | -0.0770 | -0.0131 | -0.0421 | -0.0239 |
| **ABT** | 0.2881 | -0.1562 | -0.0043 | 0.0153 | -0.0331 |
| **TNET** | 0.2697 | -0.1792 | -0.0647 | -0.0192 | 0.1207 |
| **NCTY** | 0.3148 | -0.0426 | 0.0052 | 0.0047 | 0.0247 |
| **BID** | 0.2887 | -0.1493 | -0.0073 | -0.0070 | 0.1881 |
| **VZA** | 0.2895 | -0.0930 | -0.0045 | -0.0998 | 0.0107 |
| **TSU** | 0.3001 | -0.1665 | 0.0097 | 0.0269 | 0.1685 |
| **AEP** | 0.2634 | -0.1665 | -0.0031 | 0.0482 | 0.0212 |
| **RUBI** | 0.2786 | -0.0122 | 0.2079 | 0.3969 | 0.1144 |
| **INS** | 0.2304 | -0.0869 | -0.0158 | -0.0119 | -0.0633 |
| **AAPL** | 0.2935 | -0.1355 | -0.0173 | -0.0487 | 0.0964 |
| **SALT** | 0.2507 | -0.0058 | -0.4560 | -0.6801 | 0.0131 |
| **ATSG** | 0.3029 | -0.0969 | -0.0219 | -0.0548 | 0.0099 |
| **ALK** | 0.2827 | -0.1786 | -0.0212 | -0.0111 | 0.1038 |

Summarised statistics can be found in figure 11.

**Figure 11. SVC model scores less benchmark scores summary statistics**

| | train_f1_diff | test_f1_diff | return_diff | sharpe_diff | drawdown_diff |
|---|---|---|---|---|---|
| **count** | 36.0000 | 36.0000 | 36.0000 | 36.0000 | 36.0000 |
| **mean** | 0.2952 | -0.1119 | -0.0147 | -0.0019 | 0.0473 |
| **std** | 0.0349 | 0.0561 | 0.0920 | 0.3491 | 0.0897 |
| **min** | 0.2304 | -0.2166 | -0.4560 | -0.7757 | -0.1584 |
| **25%** | 0.2769 | -0.1526 | -0.0224 | -0.0893 | -0.0027 |
| **50%** | 0.2891 | -0.1219 | -0.0054 | -0.0085 | 0.0298 |
| **75%** | 0.3055 | -0.0844 | 0.0060 | 0.0197 | 0.1012 |
| **max** | 0.3943 | -0.0058 | 0.2079 | 1.7019 | 0.2807 |

Results where the model outperformed the benchmark are color coded green; underperformed in red; and in line with benchmark in black. We can infer the following information for the SVM model:

1. The model consistently outperforms its benchmark in training, suggesting it is overfitting the data.
2. The model performs underperforms the benchmark F1 metric.

3. The model delivers better investment returns than benchmark during the test period 13 times out of 36, or 36% of the time.
4. The model investment returns difference vary between -46% (a very poor outcome) and +21% (a very positive outcome). The majority of investment returns are small positives in excess of the benchmark.
5. The model drawdown difference delivers on average better drawdown outcomes (+4.7%) and in the best case a 28% difference. In the worst outcome, the model underperfoms benchmark by 15.8%.

**Figure 12. RandomForest model scores less benchmark scores**

| | train_f1_diff | test_f1_diff | return_diff | sharpe_diff | drawdown_diff |
|---|---|---|---|---|---|
| **AMRK** | 0.0691 | 0.0000 | 0.0357 | 0.1541 | 0.0000 |
| **BAK** | 0.0090 | 0.0000 | 0.0001 | 0.0002 | 0.0000 |
| **ABX** | 0.0472 | 0.0000 | 0.0008 | 0.0017 | 0.0000 |
| **CBPX** | 0.0000 | 0.0000 | 0.0169 | 0.0998 | -0.0000 |
| **STRT** | 0.0282 | 0.0000 | -0.0006 | -0.0017 | 0.0000 |
| **AA** | 0.0254 | 0.0000 | 0.0015 | 0.0044 | 0.0000 |
| **MDWD** | 0.0357 | 0.0000 | -0.0169 | -0.0454 | 0.0000 |
| **VII** | -0.0096 | 0.0000 | 0.0001 | 0.0001 | 0.0000 |
| **AME** | -0.0021 | 0.0000 | 0.0012 | 0.0061 | 0.0000 |
| **KN** | -0.0379 | 0.0000 | -0.0241 | -0.1135 | 0.0000 |
| **VCO** | 0.0058 | 0.0000 | 0.0007 | 0.0025 | 0.0000 |
| **ADM** | 0.0020 | 0.0000 | 0.0008 | 0.0036 | -0.0000 |
| **BRG** | 0.0194 | 0.0000 | 0.0192 | 0.1070 | 0.0000 |
| **LOAN** | -0.0085 | 0.0000 | 0.0003 | 0.0005 | 0.0000 |
| **BBY** | 0.0211 | 0.0000 | 0.0007 | 0.0016 | 0.0000 |
| **NADL** | 0.1189 | 0.0000 | 0.0108 | 0.0156 | 0.0000 |
| **MMLP** | 0.0431 | 0.0000 | -0.0003 | -0.0006 | 0.0000 |
| **APA** | 0.0402 | 0.0000 | 0.0017 | 0.0049 | 0.0000 |
| **CCCR** | 0.1138 | 0.0000 | 0.0080 | 0.0124 | -0.0000 |
| **SAFT** | 0.0258 | 0.0000 | 0.0005 | 0.0028 | 0.0000 |
| **AB** | 0.0097 | 0.0000 | 0.0008 | 0.0028 | -0.0000 |
| **AGTC** | 0.0797 | 0.0000 | -0.0243 | -0.0477 | -0.0000 |
| **MGLN** | 0.0028 | 0.0000 | 0.0000 | -0.0000 | 0.0000 |
| **ABT** | 0.0127 | 0.0000 | 0.0002 | 0.0012 | 0.0000 |
| **TNET** | -0.0057 | 0.0000 | 0.0032 | 0.0139 | 0.0000 |
| **NCTY** | 0.0394 | 0.0000 | 0.0019 | 0.0025 | -0.0000 |
| **BID** | 0.0133 | 0.0000 | 0.0014 | 0.0038 | -0.0000 |
| **VZA** | 0.0141 | 0.0000 | 0.0010 | 0.0117 | -0.0000 |
| **TSU** | 0.0247 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| **AEP** | -0.0120 | 0.0000 | 0.0004 | 0.0028 | 0.0000 |
| **RUBI** | 0.0032 | 0.0000 | -0.0048 | -0.0116 | 0.0000 |
| **INS** | -0.0450 | 0.0000 | -0.0007 | -0.0011 | 0.0000 |
| **AAPL** | 0.0181 | 0.0000 | 0.0009 | 0.0038 | 0.0000 |
| **SALT** | -0.0247 | 0.0000 | -0.0611 | -0.1059 | 0.0000 |
| **ATSG** | 0.0275 | 0.0000 | -0.0009 | -0.0027 | -0.0000 |
| **ALK** | 0.0073 | 0.0000 | 0.0010 | 0.0036 | -0.0000 |

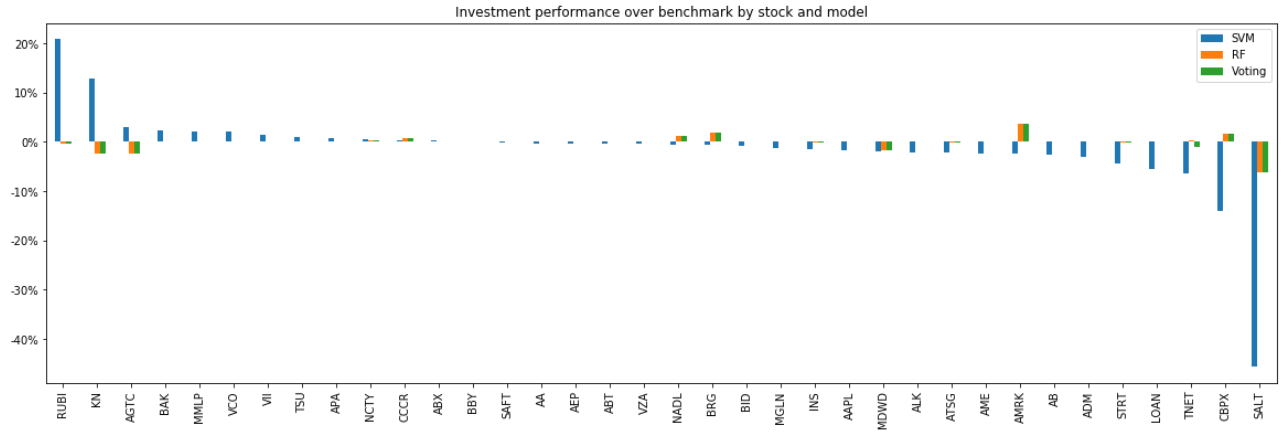
**Figure 13. RandomForest model scores less benchmark scores summary statistics**

| | train_f1_diff | test_f1_diff | return_diff | sharpe_diff | drawdown_diff |
|---|---|---|---|---|---|
| **count** | 36.0000 | 36.0000 | 36.0000 | 36.0000 | 36.0000 |
| **mean** | 0.0198 | 0.0000 | -0.0007 | 0.0037 | 0.0000 |
| **std** | 0.0349 | 0.0000 | 0.0145 | 0.0460 | 0.0000 |
| **min** | -0.0450 | 0.0000 | -0.0611 | -0.1135 | -0.0000 |
| **25%** | 0.0015 | 0.0000 | -0.0001 | -0.0002 | -0.0000 |
| **50%** | 0.0137 | 0.0000 | 0.0007 | 0.0025 | 0.0000 |
| **75%** | 0.0301 | 0.0000 | 0.0014 | 0.0045 | 0.0000 |
| **max** | 0.1189 | 0.0000 | 0.0357 | 0.1541 | 0.0000 |

The RandomForestClassifier has delivered an F1 score in line with its benchmark and modest performance gains over the benchmark investment return. This model has performed in line with benchmark when measuring drawdown.

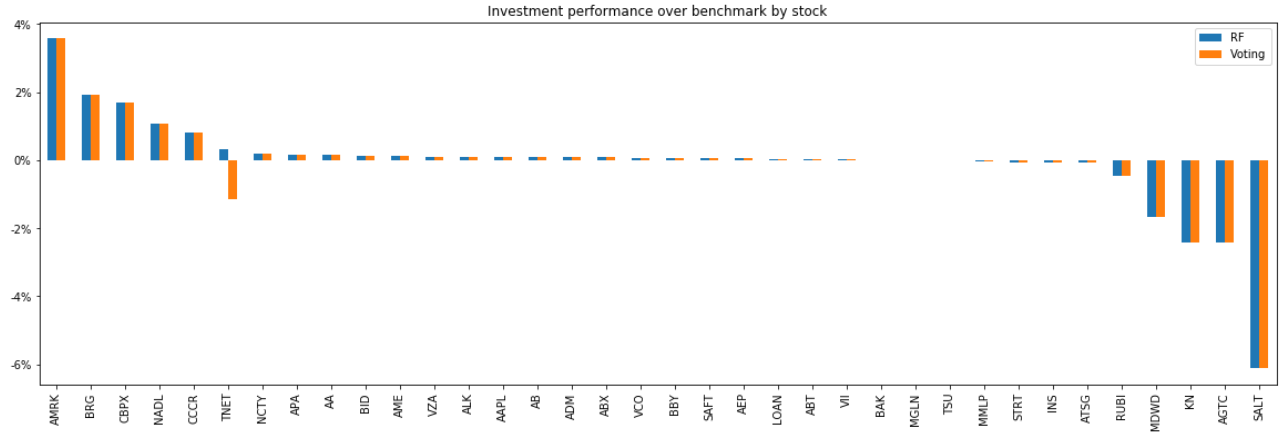
The investment return differences versus benchmark for the SVM, RandomForest and aggregated VotingClassifier are shown below:

**Figure 14. Model investment return less benchmark investment return**




The SVM model delivers the largest differences to the RandomForest and ensemble VotingClassifier. In most case the RandomForest and VotingClassifiers perform quite closely to benchmark as seen in Figure 15.

**Figure 15. Model investment return less benchmark investment return for RandomForest and VotingClassifier**

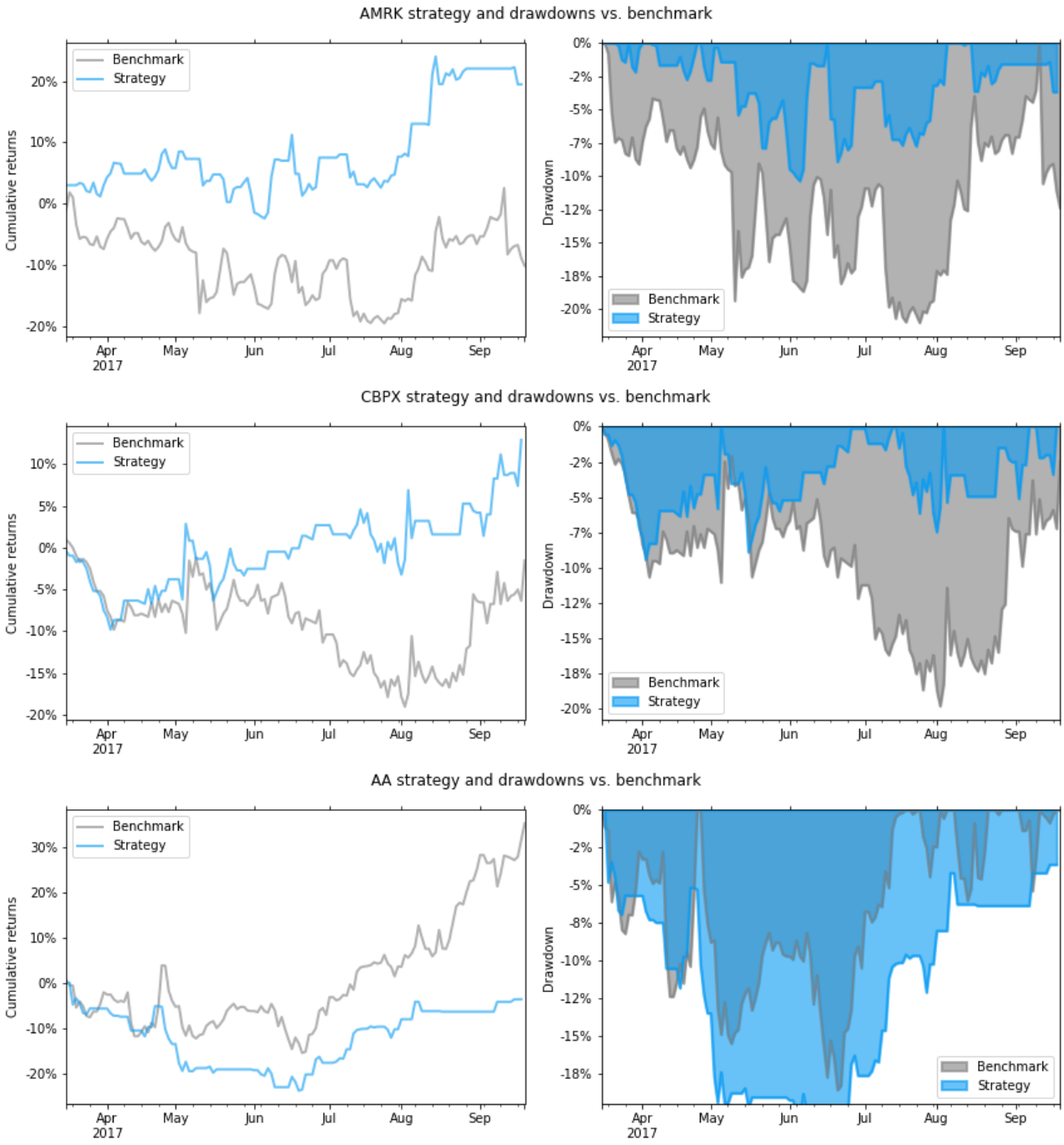



## Justification

The training and testing data set end date is 31 March 2017. The models were trained using 80% of the data available for each stock up to 31 March 2017. As such we can apply the model in the “real world” by running testing on the same stocks for data from 31 March 2017 up to today (16 October as of writing).

A sample of the model results is shown graphically below.

**Figure 16. Model investment return and drawdown compared to benchmark**




The differences to benchmark scores are summarised below:

**Figure 17. VotingClassifier model score less benchmark score using validation data**

| | test_f1_diff | return_diff | sharpe_diff | drawdown_diff |
|---|---|---|---|---|
| **AMRK** | -0.0072 | 0.2966 | 1.3744 | 0.1066 |
| **BAK** | -0.2168 | -0.0726 | 0.0754 | 0.0200 |
| **ABX** | -0.0812 | 0.0294 | 0.0796 | 0.0281 |
| **CBPX** | -0.1173 | 0.1447 | 0.6851 | 0.1036 |
| **STRT** | 0.0044 | 0.0771 | 0.2018 | -0.0046 |
| **AA** | -0.2747 | -0.3881 | -1.5654 | -0.0520 |
| **MDWD** | -0.0384 | 0.2440 | 0.8219 | 0.2246 |

| | test_f1_diff | return_diff | sharpe_diff | drawdown_diff |
|---|---|---|---|---|
| **VII** | -0.0030 | 1.2412 | 0.7372 | 0.1689 |
| **AME** | -0.2187 | -0.0835 | -0.4915 | 0.0198 |
| **KN** | -0.0271 | 0.1420 | 0.5944 | 0.0953 |
| **VCO** | -0.1687 | 0.0646 | 0.6663 | 0.0175 |
| **ADM** | -0.0660 | -0.0690 | -0.6000 | -0.0400 |
| **BRG** | -0.1546 | 0.1054 | 0.5397 | 0.1507 |
| **LOAN** | -0.0884 | -0.1508 | -0.6385 | -0.0115 |
| **BBY** | -0.1469 | -0.0546 | -0.1624 | 0.0000 |
| **NADL** | -0.0388 | 0.2870 | 0.1018 | 0.0000 |
| **MMLP** | -0.0677 | -0.0537 | -0.5026 | 0.0141 |
| **APA** | -0.0718 | 0.0829 | 0.2953 | 0.0943 |
| **CCCR** | -0.0289 | 2.9307 | 2.1330 | 0.1534 |
| **SAFT** | -0.2953 | -0.0669 | -0.7521 | -0.0279 |
| **AB** | -0.1803 | -0.0373 | -0.0196 | 0.0430 |
| **AGTC** | -0.1184 | -0.0237 | -0.2348 | 0.0140 |
| **MGLN** | -0.2831 | -0.0759 | -0.0894 | 0.0338 |
| **ABT** | -0.1840 | -0.1660 | -1.7520 | -0.0097 |
| **TNET** | -0.2042 | -0.0608 | -0.1777 | 0.0348 |
| **NCTY** | 0.0038 | 0.6011 | 1.5610 | 0.0957 |
| **BID** | -0.0700 | -0.0864 | -0.5486 | 0.0374 |
| **VZA** | -0.2313 | -0.0249 | -0.4221 | 0.0014 |
| **TSU** | -0.1959 | -0.0051 | 0.7173 | 0.0974 |
| **AEP** | -0.1945 | 0.0238 | 0.8633 | 0.0323 |
| **RUBI** | -0.0546 | 0.1233 | 0.3092 | 0.0747 |
| **INS** | -0.1353 | -0.0428 | -0.1659 | 0.0093 |
| **AAPL** | -0.0688 | -0.0364 | -0.2532 | 0.0000 |
| **SALT** | -0.0496 | -0.0061 | -0.0536 | -0.0022 |
| **ATSG** | -0.1573 | -0.4830 | -1.7952 | 0.0094 |
| **ALK** | -0.1209 | 0.0789 | 0.1296 | 0.0928 |

**Figure 18. VotingClassifier model score less benchmark score using validation data summary statistics**

| | test_f1_diff | return_diff | sharpe_diff | drawdown_diff |
|---|---|---|---|---|
| **count** | 36.0000 | 36.0000 | 36.0000 | 36.0000 |
| **mean** | -0.1209 | 0.1246 | 0.0462 | 0.0451 |
| **std** | 0.0857 | 0.5539 | 0.8343 | 0.0631 |
| **min** | -0.2953 | -0.4830 | -1.7952 | -0.0520 |
| **25%** | -0.1866 | -0.0674 | -0.4394 | 0.0000 |
| **50%** | -0.1178 | -0.0149 | 0.0279 | 0.0240 |
| **75%** | -0.0534 | 0.1099 | 0.6123 | 0.0945 |
| **max** | 0.0044 | 2.9307 | 2.1330 | 0.2246 |

This validation data shows that the model generally underperforms the F1 benchmark score, outperforming modestly in only 1 occasion (NCTY). The average return difference however is very positive (+12.5%). This mean value is clearly substantially positively skewed by the best performer outperforming benchmark by a 293% margin, with the worst performance of the model underperforming by 48% compared to a simple buy and hold strategy. This outperformance in general appears to be driven by the models ability to avoid losing days, as evidenced by the positive drawdown differences: the model on average loses less money on down days compared to a simple buy and hold strategy.

# V. Conclusion

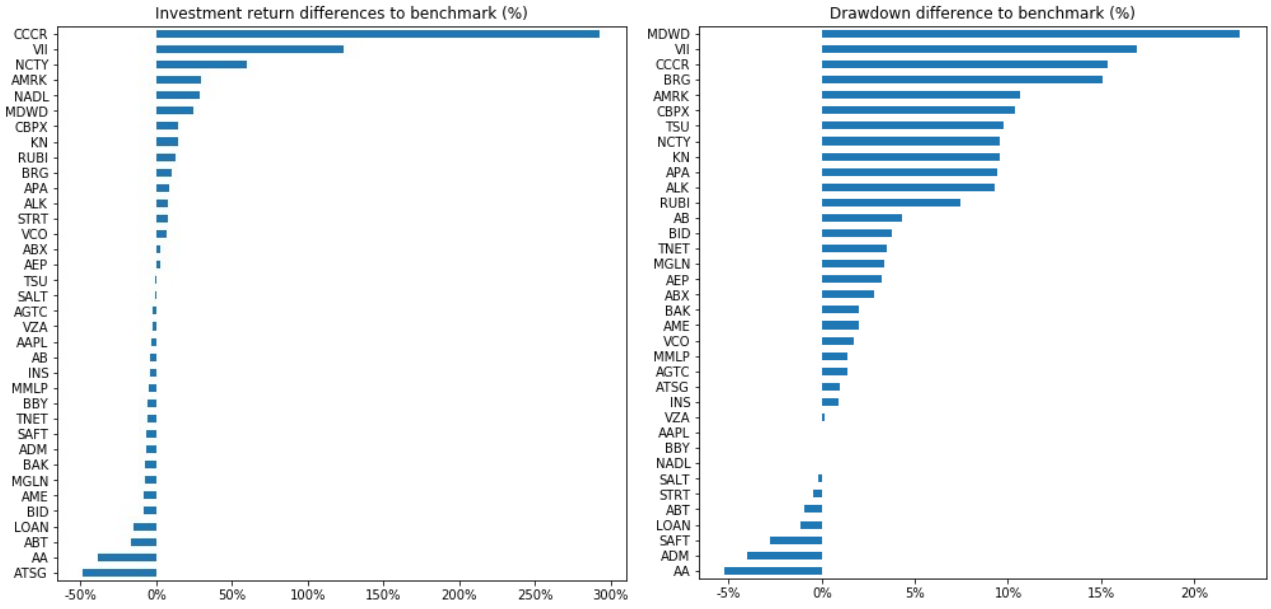
The observation that the model in validation has a better drawdown profile than the benchmark is an important one. In investing, the mathematics are quite simple. If I have $100 to invest and need $110 by the end of the year, I need to take risk commensurate with achieving a 10% return. If I lose 50% of the investment in the first period, I now need to generate a 100% return to get back to where I started, plus the 10% I needed originally. In short, I need to take a lot more risk to achieve my goals, in taking more risk, it is obvious that the risk of losing more money also increases.

By mitigating periods of drawdown, I should in theory be able to achieve desired returns with lower risk and therefore enjoy a smoother return profile (e.g. not a rollercoaster).

This is an important consideration in investing, as berhavioral finance suggests that investors are loss averse rather than risk averse, as first demonstrated by Tverksy and Kahneman[1]

The charts below show the return profile and drawdown for the validation data versus repsective benchmarks.

**Figure 19. VotingClassifier investment return and drawdown less benchmark, 31 March 2017 to 16 October 2017**




## Reflection

The project can be summarised according to the following steps:

1. The problem was identified (when to buy and sell stocks)
2. A benchmark and relevant performance metrics were specified
3. Freely available data was downloaded from Yahoo! Finance
4. A subset of stocks was identified to reduce the amount of training data into a representative sample
5. Data was preprocessed including reindexing the price data to every week day and then checked for any obvious outlier data points.
6. Features were then calculated for 5, 10, 20, 60 and 120 day periods, corresponding to 1 week, 2 weeks, 1 month, 3 months and 6 months respectively.
7. The data was segmented into 80% training data and reserved 20% for testing. The training data for each stock in our subset was concatenated into one large dataset and then shuffled. Time series ordering was retained for individual stock test sets.
8. The classfiers were trained on the data and grid search was employed to tune the model parameters.
9. The final tuned model was then applied to the test set data and results obtained.

In the original proposal document, I suggested using an SVC, RandomForestClassifier and Q-learning agent to combine into an ensemble. I found the reinforcement learning  piece too much additional work to implement in addition to the above structure. As such it is omitted from this final model. Furthermore the training and model tuning section was quite challenging. As there were approximately 117,000 datapoints with 50 features, the training time for the SVC varied between 10 and 20 hours. The end VotingClassifier took 10 hours to train. Frustratingly this had to be done several times due to either crashes or simple coding errors following the initial training.

The most interesting aspect of the project has been developing an end-to-end model training to prediction solution, which could now potentially be used in a production environment.

The model performance is – I believe – a satisfactory start given the simplicity of the features used. With additional fundamentals and economic data (generally paid-for datasets) the model performance might improve.

## Improvement

I think there are 3 key improvements that could be made to this project:

1. The addition of company fundamental data (earnings, sales, debt etc.) and macroeconomic variables (GDP growth, inflation, energy prices) might aid the model in generalising better to unseen data. Clearly these are all factors which drive economic activity but not explicitly captured in this model at present.

2. The use of principal component analysis on the existing features could help to reduce the dimensionality of the data. Furthermore, prinicpal component analysis might also aid in understanding when the model does well and when it does poorly (e.g. does it do better in high or low volatility market regimes?).

3. We used the open source Scikit-learn library implementation of SVMs and RandomForest Classifiers. I think Scikit-learn is great but for large datasets such as this, SVMs are quite slow to train. The lack of GPU implementation of the library is something we might be worth improving on in the future. I note that Tensorflow have recently added a SVM capability to that framework which might be worthy of investigation.