Domain Background

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 "-1" 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 features for each stock that I propose to use are:

1. N-day percentage return
2. N-day volatility
3. N-day sharpe ratio (or the volatility adjusted return over N days).
4. Percentage distance to N-day price moving average.
5. Percentage distance to N-day volume moving average
6. N-day skewness.
7. N-day kurtosis.
8. Hurst exponent.

The Hurst exponent is a measure of the "long-term" memory of a timeseries. A value of 0.5 suggests the timeseries is a "random walk" or that there is no autocorrelation of the series. A value below 0.5 suggests a mean-reverting timeseries, whereas a value greater than 0.5 suggests a trending timeseries.

Selection of N, or the period over which to calculate data will be important. For small values, the features will be too noisy. For large values, our models will not have enough data to train on. As such I propose using a value of 60 business days (or approximately 3 months of daily data).

Specific learning algortihms will be discussed in the solution and workflow sections.

Dataset and Inputs

One reason this proposed project lends itself well to machine learning techinques is the wealth of freely available data. I propose using Yahoo! Finance data for US stocks to retrieve historical stock market price and volume data. There are two possible ways we could use this data in machine learning: firstly, we could use regression techniques to attempt to predict the future price of a security and use this information to help the model decide whether to buy or sell. Secondly, we can simply label our historical data as to whether the price is higher or lower *N* periods ahead.

Using this historical price data, we can calculate many other statistics that may help our learner decide when to buy or sell a security.

The Yahoo! Finance dataset for a given company, in this case Alphabet, Inc. looks as follows:

Date	Open	High	Low	Close	Adj close*	Volume
28 Mar 2017	820.41	822.55	814.03	822.55	822.55	948,824
27 Mar 2017	806.95	821.63	803.37	819.51	819.51	1,894,300
...						
20 Aug 2004	50.32	54.34	50.06	108.01	53.95	22,942,800
19 Aug 2004	49.81	51.84	47.80	100.07	49.98	44,871,300

Alphabet is one example of a company with many years of trading history and therefore many datapoints from which a learner can learn. There are approximately 6000 tickers on the AMEX, NYSE and NASDAQ combined and around 3000 on the UK stock exchange. This wealth of data is important as we will be able to train and test our model using differing stock market regimes (for example the Global Financial Crisis in 2007-2009 and the subsequent period of recovery).

Using pandas, we can very simply and quickly label all of our data to train our learner: if the price is higher *N* periods ahead, label as 1. If it is lower, label -1.

Solution Statement

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 -1. 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.

Finally, I would like to also add a third model to the process and train a reinforcement learning agent. Each historical data point we use can clearly define a state. For example: volatility is high; return is negative; distance to moving average is very negative. Further, rewards (and discounted future rewards) can be clearly defined for each state, giving our agent something to learn from. Importantly, we should note that an action taken for a given state does not affect the state in the next period, but can affect what kind of reward our agent will receive. In the Bellman equation we should therefore include some kind of future discounted reward for a given action.

It will be instructive to analyse the output from each model to determine how much weight each puts on the various input factors and whether in fact there are differences in the predicted labels.

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. Further, the resulting return from the learner strategy should also be greater than simply buying and holding the security for the period in question.

For example:

Period	Security performance	Labelled data	Learner Label	Benchmark label	Learner performance (t-1 label * t performance)	Benchmark performance (t-1 label * t performance)	Learner accuracy	Benchmark accuracy
t1	0.02%	1	1	1			1	1
t2	0.42%	-1	1	1	0.42%	0.42%	0	0
t3	-0.35%	1	1	1	0.35%	-0.35%	1	1
t4	0.55%	1	1	1	0.55%	0.55%	1	1
t5	0.00%	1	1	1	0.00%	0.00%	1	1
t6	0.00%	1	1	1	0.00%	0.00%	1	1
t7	0.00%	1	-1	1	0.00%	0.00%	0	1
t8	0.00%	-1	-1	1	0.00%	0.00%	1	0
t9	-0.88%	-1	-1	1	0.88%	-0.88%	1	0
t10	-0.32%				0.32%	-0.32%		

Learner accuracy	77.8%
Benchmark accuracy	66.7%
Learner performance	2.54%
Benchmark performance	-0.60%

Evaluation Metrics

In order to evaluate the performance of both the benchmark and the model, I propose using the following evaluation metrics:

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:

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

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 σ_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:

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

Project Design

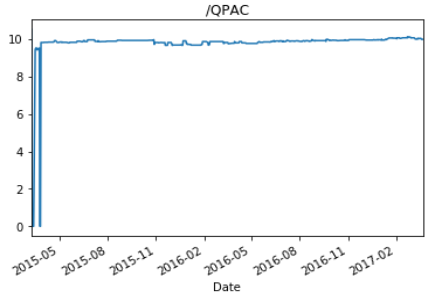
The proposed project workflow is:

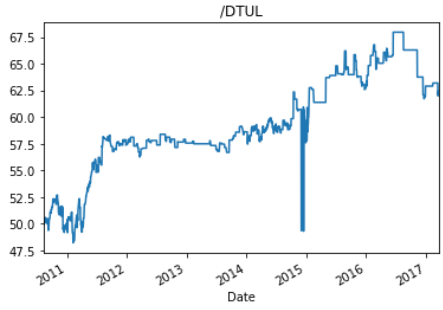
1. Data collection, analysis and screening.
2. Feature calculation and scaling.
3. Model training and tuning, testing and collection of evaluation metrics.
4. Model evaluation and validation using held back data (to simulate a live environment).

Data collection, analysis and screening

As noted earlier, I propose using the freely available stock market data for US stock markets (NASDAQ, NYSE and AMEX) from Yahoo! Finance. In order to improve efficiency of model training, it would be useful to download the information from Yahoo! to a local hard drive to speed up model training, rather than rely on a constant internet connection. We need sufficient data points for our models to learn from, as such we should filter out any stock which has less than 3 years of data. Further, we will need to determine that the historical price series does not contain too many stale prices, or obviously incorrect datapoints, for example:

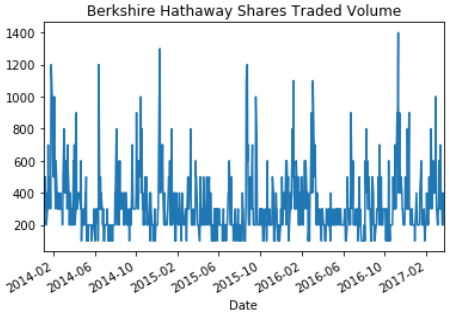
The left chart below shows an example of potentially stale prices, whilst the chart on the left shows a potentially bad datapoint where the line spikes down and back up quickly towards the end of 2015.





This problem can potentially be mitigated by creating an initial screen for stocks with market capitalisations greater than a threshold value (as larger companies will typically have more shareholders, their shares are likely to therefore be more liquid and therefore more free of data errors.

Not all large companies trade frequently. Berkshire Hathaway is a good example. The below chart shows the volume of shares traded since the beginning of 2014:



We will therefore also wish to set a volume (liquidity) threshold. This threshold should be determined through initial analysis of the dataset.

Feature calculation and scaling

The proposed features were defined in the problem statement. Many machine learning algorithms require features to have zero mean and unit variance, so scaling our data will be important prior to training. Where it is necessary to normalise values, we should calculate z-scores for the relevant input (in this instance moving-averages). For standardized moments (volatiliity, skewness, kurtosis) no normalisation should be performed. For returns based data, no normalisation should be performed. It is well evidenced that financial markets can exhibit “fat tails” and do not wholly conform to a normal distribution. I would argue this is a feature of our data on which we wish to train, and therefore not something that we should normalise away.

Model training and tuning, testing and collection of evaluation metrics

My approach to this problem will be to label datapoints as being a buy or sell opportunity. As such this is a supervised learning or classification problem. I would like to explore the use of ensemble methods with soft voting to predict labels.

I propose using 3 potential classification algorithms and discuss their potential below:

Decision Trees / Random Forests

Decision tree output can be reasonably easy to interpret, but can overfit to the data. Random forests can help correct for this overfitting tendency. For example, a decision tree may split on whether recent returns are negative. The next split may be made on how far the price is from its moving average. Subsequently the decision tree may then split on whether the stock is mean reverting or trending. By running this multiple times and collecting the mode of the output classes for each iteration, we can potentially avoid (or reduce at least) potential overfitting.

Support Vector Machines (“SVMs”)

Given the number of features I propose using for this classification problem, it is highly possible that the data will not be simply linearly separable. SVMs can efficiently perform non-linear classification using the “kernel trick” which maps input features into a multi-dimensional feature space. One drawback of SVMs is that the decision output (separation by a multi-dimensional hyperplane) can be hard for humans to interpret in a classic 2-dimensional sense. However, SVMs have shown high accuracy in image classification, protein classification and text classification. As such I believe this algorithm – and most importantly its ability to work in higher dimensonal space – lends itself well to this problem of binary classification with multiple input features.

Reinforcement Learning (Q-learning)

Q-learning can be used to find the optimal action-selection policy for a given state. More simply, given a particular state (e.g. high volatility, negative return, high skewness) the learning agent will select a particular action (buy or sell in this instance) and will either be rewarded or penalised for its decision. As the agent sees more and more states, it can learn to select a better action. This more closely approximates how people learn from experience and has direct application to financial markets. For example, at the beginning of March 2009, financial markets reached a bottom following the financial crisis. The subsequent returns in the stock market over the next year and beyond were very high. The agent could potentially learn this policy from historic data and apply that experience given similar future states.

In training, the agent will of course see every historic state and gradually build a Q-table of state/action pairs based on the reward received for that action (initially chosen at random). In testing, the agent may encounter states that do not exist within its Q-table. It may therefore be useful to add a distance calculation (i.e. how similar is this state to previously seen states) to facilitate optimal action selection.

Model tuning

For decision trees and support vector machines, model tuning through a grid search and cross validation will be a critical part of the process.

Whilst reinforcement learning for a classification problem may be considered overkill, it is possible to define relevant actions for a given state and as such we can train an agent. Definition of the relevant state space in reinforcement learning will be critical in order to achieve convergence within an acceptable period of time. Continuous values defined in the previous section should be discretised to reduce total state space.

For each model, we should record the accuracy (F1 score) for training and testing as well as the relevant financial performance metrics.

Soft voting is necessary in this scenario to enable a form of “conviction scoring” on the predicted position. If all 3 models predict a “1” (or buy position) then we have high conviction and the position size should be fully scaled (i.e. 100% of the position). If only 1 model predicts a “1” then the learner (in aggregate) is less sure, therefore has lower conviction and we should scale our position size accordingly (i.e. by one-third).

Evaluation of training, testing and validation scores

Finally, it will be instructive to validate the best performing models (e.g. model F1 score > benchmark F1 score) using the held-back data to simulate a “live” environment. We can then apply the same metrics (F1 score and financial performance metrics) to see how our model may have performed in real life.