STOCK PREDICTOR

Machine learning Nanodegree Capstone

FEBRUARY 1, 2017

UDACITY

Table of Contents

{ TOC \O "1-3" \H \Z \U }

Definition

Project Overview

The top level aim of this project is to utilize various machine learning techniques to determine if any o f them can be used to assist with making stock purchase decisions.

Problem Domain

The stock market is self-adjusting: any opportunity to gain from a trade quickly dissipates as the market adjusts to whatever new condition is imposed on it. Any given stock on the market looks to have a number of factors that contribute to its price:

- Fundamental factors such as the strength of the economy under the stock in question
- Financial factors e.g. the profitability of the company for the stock
- Technical factors being the variances or 'trends' in trading from other traders and general vol atility of the market e.g. a rising stock will continue to rise as other traders 'jump on the bandw agon'
- Random factors that cannot be attributed to any of the above Stocks will have a combination of the above factors influencing their price, of which makes for a very interesting problem to solve with machine learning as there are a number of different ways, each flawed in their own way in which to approach this problem.

Problem Statement

The key problems in trying to predict any stock price will be:

- Domain knowledge to assist with prediction of fundamental factors, e.g. knowing the strength of the underlying economy will assist with determining fundamental factors that influence
- Random factors that are unaccounted for in any of the above (e.g. false but accepted news that may negatively affect price)
- Past patterns do not mean future patterns: due to changes in the underlying financial environment, a previously discovered pattern may not replicate in full due to
- Transmission of news: the market takes time to react to any new and sudden news e.g. CEO resigning, profit statement better than predicted, etc.

The key items that will assist in prediction of stock price are:

- 'Blue chip' stocks don't tend to have a lot of volatility
- It is known that some stocks tend to ebb in flow in price in accordance with monthly cycles and company press releases
- Previous days price is often a good indicator of the next day's price
- Traders follow patterns and trends, themselves influencing the markets they trade within. If these patterns can be detected then we effectively 'stand on their shoulders'

The algorithms I have chosen to solve this problem with are as following. I have explained these more in Algorithms section.

- Linear Regression
- SVM
- Neural Networl
- Recurrant Neural Network

Strategy for Solving

This project will take a subset of the above and refine an approach that may help in prediction. The project scope is within following criteria:

- Select from a list of potential candidate stocks, a single stock for further processing by machine l earning. The stock will have statistical analysis performed on it to find which from the list is the potential best candidate for ML given its variance, growth, return and volatility. Analysis will be in the form of computing the percentage daily variance and evaluating based on this, as it's th daily variance we are most interested in.
- Candidate stock will be chosen by a comparison of the number of outliers, and mean percentage increase. The logic behind this is that stocks which have large spikes up or down are likely affected by external factors that we are unable to predict, hence we are not interested in stocks that have a large number of outliers (>1.5 IQR).

- Mean percentage increase will give stocks that have increased overall across the time period, as without significant predictable movement then there is no money to be made in trade.
- Rank and select the single candidate stock and then fetch historical data about the stock from Yahoo! Finance API. The logic here is that previous year's volatility is a good indicator of current period volatility (i.e. the financial stability of the company, whether the stock is 'high risk' or not), and the previous day's price is a good indicator for tomorrow's price.

Date	Open	High	Low	Close	Adj Close*	Volume
Dec 30, 2016	782.75	782.78	770.41	771.82	771.82	1,770,000
Dec 29, 2016	783.33	785.93	778.92	782.79	782.79	744,300
Dec 28, 2016	793.70	794.23	783.20	785.05	785.05	1,153,800
Dec 27, 2016	790.68	797.86	787.66	791.55	791.55	789,100
Dec 23, 2016	790.90	792.74	787.28	789.91	789.91	623,400
Dec 22, 2016	792.36	793.32	788.58	791.26	791.26	972,200
Dec 21, 2016	795.84	796.68	787.10	794.56	794.56	1,211,300
Dec 20, 2016	796.76	798.65	793.27	796.42	796.42	951,000
Dec 19, 2016	790.22	797.66	786.27	794.20	794.20	1,232,100
Dec 16, 2016	800.40	800.86	790.29	790.80	790.80	2,443,800
Dec 15, 2016	797.34	803.00	792.92	797.85	797.85	1,626,500
Dec 14, 2016	797.40	804.00	794.01	797.07	797.07	1,704,200

FIGURE { SEQ FIGURE * ARABIC }: SAMPLE DATA FROM YAHOO! FINANCE SHOWING 'GOOG' STOCK PRICE

- Feature selection: User will be able to choose from any of the data columns e.g. open price or close price, etc. I have deliberately excluded using ML techniques to select this to limit scope. From a column of selected data e.g. close price, the features will be a sequence of close prices, and the predicted value the value at the day ahead of where we wish to predict.

Metrics

The metrics that will be used to determine model accuracy are as following:

- R2 score for coefficient of determination for the test and train datasets
- Percentage difference of predicted value from actual value

- Variance of return: difference in stock price vs predicted difference in stock price

The R2 score will let us know the quality of the fit of the model against the underlying data. R2 scores range from zero (the model seems to disregard the input data) to one (the model is a perfect representation of the data) and can go to negative (as in, the model is doing worse than ignoring the data). R2 is calculated as 1- (Sum of Squares of residuals/Total Sum of Squares).

The percentage difference will allow gauging of the effectiveness of the prediction in real- world terms, and is simply the percentage difference of the predicted price vs the actual price, (actual – predicted)/actual. This gives a simple to understand metric that allows broad generalization as to the accuracy of the algorithm. Results that have a low percentage difference will mean that the algorithm used was able to predict the price very accurately.

The third metric, variance of return, will assist in determining what would have happened had we trusted the prediction and made the trade. Variance of return will let us know the quality of the prediction (as opposed to just plain accuracy) – it will inherently let us know if the algorithm is able to predict complex patterns, rises or falls in the stock price. The variance of return is calculated as (predicted price – actual price)/(actual price – price at end of data series).

E.g. if the stock price at the time of trade (i.e. at the end of our data series) was $15, and the program predicted $19, but the actual was $20, then this would mean a variance in return of +$1 or 1/5 = +20%, i.e. had we made the trade, there would have been 20% difference in what we would have made, given the total movement in the stock during the period.

Analysis

Data Exploration

The dataset size can vary by definition – the user can choose how many date periods and how many stocks they wish to evaluate. For the exploration, dataset has 1762 samples with 7 different features.

Most stock line charts look quite erratic – lots of ups and downs, and depending on the stock sometimes quite suddenly. As this is a stock prediction algorithm, and in turn what we are interested in is making money from variances in price, a good place to start with was the percentage change. I used pandas to fetch in candidate stocks and then calculated percentage change across the dataset, then used 'describe' function to analyze.

FIGURE { SEQ FIGURE * ARABIC }: GOOGLE STOCK ADJ_CLOSE PRICE TREND IN SEVEN YEARS

Let's briefly describe data features. Open is the price of the stock at the beginning of the trading day (it need not be the closing price of the previous trading day), high is the highest price of the stock on that trading day, low the lowest price of the stock on that trading day, and close the price of the stock at closing time. Volume indicates how many stocks were traded. Adjusted close is the closing price of the stock that adjusts the price of the stock for corporate actions. While stock prices are set mostly by traders, stock splits (when the company makes each extant stock worth two and halves the price) and dividends (payout of company profits per share) also affect the price of a stock and should be accounted for.

Below are some visualizations of these features:

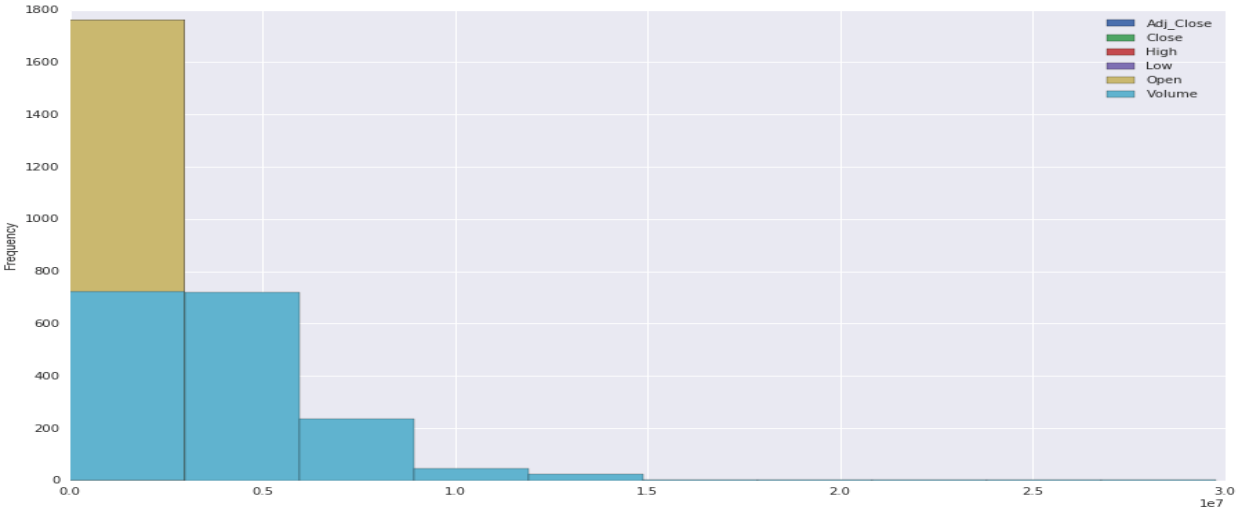


FIGURE { SEQ FIGURE * ARABIC }: FEATURE HISTOGRAM

Above displays a histogram of features distribution. We can see here that feature histogram is skewed right or right-tailed. For this type of distribution, we can say the mode should be somewhere around the left and we can argue that the median should be less than the mean.

Below displays the density of each feature. Density plots are another way of getting a quick idea of the distribution of each attribute. The plots look like an abstracted histogram with a smooth curve drawn through the top of each bin, much like your eye tried to do with the histograms.

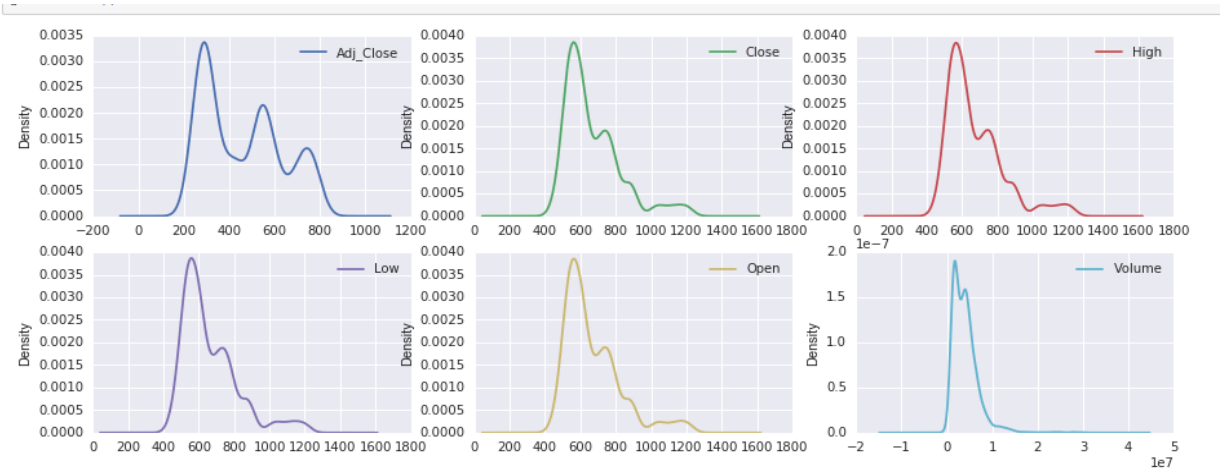


FIGURE { SEQ FIGURE * ARABIC }: FEATURES DISTRIBUTION

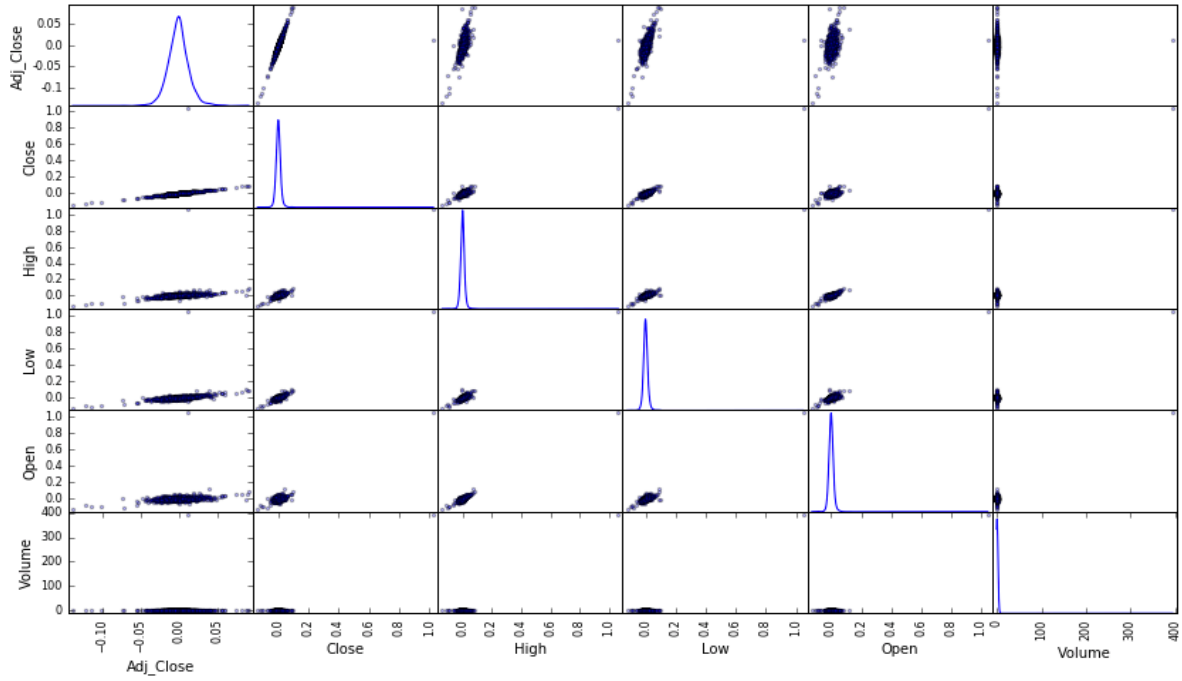


FIGURE { SEQ FIGURE * ARABIC }: FEATURES DISTRIBUTION MATRIX

As we can see from the distribution matrix features are mostly linearly correlated. You can also see some outliers. Regarding to the feature distribution Adjusted Price is well distributed meanwhile Volume has the least distribution and it is somehow linear.

To analyze the data better we need to know the adjusted price percentage change from one day to another. From the below, we can see that mean percentage change for all these stocks is quite small (understandably as this is a daily change). Other than that, there isn't too much to work with: the SIP stock has a significantly higher standard deviation and variance suggesting it varies wildly with its price (which it does), and the other stocks seem somewhat comparable in terms of quartiles and variance, and outliers.

	Adj_Close	Close	High	Low	Open	Volume
0	0.014213	0.014213	0.004024	0.011046	0.000741	-0.578343
1	0.014213	0.014213	0.004024	0.011046	0.000741	-0.578343
2	0.002887	0.002887	0.010561	0.005495	0.013238	0.526139
3	0.008280	0.008280	0.004570	0.005691	-0.003805	-0.303346
4	-0.002072	-0.002072	-0.006417	-0.000479	0.000278	-0.209986

FIGURE { SEQ FIGURE * ARABIC }: DATA PERCENTAGE CHANGE

	count	mean	std	min	25%	50%	75%	max	variance	outliers	mean_x_outliers
Adj_Close	1762.0	-0.000383	0.015522	-0.138321	-0.008367	-0.000231	0.007002	0.091435	0.000241	388	-9.863534e-07
Close	1762.0	0.000193	0.028980	-0.138321	-0.008367	-0.000231	0.007002	1.026943	0.000840	389	4.962778e-07
High	1762.0	0.000166	0.028803	-0.139055	-0.006702	-0.000186	0.005672	1.062617	0.000830	404	4.113046e-07
Low	1762.0	0.000185	0.029012	-0.124031	-0.008160	-0.000929	0.006225	1.046402	0.000842	415	4.445897e-07
Open	1762.0	0.000182	0.029328	-0.129245	-0.009158	-0.000447	0.007277	1.045787	0.000860	379	4.799700e-07
Volume	1762.0	0.287888	9.401630	-0.971152	-0.188675	0.018977	0.256817	394.358779	88.390650	184	1.564608e-03

FIGURE { SEQ FIGURE * ARABIC }: PERCENTAGE CHANGE AND OUTLIERS

As the algorithms, can only process one stock at a time, I have created a process to select what could be the 'best' stock for prediction. The program fetches in the list of candidate stocks, and then forward fills and backfills any nulls. The decision to backfill and forward fill here instead of dropping nulls was because we need as many rows as possible to determine which stock is valid, and we don't want to risk dropping any critical rows of data for the sake that a different stock didn't trade on that day for example.

The date field is dropped as we are only selecting a stock at this point, and the index is sufficient to know progress over time. Using the pandas function *pct_change()* gives a dataset that has just the percentages change up or down across the period.

I think calculate the outliers: outliers are calculated by using numpy functions, and I define an outlier as being 1.5 IQR above or below the IQR:

```
outliers = pd.DataFrame(np.where((data > 1.5 *((data.quantile(0.75) -
data.quantile(0.25)) + data.quantile(0.75))) | (data < 1.5 *
(data.quantile(0.25) - (data.quantile(0.75)) - data.quantile(0.25))), 1, 0),
```

Then, a 'describe' and 'transpose' is performed on the dataset, and the 'outliers' column added. I think multiply the mean by 1/number outliers to give a ranking metric. The results are then ranked by this metric, and the top scoring item is selected for processing by the algorithms.

Feature Selection

The main feature selection used is the price column (as we are training on the sequences within the price data itself), and this is used is left up to the operator: they can select from open price, close price, or any other feature they wish to predict. Once this is selected, the data is transposed and broken into N-S-D datasets, where N is the number of rows fetched in total, S is the parameter 'Sequence length' and D the number of days forward that is desired to be predicted. The result of this processing is that we have data rows comprising of sequences of prices, and then labels that correspond with the actual price D days ahead from the last price in the sequence.

The idea behind this is with the field technical analysis of stocks: patterns in price movement can often lead to determination of future price. E.g. a declining sequence of prices often leads to a lower price, and an oscillating price often settles on a value once the market adjusts.

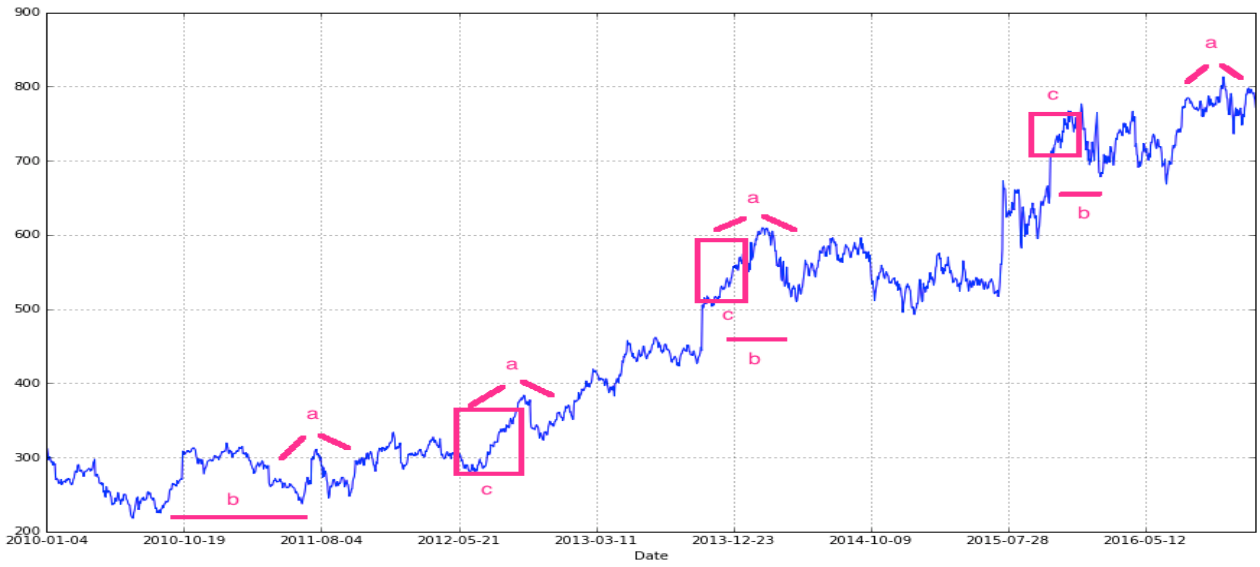


FIGURE { SEQ FIGURE * ARABIC }: GOOGLE STOCK OVER 7 YEARS

The above chart shows some of the types of patterns I hope that analysis of the price data might uncover: pattern 'a' is a spike followed by a large dip, pattern 'b' is a double spike, pattern c is steady (albeit irregular) growth.

By training the machine learning algorithms on the sequences of prices, it is anticipated that they may be able to see that a sudden spike in prices will lead to a sudden price drop soon, that often a spike is followed by another spike, and so forth.

Algorithms and Techniques

I have used several techniques for comparison, of which should be good candidates to providing some insight into this problem. The problem itself is not a classification one but one that involves continuous data, which results in selection of the following algorithms:

- Linear Regression: useful where there is a definite, linear trend to the data. Linear Regression was selected as it is very good at finding (as per it's title) linear relationships within data. Linear regression looks to find a line that provides least squares error in accordance with the data points, which could work well for this project in the case if the stocks are simply progressing up (or down) over time. It may not perform as well when the pattern is more complex than a straight line.
- Support Vector Machine: a variant of the SVM, the SVR, is very good at determining multinomial factors in data, and can work with continuous data. Support Vector Machine was chosen as I have had good results with this in other Udacity projects, and am curious to see how it performs here. It is also a methodology that has a variant (SVR) of which can work with continuous data. A SVM is a nonlinear classifier that looks to find a hyperplane (with or without kernel trick) to separate data into classes. The SVR is a variant of this of which uses a non-linear kernel to provide a continuous output instead of a classification. This should perform well here on the more complex stocks that don't look to follow a simple linear regression, and may be able to detect some of the more complex patterns that can be seen. The main problem with this algorithm is in overfitting: by tuning the hyper-parameters I hope to mitigate this.
- Neural Network: I selected Neural Network more for my own curiosity to see how this type of ML algorithm would perform on a regression task, and boost my learning in the process. The Neural Network should have a good chance of performing well, however tuning to the problem and the dataset will be the big problem. The type of network chosen is a single layer unidimensional regression network consisting of 2 layers. Neural Networks work by accepting input, passing it through an activation function to see if meets an expected output, and then adjusting a weight parameter if it does not, then repeat this process for several epochs. Neural Network has the potential to perform well here, but it will be very dependent on tuning.
- Recurrent Neural Network: a bit of a 'sequence specialist', very interested to see what can come of training a RNN over sequential stock data. RNNs have had good success in predicting sequences in other fields, I put this in for my own learning as well as seeing if it can be adapted for this task. An LSTM RNN is a special type of RNN, each LSTM unit contains 4 interacting layers that perform operations to progressively forget (or reinforce), based on gate values, what it has learnt in the past. As we are mainly dealing with sequences, and these sequences can disappear over time (i.e. a company can be turned

around, and erratic stock price behavior of the past dissipate), LSTM RNN has great potential here if I can get it right, but tuning the parameters will be the difficult part.

Benchmarks

It is difficult to provide a benchmark without having known the trading strategy or the audience using the software, but as a guide I will aim for:

- Bettering of the score returned by simple Linear Regression
- Within 5% of the actual stock price 1 week ahead
- Within 10% of the actual stock price 2 weeks ahead
- Variance of return ideally within 10% (but unlikely)

Methodology

Data Preprocessing

The initial data preprocessing is in the form of fetching data within the selected date periods on the set of stocks, and picking a metric (e.g. close price, open price), and then running some analysis on the data to determine which stock we will investigate further.

The data is saved to disk to prevent getting locked out from the Yahoo! Finance servers. A Pandas data frame is constructed from the chosen metric (e.g. Close Price) and the stock in question.

Next I forward fill any data – the stocks may be in different markets and have different public holidays. Forward filling lets us assume that if the stock did not trade on that day, that the effective price was the same as the day previous. Then I backfill any data, given that for some stocks we may not have history all the way back for the dates provided. The method chosen here is to prevent any skew from nulls.

Further preprocessing stage involves calculating the number of business days ahead, translating the date field into an integer (number days since data 'start') and then forward filled, given that some stocks will be in different markets and may not have traded on the day. Outliers are left in the data: this was a difficult decision and I ended up deciding on leaving the outliers in, as the outliers sometimes are part of the pattern that we want to detect, and if I did choose to exclude the outliers, then I could not find a way to exclude then that would not negatively affect the results. E.g., if I removed the outlier, then for that time step there would be a null, affecting the learning. If I normalized the outlier to mean, then this too would negatively affect the outcome as the algorithm would look for it in sequence as a pattern.

The data is then lagged in preparation for sending to the ML algorithms: the data is a sequence leading up to the end date, and the label being the value on the period ahead.

The means the rows are the sequences leading up to the day of length X (where X is determined empirically, and can be thought of as another tuning parameter: a minimum number of consecutive days in which to observe any pattern) and the label is the value at the number predicted days ahead, in the above example 2 days ahead.

For all examples except RNN, the data is then normalized using the **Scikit** Learn '**StandardScaler'**. SVM/SVR are not scale invariant and it is recommended to scale the data to either within [-1,+1] or [0,1] before processing. It is also recommended that Neural Networks have their input data be scaled between [0,1], so a StandardScaler was chosen as this defaults to [0,1] range. There doesn't seem to be any benefit to scaling or not scaling data for Linear Regression, so it is passed the scaled data for convenience.

In the case of the RNN, the data is not normalized: most RNNs seem to be OK with not normalizing the data, and I seemed to get worse performance when running normalized data.

Implementation

Linear Regression

The normalized data is split into train and test sets using the scikit-learn 'train_test_split' function. A random seed of 42 is used so that results are reproducible. A test/train split of .25 is chosen (25% of the data will be for testing, 75% for training).

Parameters of 'fit_intercept' [True, False], 'normalize' [True, False] and 'copy_X' [True, False] are passed to GridSearchCV for parameter space exploration during training. The model is trained using the 'fit' function. In order to make predictions, the last line of data from the train set is fed to the model.predict function.

Support Vector Machine

The normalized data is split into train and test sets using the scikit-learn 'train_test_split' function. A random seed of 42 is used so that results are reproducible. A test/train split of .25 is chosen (25% of the data will be for testing, 75% for training). Parameters of 1 and 10 for C, as well as 0.1, 0.01, 0.001 for epsilon are given to GridSearchCV. C is the penalty associated to the instances which are either misclassified or violates the maximal margin. Using C is great in cases when we are mapping the samples to a higher dimensional space using the kernel function.

The parameters were derived from the recommendations in the documentation, as well as some experimentation (I found that in most cases the optimal C value is 10 and the epsilon 0.01, allowing a degree of magnitude either side for epsilon and one below for C (documentation recommends keeping C values low, near 1 if possible) would provide enough flexibility). An R2 scorer is set up and used as the scoring function to find the best model parameters. Adding in cross validation and shufflesplit resulted in a longer runtime but large improvement in the r2 scores.

Neural Network

The neural network started as something simple: just 2 layers and tanh activation function.

Initial r2 scores were around 0.85 and worth pursuing, and experimentation with addition of layers and activation functions resulted in r2 scores in the range of 0.9.

The model appeared to be overfitting though, and addition of the dropout layer started to show more epochs required for training but a more accurate and reliable set of results.

Dropout is a technique where randomly selected neurons are ignored during training. They are "dropped-out" randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

As a neural network learns, neuron weights settle into their context within the network. Weights of neurons are tuned for specific features providing some specialization. Neighboring neurons become to rely on this specialization, which if taken too far can result in a fragile model too specialized to the training data. This reliant on context for a neuron during training is referred to complex co-adaptations.

You can imagine that if neurons are randomly dropped out of the network during training, that other neurons will have to step in and handle the representation required to make predictions for the missing neurons. This is believed to result in multiple independent internal representations being learned by the network.

The effect is that the network becomes less sensitive to the specific weights of neurons. This in turn results in a network that is capable of better generalization and is less likely to overfit the training data.

Recurrent Neural Network

The RNN follows the Keras documentation for putting it together: most of the configuration is in the methodology of how to structure and predict the data. Initially I had the step size set to 1, and predictions were fed back into the model to determine X days ahead, e.g. the model was trained to predict 'tomorrow's' price, hence to pick the price in 5 days' time, I would then pick the price for tomorrow, and then add that to the previous days, make another prediction based on that for the day after, and so forth.

I found this approach not very reliable, and each step seemed to introduce greater probability of going on a tangent. I settled on stepping the data, like the other approaches, and then predicting the step value out.

Results

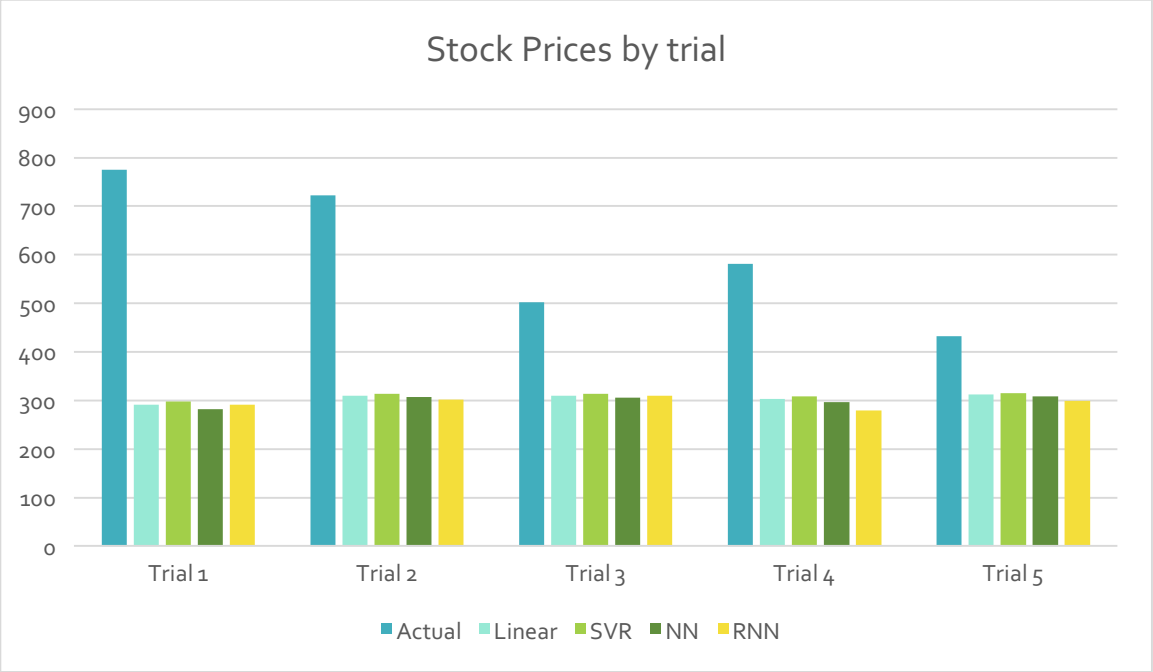
I ran 5 trials, with different dates for each and different stock subsets to try and gauge average performance statistics for the different methodologies:

Trial 1:						
Start 2010-01-01, end 2016-12-30, query 2016-09-29						
Selected Stock: GOOG	**Days Ahead: 20**					
	Train Score	**Test Score**	**Actual Price**	**Predicted Price**	**% Difference in price**	**% Variance of return**
Linear Reg.	0.9716	0.9697	775.01	290.992	62.4530 %	7769.1593 %
SVR	0.9736	0.9707	775.01	298.002	61.5486 %	7656.6503 %
NN	0.9741	0.9721	775.01	282.40948	63.5605 %	7906.9346 %
RNN	0.9774	0.9802	775.01	290.78	75.963%	8108.96%

Trial 2:	
Start 2010-01-01, end 2016-06-01, query 2016-06-03	
Selected Stock: GOOG	**Days Ahead: 2**

	Train Score	Test Score	Actual Price	Predicted Price	Difference in price	Variance of return
Linear Reg.	0.9964	0.9963	722.34	309.39	57.1687 %	3496.6329 %
SVR	0.9951	0.9947	722.34	313.23	56.6372 %	464.1251 %
NN	0.9961	0.9958	722.34	306.74	57.5340 %	3518.9748 %
RNN	0.9674	0.9602	722.34	301.98	53.5340 %	3818.9441 %
Trial 3:						
Start 2010-01-04, end 2015-01-02, query 2015-01-6						
Selected Stock: GOOG	**Days Ahead: 4**					
	Train Score	**Test Score**	**Actual Price**	**Predicted Price**	**Difference in price**	**Variance of return**
Linear Reg.	0.9964	0.9963	501.96	309.38	38.3643 %	842.7715 %
SVR	0.9951	0.9947	501.96	313.227	37.5995 %	825.9700 %
NN	0.9775	0.9808	501.96	305.49	39.1402 %	859.8150 %
RNN	0.9959	0.9975	501.96	310.227	39.1402 %	857.9866 %
Trial 4:						
Start 2010-01-04, end 2014-08-29, query 2014-09-09						

Selected Stock: GOOG	Days Ahead: 7					
	Train Score	Test Score	Actual Price	Predicted Price	% Difference in price	% Variance of return
Linear Reg.	0.9883	0.9882	581.01	302.76	47.8904 %	2956.9480 %
SVR	0.9879	0.9869	581.01	308.30	46.9370 %	2898.0799 %
NN	0.9885	0.9875	581.01	296.36	48.9926 %	3024.9976 %
RNN	0.9872	0.9860	581.01	279.98	49.3478 %	2998.998 %
Trial 5:						
Start 2010-01-04, end 2013-08-19, query 2013-08-20						
Selected Stock: GOOG	Days Ahead: 1					
	Train Score	Test Score	Actual Price	Predicted Price	Difference in price	Variance of return
Linear Reg.	0.9983	0.9981	432.27	311.918	27.8433 %	104772.4319
SVR	0.9971	0.9966	432.27	314.857	27.1633 %	102213.6254 %
NN	0.9980	0.9976	432.27	308.55	28.6229 %	107706.0398 %
RNN	0.9774	0.9802	432.27	289.918	31.8472 %	134772.483



Results summary

Several interesting observations I have found are as followings:

- It seems that for some cases, all algorithms meet a somewhat 'optimal' score on the train and test sets: i.e., they all report very similar scores (e.g. in trial 1) for both test and train sets, suggesting that each model has possibly found the best possible relationship within the data (albeit not a perfect correlation)
- Overall, the neural network performs best when averaged out across all scenarios, followed by the SVR, then the Linear Regression, and the RNN last. The RNN is very sensitive to tuning and conditions and I feel if more time is available, comparable or better performance could be obtained.

- Some stocks are easier to predict than others. E.g. the SIP (Sigma Pharmaceuticals) was one of the better performing stocks in the ASX this month, but as they have had a few years of little movement neither algorithm could predict the big bump in stock price this month.

- On average, every approach could get within 5% of the price. However, variance of return was on average in the order of 100%, which to me would not be an acceptable risk for trusting this as an investment tool. However, interestingly and probably worthy of more research, in most cases the predictions are ***under*** the actual price, meaning that the variance is positive and would mean that

extra money would have been made as opposed to lost.

- Top score was obtained by the neural network, predicting the RIO price 2 days ahead to within 0.0626% and a variance of return just over 10%. However, the total stock price movement in this period was just 0.57% meaning that we would need to invest $100,000 just to make $570.
- In some cases, the test score is higher than the train score! What I think is happening here is that the train test split has randomly allocated more results that are 'easy' to predict, or conform with the model, into the test set. I have regarded these as a statistical anomaly, and running the test again but with a different random seed in train test split returns different results, partially confirming my suspicion.

Conclusion

The project met most of its goals. We could determine stock price within 5% within 5 days out, and sometimes even 10 days out. The project was not able to get variance of return consistently around 10%, only shorter-range predictions could satisfy this. I could beat Linear Regression, sometimes.

Variation in performance is observed over different datasets: some stocks are easier to predict than others. Some time periods are easier to predict then others. For example, in the worst performing trial, all algorithms were out by 20%. However, on inspection of the stock price we can see that the stock has trended flat for a very long time, and then only recently started gaining in value, and then during the prediction period experiences a short, sharp spike unseen in the stock history. This is a pattern that was not known or learnt by the algorithms, and they were unable to predict it.

Conversely, the best performer (Trial 3) I can see that the stock approximates a sinusoidal line during the selected period, decreasing in amplitude and tilted right. Here the algorithms would be able to determine a function that represents the data reasonably well, and this is reflected in the results.

Using machine learning in the methods described above could have potential; however, I think more work needs to be done in preprocessing the dataset. Namely:

- Rather than a single sequence, perform initial data analysis using PCA or correlation chart to find stocks that are positively or negatively correlated
- Take sequences of all these stocks and pass this to the ML algorithm: with more market

information, potentially better predictions can be made as the ML can learn more trends and patterns

- Identification of market trends prior to assessment of stock: potentially using ML to look for patterns in the market that may lead to stock price increases overall. One of the problems identified is that even when the ML is very accurate, the market volatility was low and there was not much to be gained by trades
- Better processing of outliers and non-trading days

Regardless, the results are encouraging. The difficulties in this project were deciding on ways of how to predict, how to structure the data for the ML to best use it, dealing with data where there is no data (i.e. days where the markets are closed), working with outliers. The Neural Networks added a great deal of difficulty in just getting them to run and not return poor results. The way I worked around this was to experiment with hyper parameter tuning: however, this makes the run times incredibly long (exploring the parameter space, changing layer sizes, adding optimizers etc.) would mean adding additional weeks to this project, so a 'best approximate fit' set of parameters were chosen based on an initial dataset, and then only slightly altered as required to fit other datasets. There are still problems though – if the predicted date is not very far out, or the amount of data supplied to the network is not great, the network trains over too many iterations and starts to overfit, resulting in poor performance. Ideally the number of epochs would be relative to the amount of queried history.

In working with the structure of the data, many options came to mind. One was to use the date as data and then use the price as a label, but this I felt would not have given enough information to be able make any meaningful prediction. Another option I considered was to use machine learning to determine correlated stock pairs, but then the problem of how to predict X days ahead can't be done unless we can predict the correlated stock's price. The sequence idea (train the ML on a sequence of prices to determine the next price) seemed best given the project description as it allows prediction any number of arbitrary days ahead.

Filename: Capstone Project.docx
Folder: /Users/nazanindelam/Library/Containers/com.microsoft.Word/Data/Documents
Template: /Users/nazanindelam/Library/Group Containers/UBF8T346G9.Office/User Content.localized/Templates.localized/Normal.dotm
Title: Stock Predictor
Subject: Machine learning Nanodegree Capstone
Author: Nazanin Delam
Keywords:
Comments:
Creation Date: 2/19/17 5:17:00 PM
Change Number: 2
Last Saved On: 2/19/17 5:17:00 PM
Last Saved By: Nazanin Delam
Total Editing Time: 1 Minute
Last Printed On: 2/19/17 5:17:00 PM
As of Last Complete Printing
Number of Pages: 21
Number of Words: 5,049 (approx.)
Number of Characters: 28,782 (approx.)