Machine Learning Engineer Nanodegree

Capstone Project: Stock Price Prediction

Sam Whitehill, CFA
February 12, 2017

Stock Price Prediction Background

Stock market prediction is an age old challenge which is as rich in techniques and models as it is in criticisms and refutations. The research and exploration in this field goes back many decades. Given the large reward for being even mildly successful over even a short period of time, there continues to be substantial research as technology advances. More recent research includes many prediction studies done within the context of machine learning.

Stock price prediction is of course a type of time series forecasting, to which machine learning can be readily applied. There are numerous articles which discuss using popular machine learning algorithms[1] such as Support Vector Regression (SVR) [2] and Neural Networks[3], among others. Thus, stock price forecasting is an interest problem to examine within the realm of machine learning.

Problem Statement

The problem to solve is the prediction of stock prices over various future time periods. These future time periods are one day ahead and five days ahead of the current time. The prediction of these future stock prices is done indirectly, by actually predicting the daily change in the log of the adjusted closing prices, and not simply the future price itself. The reason for this indirect prediction approach is because historical stock prices are widely believed to be non-stationary, or exhibiting a mean and variance which changes over time (among other issues).

This non-stationarity issue introduces a significant challenge in using a time series forecasting model. However, this challenge is largely if not completely resolved, by forecasting the differences of the log of the stock price instead of the actual prices. These computed log-differences are quite arguably stationary, and generally mitigate this stationarity issue[4]. This capstone project studies the effectiveness of a LSTM RNN[13] model in predicting future stock prices. More precisely, this model is a long short-term

memory recurrent neural network and will be referred to as an "RNN" from here onwards.

The effectiveness of this model is measured primarily by the Accuracy Ratio[4], which is essentially the ratio of the forecasted outcome over the actual outcome. The Accuracy Ratio overcomes shortfalls with the more popular MAPE (Mean Average Precision Error) measure, which has a bias towards forecasts that are too low, among other drawbacks. The Evaluation Metrics section discusses this measure in more detail later.

The input data to this problem is historical stock price data and computed technical indicators, while the output is the predicted stock prices over a specified time period (i.e., one and five days ahead). The features used include the historical stock prices as well as the technical indicators described in the dataset and inputs section.

The solution to this stock price prediction problem is to use historical stock prices and technical analysis indicators within an RNN model, to forecast future adjusted closing stock prices. Various features and lookback windows (i.e., how many past historical days) are fed into the RNN model, in an attempt to produce a successful forecast. The solution's success is measured by the Accuracy Ratio, which essentially quantifies how well the model can predict future stock prices (further described in the evaluation metrics section). The solution seeks out the combination of model parameters, features and lookback windows that produce the best Accuracy Ratio. The RNN model parameters (e.g., learning rate, number of neurons, layers etc.…) are tuned via grid searching, in order to maximize the Accuracy Ratio. This process can be easily repeated over the different training periods.

Evaluation Metrics

The Accuracy Ratio will be used to evaluate the performance of the RNN model. It is calculated by first taking the natural log of the ratio of the forecasted price divided by the actual price. This ratio calculation is then aggregated over the series of forecasted prices by taking the sum of the squared ratios, in order to summarize the model performance in one metric. More formally:

$$\sum_{t}^{n} \log(\mathrm{Pt} / \mathrm{At})\text{^}2$$

In this equation, t is the day number out of n total number of days in the forecasted dataset. For example, if a single stock's price is forecasted on a daily basis for the next 100 days, then n=100, and P_t and F_t are the forecasted and actual prices on a particular

day, respectively. Ultimately, the sum of the square of the natural log of the ratio between predicted and actual prices would be computed over each of the 100 days in question.

This evaluation metric is measured across multiple ETFs and different historical time periods in order to assess which type of ETFs and type of markets are best forecast by the RNN model. That is, the model is evaluated under different economic environments (e.g., recessionary and expansionary). Further, multiple forecast periods (i.e., number of days ahead) are also evaluated to see whether short term or possibly longer term forecasts perform best.

Datasets and Inputs

The RNN model studied in this project largely uses the principles of technical analysis[6], in order to forecast future stock prices using historical price data and other dependent features. This historical stock price data is obtained from Yahoo's financial website[5], via the Python API "get_data_yahoo", which is housed within the Pandas.io module[7]. This includes daily opening, closing, low, and high prices as well as the volume traded, for a particular stock over some historical time period. In addition to the historical price data just mentioned, various technical indicator features are calculated using common algorithms from the field of technical analysis. *Note that the following technical indicator features are calculated on the differences of the logged prices and not on the actual prices, to remain consistent with the stationarity of the time series*. These technical features[6] include:

1. MACD (Moving Average Convergence Divergence)
2. Force Index Volume indicator
3. EMV (Ease of Movement) indicator
4. Real Body Candlestick indicator
5. Upper Shadow Candlestick indicator
6. Lower Shadow Candlestick indicator
7. Volume of shares traded
8. Rolling standard deviation of logged price differences
9. OBV(On balance volume) indicator
10. Slope of the volume over lookback window
11. Adjusted Closing Price
12. Sine wave function fit amplitude parameter
13. Sine wave function fit frequency parameter
14. Sine wave function fit phase parameter

15. Sine wave function fit offset parameter

These features are explained in more detail later.

The model will forecast the future price of these four ETFs (Exchange Traded Funds):

1. SPY
2. QQQ
3. VXX
4. OIL

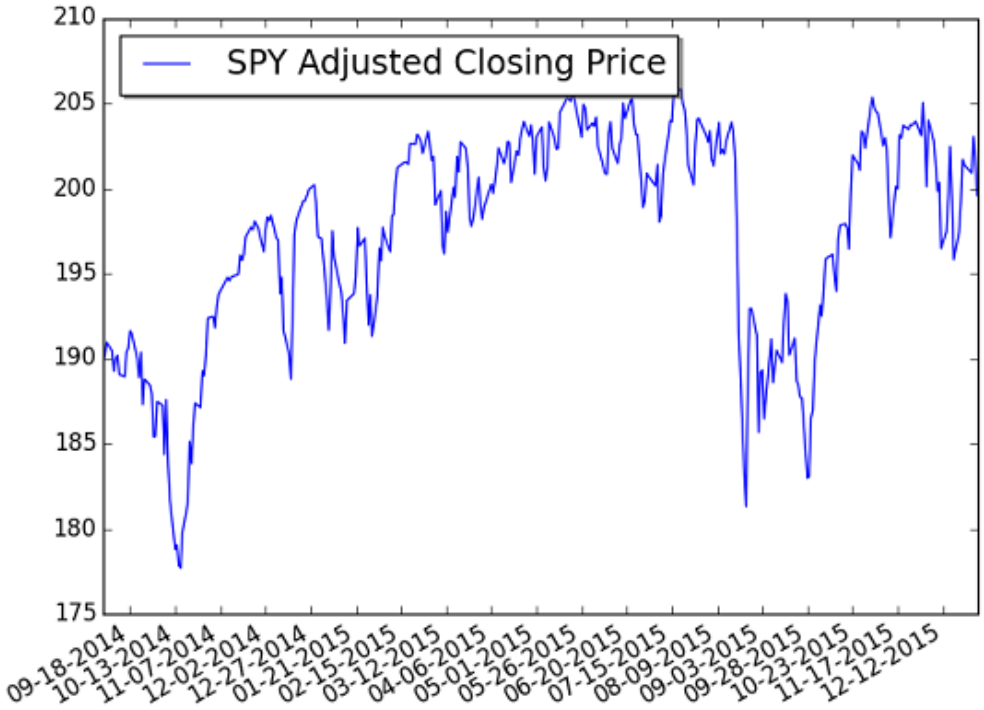
For each of these four ETFs, the following date ranges are used for training and testing:

ETF	**Training Data Range**	**Testing Data Range**
SPY	Sept. 3, 2014 - December 31, 2015	January 1, 2016 - May 31, 2016
QQQ	March 10, 1999 - January 31, 2000	February 1, 2000 -July 31, 2000
VXX	March 1, 2009 -January 29, 2010	February 1, 2010 -July 30, 2010
OIL	February 1, 2007 - March 31, 2008	April 1, 2008 - August 29, 2008

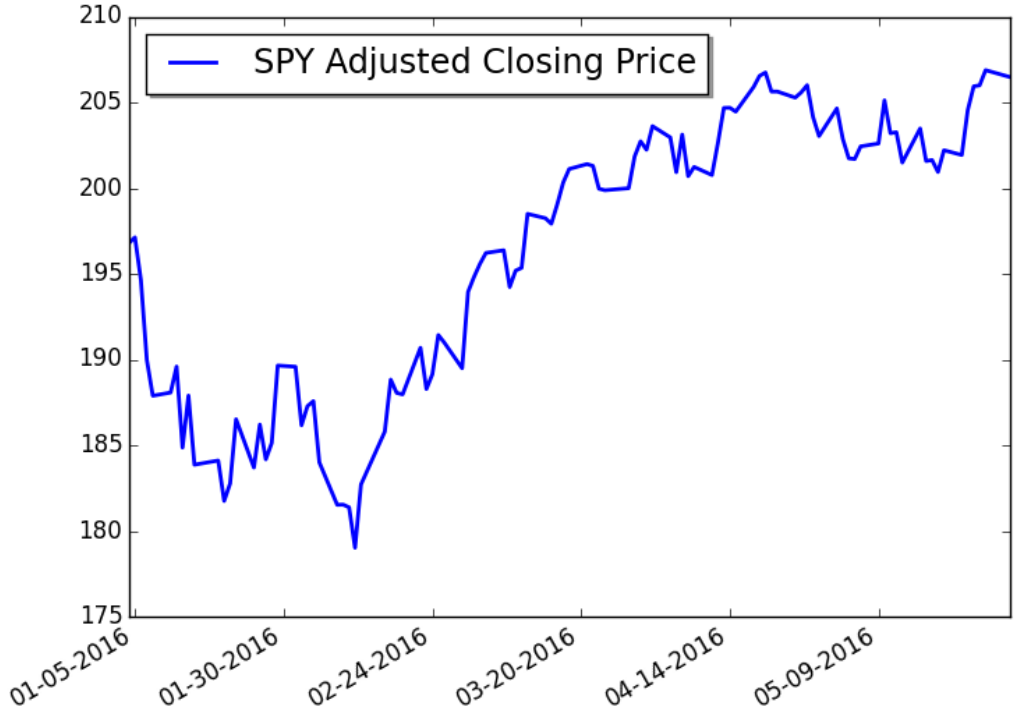
The following charts illustrate the market characteristics of these four ETFs mainly in terms of their volatility and trending behavior.

First, the adjusted closing prices for the ETF: SPY are illustrated from 2015 to 2016. SPY tracks the S&P 500 index, which is a widely considered a proxy for the stock market in the United States. The following two charts show the training and testing data for SPY.

Training Data: SPY Historical Prices

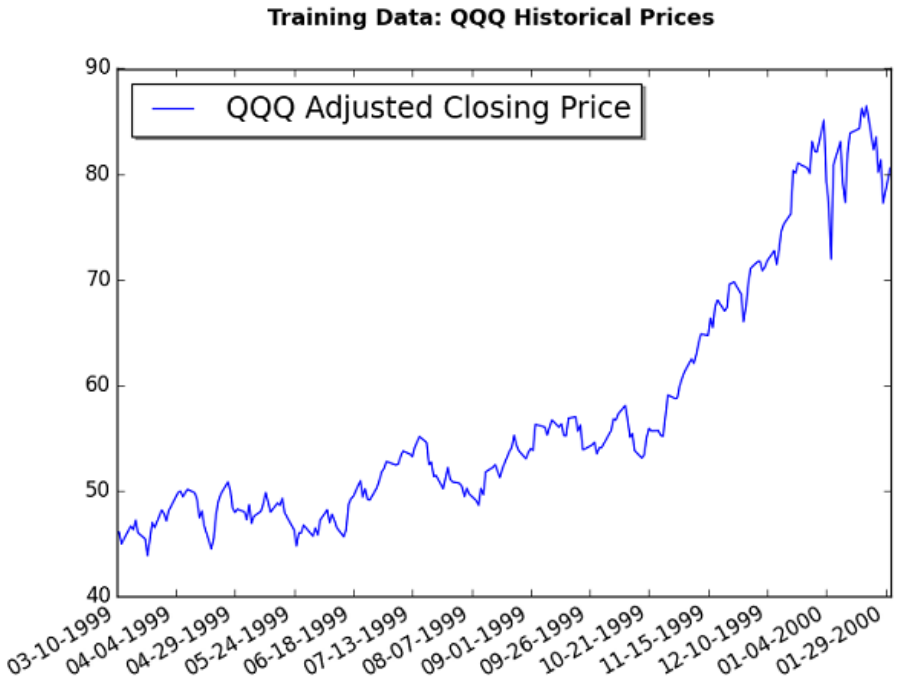


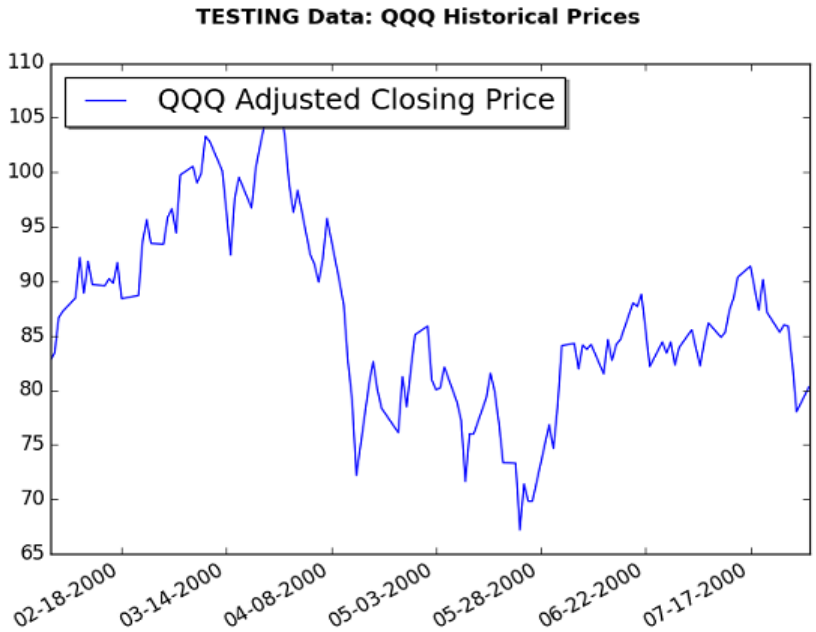
TESTING Data: SPY Historical Prices



What is noteworthy about these two SPY plots is that the training data consists of a fairly volatile and somewhat trending series, while the testing data appears to consist of a more stable, trending market.

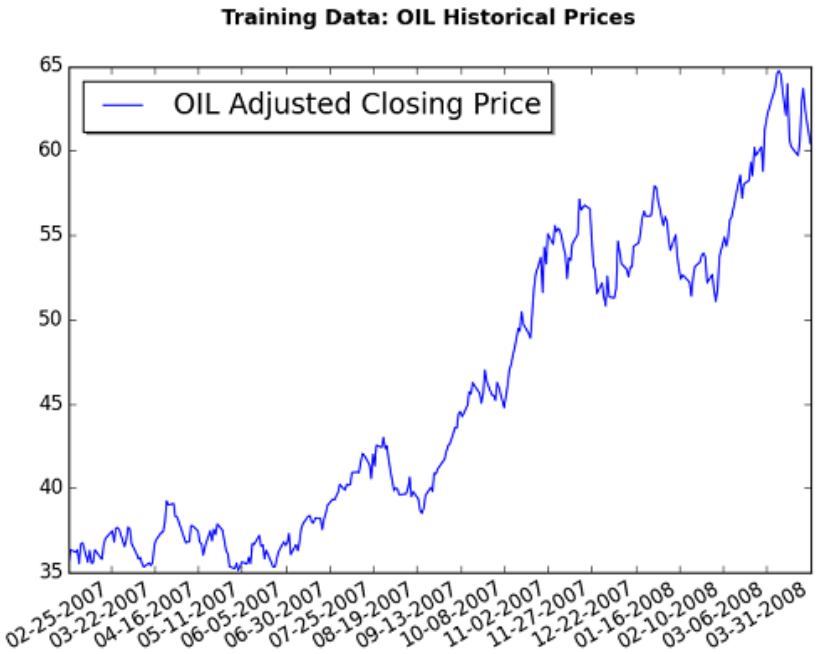
Second, the ETF: QQQ which tracks the NASDAQ 100 index is illustrated over the period from 1998 to 2000.

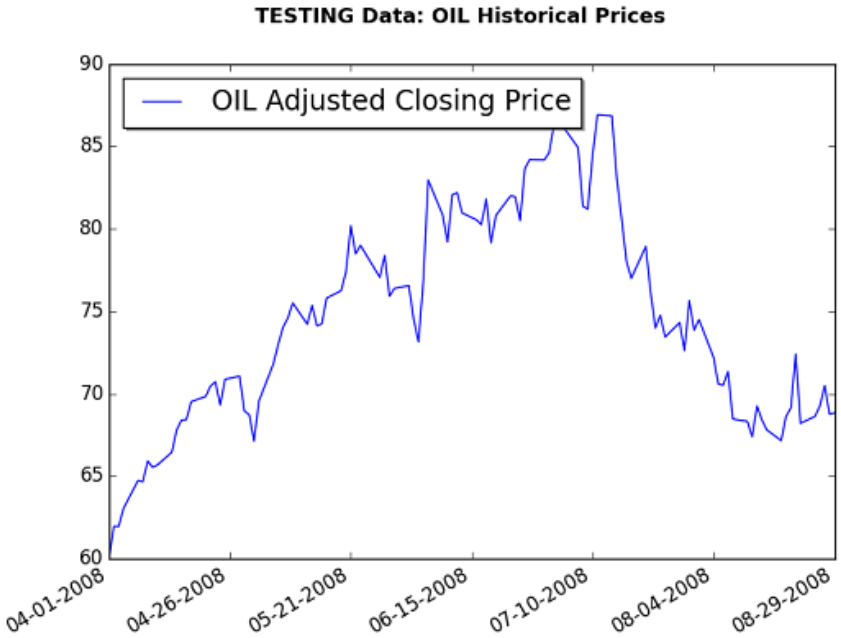




Here the QQQ index shows the bubble type run up to the dot com crash of 2000 over the training data, while the testing data shows the susequent crash and increased volatility. This should be a very challenging prediction.

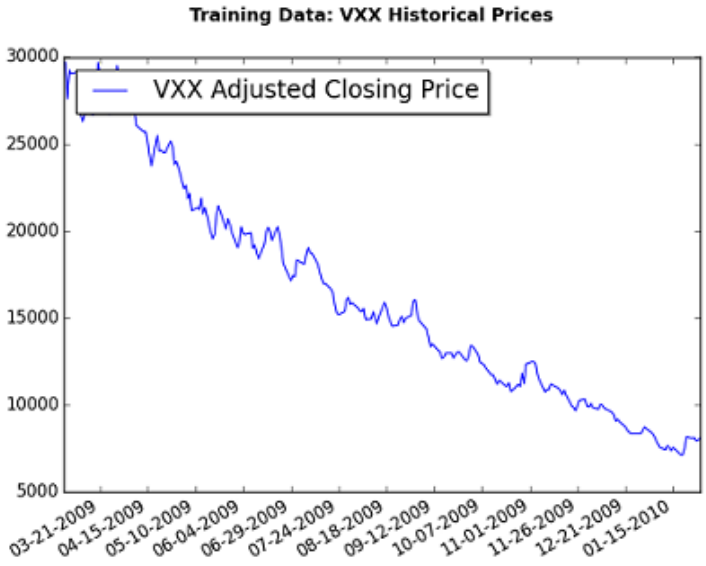
The third ETF forecasted is the ETF: OIL, which tracks the price of oil.

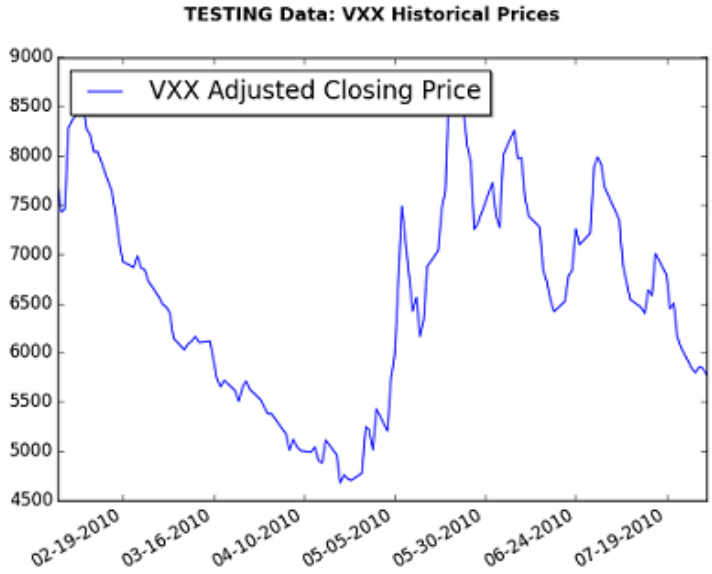




What is noteworthy about the OIL plots is the parabolic uptrend in the training data followed by the crash in the latter part of the testing data.

The last ETF forecasted is the VXX. VXX tracks exposure to the CBOE Volatility Index. This ETF is interesting because it is inherently volatile and typically in a downtrend, as oppposed to the more common uptrend that occurs across the US stock market.





The VXX training data is in a clear downtrend, while the VXX testing data contains more reversals and volatility spikes.

In addition to these ETFs and their date ranges, **several lookback windows** of historical data (i.e., how many days' historical data to be used in the model) are specified in terms of the number of historical days prior from the initial start date to use in the model's forecast. One window is specified purely for the transformed price data, while two other windows are used in calculating the MACD as well as the, slope, and rolling Standard deviation features. The forecast time periods are one day ahead and five days ahead.

Algorithms and Techniques

The LSTM RNN model is chosen for its strong ability to forecast time series data. More specifically, the ability of the recurrent neural network to handle sequence dependence is one advantage over other neural networks and other machine learning algorithms in general. Further, the LSTM (long short-term memory) RNN is capable of training large architectures[19]. Last, the LSTM model is not susceptible to the vanishing gradient problem, as compared to other neural networks.

For this stock prediction problem, a sequential and stateful LSTM RNN model is implemented. The reason for explicitly choosing a stateful model is the potential for the model to remember the sequences of the stock price changes and use them in future predictions. The sequential choice fits with the problem statement of predicting a sequence within a time series dataset.

The RNN is optimized using the RMSProp optimizer and the model is compiled using the mean squared error loss function. The number of neurons and layers used, the learning rate and the dropout percentage are all configured per the dataset chosen (i.e., ETF) via grid searching. That is each dataset will potentially use a different network structure and set of parameters depending on the grid search results.

Since the RNN model is stateful, the state of the network must be reset after each training epoch and also just before making predictions[19]. This requires explicitly resetting the state of the network after **each** training epoch as well as resetting it before performing predictions. Further details of the RNN model construction are discussed in the implementation section.

Benchmark Model

The benchmark model used to compare against the RNN model is simply a naïve[8] no change forecast, such that the predicted value (or predicted log first difference) is merely the last known value. More specifically, since the time series is actually converted into a stationary one using first differences, the benchmark simply uses the last known

change in the log adjusted closing price. This naïve benchmark is actually not so easy to beat and serves its purpose. This benchmark model is evaluated using the same Accuracy Ratio as in the RNN model.

Data Preprocessing

The dataset initially consists of historical stock price data (e.g., open, high, low, close, volume). As previously discussed, this data is non stationary and therefore poses a significant problem when performing a time series forecast due to the changing mean and variance. Therefore, the historical stock data is transformed into a stationary time series.

This transformation is a two step process. First, the adjusted closing prices are converted into their natural log values. Next, first differences are computed over the time series by calculating the one day difference between the natural log of the adjusted closing price at the current day and same value on the previous day. More formally, the following equation is used to calculate the log price from prices:

$$\text{Log Price} = \text{Ln}(\text{Price}_t)$$

$$\text{First Difference} = \text{Ln}(\text{Price}_t) - \text{Ln}(\text{Price}_{t-1})$$

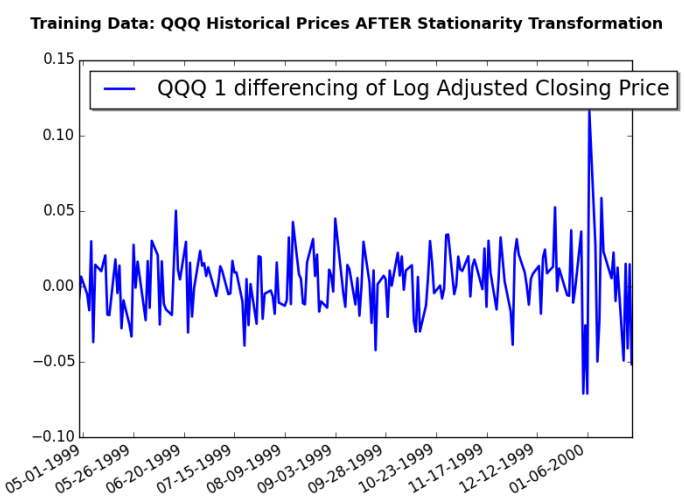
t =current time step or day , t-1 is the prior day

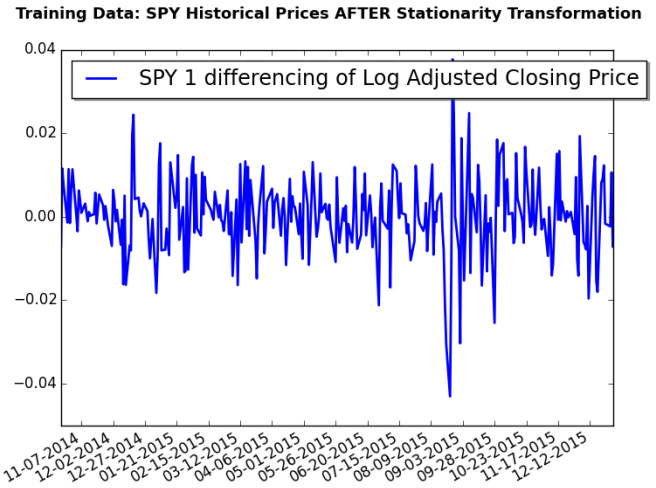
The natural log transformation helps to stabilize the variance across the time series[17]. This first differencing stabilizes the mean of the time series. Both of these transformations combine to change the stock time series from a non-stationary series into a stationary one. That is, a series with a constant (or nearly) mean and variance

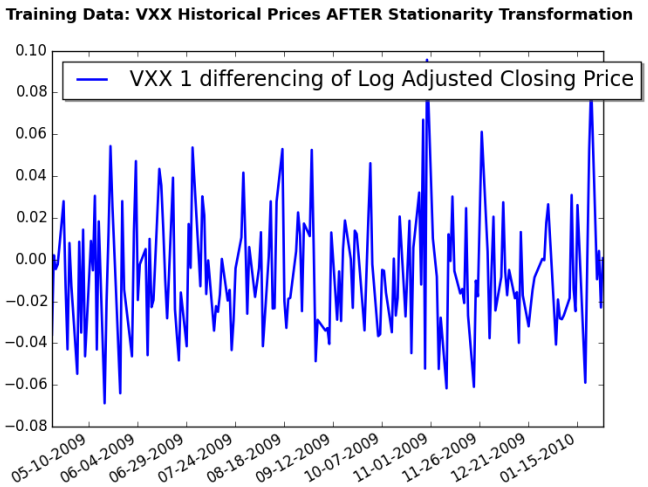
This stationarity is tested using a Dicky-Fuller test at the 90% confidence level, by comparing the DF test statistic against the critical value at the 10% level. If the test statistic is lower than this critical value, then the transformed stock data is considered stationary. Below are the Dicky-Fuller tests for the four **training** data sets:

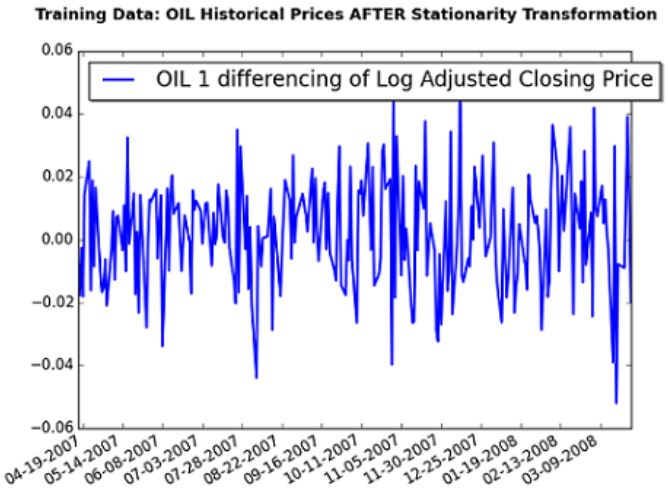
ETF	Dickey-Fuller test statistic	Critical Value (10%)
SPY	-17.16	-2.5714
QQQ	-16.15	-2.57935
VXX	-15.716	-2.5735
OIL	-18.86	-2.5721

The table shows the training data set are deemed stationary by the Dickey-Fuller test (at the 10% critical value) as all test statistics are less than their associated critical values. A plot of each of the transformed datasets is shown below:









Implementation

The implementation can be conceptually broken down into the following components:

1. Data retrieval
2. Data transformation
3. Feature generation
4. Data preprocessing and feature scaling
5. Model construction
6. Model tuning
7. Model prediction and testing

The first component consists of retrieving historical stock price data from Yahoo's financial website according to user specified inputs for the stock, prediction period and lookback windows. As discussed earlier, there three lookback windows to use when performing a forecast, as using multiple historical time periods is very valuable in attempting to produce accurate forecasts.

For example, suppose that one requests to see a forecast of the stock SPY, for one day ahead of the date: December 15, 2016. Further, suppose that one specifies a lookback window of 8 prior days. That is, use historical trading data 8 days before December 15, which starts on December 6th. This amounts to retrieving historical data for the stock SPY starting on 12/6/2016 and ending on the forecasted date of December 16, 2016.

Next, the data must be transformed from price data into the first differences of log prices due to the non-stationarity issue previously discussed. After this transformation the technical features are generated.

Feature generation is performed to generate the technical indicators discussed in the previous dataset section.

The features are calculated as follows:

1. The MACD (Moving Average Convergence Divergence) is calculated by subtracting the 26-day exponential moving average (EMA) from the 12-day exponential moving average EMA.
2. The Force Index Volume indicator[15] is calculated by subtracting yesterday's close from today's close and multiplying the result by today's volume. If closing prices are higher today than yesterday, the force is positive. If closing prices are lower than yesterday's, the force is negative.

3. The EMV (Ease of Movement) indicator[16] is calculated as [(High(today)+ Low (today)) / 2 - (High(yesterday) + Low (yesterday)) / 2] / (Volume / (High (today) - Low (today)))
4. The Real Body Candlestick indicator is calculated as the (Closing Price –Opening Price) divided by the Opening price
5. The Upper Shadow Candlestick indicator is calculated as:
   If the Closing price is >Opening Price then (Closing Price -Opening Price) divided by (High -Open Prices), otherwise, calculate the (Opening price - Closing price) divided by the (High - Closing Prices).
6. The Lower Shadow Candlestick indicator is calculated as:
   If the Closing price is >Opening Price then (Closing Price -Opening Price) divided by (Close -Low Prices), otherwise, calculate the (Opening price - Closing price) divided by the (Opening - Low Prices).
7. Volume is simply the number of shares traded per day
8. Rolling standard deviation is simply the standard deviation of the adjusted closing prices over a look back window on a rolling basis
9. OBV(On Balance Volume) indicator is calculated by adding the current day's volume to a cumulative total when the stock's price closes up and subtracting the current day's volume when the stock price closes down.[18]
10. Volume Slope is the slope of the regression line[21] which best fits the daily volume over the lookback window
11. Adjusted closing price is simply the historical closing price adjusted for stock splits, etc….

The last four parameters (sine wave) are calculated by fitting the following custom sine wave function to the adjusted closing prices within the lookback window. The sine wave function is:

sin(x * frequency + phase) * amplitude + offset

x is the log difference of the adjusted closing price

The frequency, phase, amplitude and offset parameters are fit to the time series within the lookback window using a non-linear least squares approach[22]. These four parameters comprise the last four features.

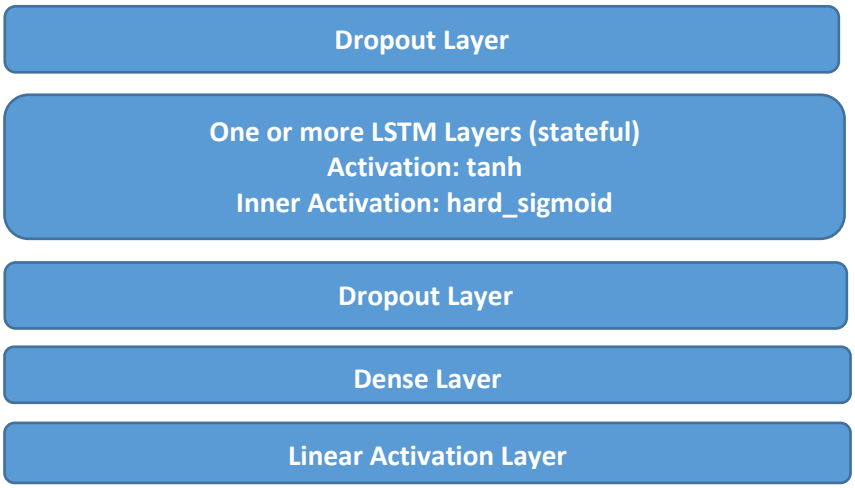
The next step in the project design is to preprocess the data. This primarily involves manipulating the data such that historical price differenced data and associated features (spanning multiple days) is stored in the format which the RNN model expects. This is specifically in the (samples, time steps, features) array format[19]. The RNN model requires a specific format for the historical data before it can be trained. This does have a small

impact on run time performance as the feature data must be translated into the correct format.

After all features have been built and processed, they are then scaled via the MinMaxScaler from (-1 to +1). This scaling is critical to prevent features with much bigger variances (i.e. volume traded) from skewing or negatively impacting the model performance.

The last preprocessing step involves building the training and testing data sets to essentially match the lookback window and forecast horizon. The key aspect of this step is the construction of the target values. When the horizon is simply one day forward, then each sample corresponds to one target value (i.e., the next trading day's value). However, when the horizon is two more days, then the target value is actually a set (e.g., array) of ***multiple*** values corresponding to the horizon number. For example, if the horizon is five days ahead of today, then the target per sample will include the next five trading day's values. This is to accommodate the RNN forecast model, which would also sequentially forecast the next five days values in this example.

The next step in the implementation process is to construct the RNN model. A sequential LSTM RNN model is built as shown below:



In addition, the learning rate uses a custom decay function which reduces the learning rate after each epoch iteration via the following equation:

Learning rate after each epoch = initial Learning Rate *exp(0.000065 * epoch)/(1 + exp(0.006 * (epoch-600)))

The purpose of this custom learning rate decay function is to effectively improve model performance by making the learning rate drop sharply after a specified number of epochs, and then level off again.

Despite the power of stacking *multiple* layers in LSTM networks, grid search results showed the best performance with only one single layer. Hence, this single LSTM network structure was chosen as it gave the best results for the four datasets described earlier. All other network parameters aside from the learning rate and activation functions, were left at the default setting. It is also important to note the backend to this Keras RNN is Theano, and uses the G++ compiler installation for performance reasons.

Refinement

Now that the model has been built, it is ready for training. However, this initial fitting and training will almost certainly not produce acceptable or accurate prediction results.

In short, the RNN model requires substantial parameter tuning before it can be used for a prediction. The reasons are due to the drastic differences in stock price behavior over time and across different stocks, as well as the complexity of the RNN model structure. As such, the RNN model is tuned (to the extent feasible) primarily using grid search cross validation (GridSearchCV).

Within the grid search tuning algorithm, only the following select parameters are tuned mainly due to performance time limitations. That is the enormous time it takes to perform the grid search using the RNN. The RNN must be trained over a significant number of training epochs (e.g., 500) upon each run of the grid search. This epoch training is very time consuming. Given this constraint, the following RNN parameters were chosen for tuning:

1. Number of Network Layers
2. Number of Neurons per layer
3. Dropout rate
4. Learning rate

The first parameter (network layers) makes up a fundamental part of the network architecture and should have a significant impact on the RNNs performance. That is, stacking layers within the RNN should allow it to better model the complexity of the time series sequence. Similarly, the second parameter (number of neurons) also has an

important performance impact as more neurons allows for more complex modeling of the stock predictions. Third, the dropout rate parameter is also vital as it helps to prevent overfitting on the training data. Fourth, the learning rate is also a key parameter in this tuning algorithm. A learning rate that is too large may cause the model to oscillate around the best fit, while too small a learning rate may require too many training epochs to reach convergence.

The grid search is run using the following parameter grid:

Parameter	**Choices**				
# of Network Layers	1	2	3	-	-
# of Neurons per layer	15	30	-	-	-
Dropout rate	10%	25%	-	-	-
Learning rate	0.02	0.008	0.004	0.001	0.0003
Batch size	8	-	-	-	-
# Days look back	9	-	-	-	-
Forecast horizon (days)	5	-	-	-	-

In addition, the grid search is implemented using 150 training epochs and with a time series validation split of two[20]. The batch size, look back and forecast horizon are held constant during the grid search, but are later stressed for sensitivity analysis.

For the following datasets the results of the grid search best fitting parameters are shown below:

ETF	Number of Network Layers	Number of Neurons per layer	Dropout rate	Learning rate
OIL	1	15	10%	.004
VXX	1	30	10%	.001

The grid search is run again using the same parameter choices but using a one day forecast horizon. The grid search best fitting parameters for the ***one*** day horizon are:

ETF	Number of Network Layers	Number of Neurons per layer	Dropout rate	Learning rate
SPY	1	15	10%	0.0003
QQQ	1	30	10%	0.001

One noteworthy choice made by the grid search is the number of layers is consistently one. This is despite the apparent power of stacking multiple layers within an RNN model. This may mean a larger (or different) grid space is needed to obtain more effective performance.

After the RNN model has been fit and tuned, it is used to predict the chosen ETF prices over the specified time period.

Model Evaluation and Validation

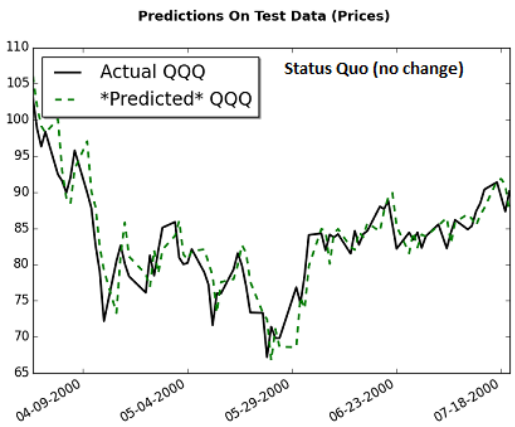
Since the RNN model is fairly complex and contains numerous parameters and settings it is important to understand the sensitivity to these configurations. To illustrate the sensitivity, the batch size, number of layers, neurons, look back window and learning rate are stressed to show the impact on the forecasts. In this example, the ETF QQQ is forecasted using the following configuration:

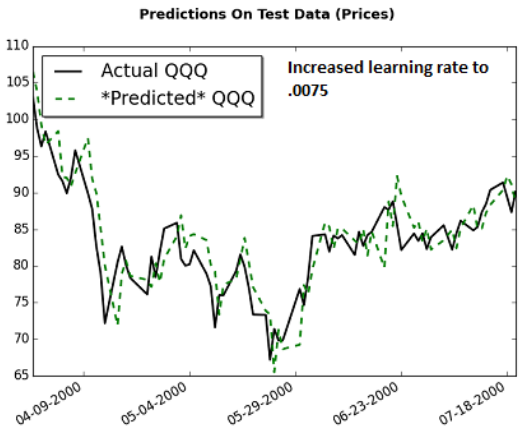
Parameter	**Value**
# of Network Layers	1
# of Neurons per layer	15
Dropout rate	25%
Learning rate	0.005
Batch size	8
# Days look back	9
Forecast horizon (days)	5
# Training epochs	350

The following table shows the sensitivity to parameter changes:

Parameter Change	Accuracy Ratio on QQQ Test Data Using RNN Model	% change in Accuracy Ratio	Accuracy Ratio on Test Data Using Benchmark
None (Status Quo)	0.1348	0%	0.256
Batch size reduced to 4	0.1483	-10%	0.256
Batch size increased to 12	0.1584	-18%	0.256
Neurons per layer increased to 30	0.1519	-13%	0.256
Layers increased to 2	0.1368	-1%	0.256
Increase learning rate to .0075	0.173	-28%	0.256
Increase lookback to 12 days	0.1472	-9%	0.256
Decrease # epochs to 250	0.1545	-15%	0.256

The above table shows the RNN model's sensitivity is greatest to the change in the learning rate, followed by the batch size, epochs, neurons size, and look back window. The following plots of the QQQ price forecast show the forecast using the status quo (left) and with the increased learning rate (at .0075). The right plot (increased learning rate) starts to deviate further from the actual price in the later months.





In the status quo forecast, the average absolute difference between the predicted and actual adjusted closing price is 2.68, while the minimum difference is .019 and maximum is 8.28. As shown above, the model is sensitive to parameter and configuration changes. This is further exacerbated by the fact that the stock price dataset is dynamic and changes quickly over time with respect to volatility and trend, among other factors.

Justification

The RNN model is ultimately evaluated on how well it can predict future stock prices against the benchmark. However, as previously discussed, the model is actually predicting one day changes in the log of the adjusted closing price, due to the stationarity issues. As such, these initial model outputs must first be converted into actual future stock prices to assess performance.

This output conversion is done by adding the model's one day change to the corresponding log of the adjusted closing price. Next, this "updated" log adjusted closing price is converted into a normal closing price by taking the exponential as:

Forecasted Adjusted Closing Price =exp (LN Updated Adjusted Closing price)

At this point the forecasted adjusted closing price can be compared to the actual price via the Accuracy Ratio:

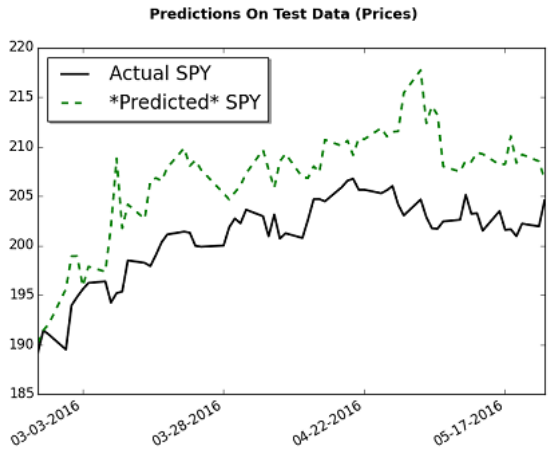
$\sum_{t}^{n} \log(\mathrm{Pt} / \mathrm{At})\text{^}2$, where Pt is the predicted price and At is the actual price.

The **lower the** Accuracy Ratio, the better the predictive performance as it measures the differences between actual and predicted values.

The RNN is first run for a one day forecast using a nine day lookback window on the following two datasets. It is run ***using the grid search best fit parameters***, and 500 training epochs.

ETF	Benchmark Accuracy Ratio	RNN Model Accuracy Ratio	# of data points tested
SPY	0.0076	0.065	64
QQQ	0.275	0.437	88

Below is a plot of the SPY forecast for a one day horizon:

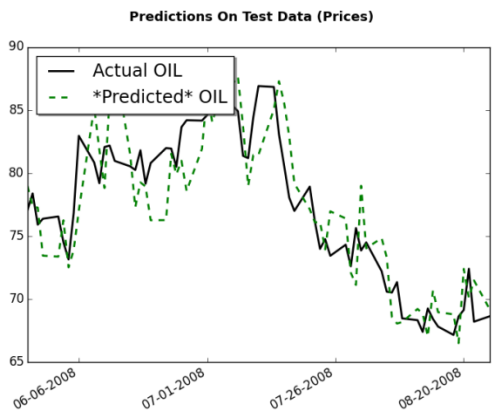


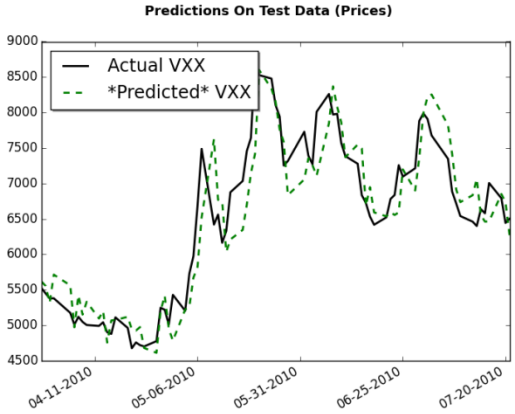
As the Accuracy Ratio (.065) and above chart indicate, the RNN model performs very poorly even on the one day ahead forecast using the parameters from the grid search. This may point to a need for a wider grid search space, or further redesign of the model.

The RNN is run again for a ***five*** day forecast using a nine day lookback window on the remaining two datasets. It is run, ***using the grid search best fit parameters***, and 500 training epochs.

ETF	Benchmark Accuracy Ratio	RNN Model Accuracy Ratio	# of data points tested
VXX	.399	0.311	80
OIL	.0868	.0951	64

Here the VXX forecast narrowly beats the benchmark with an Accuracy Ratio slightly lower than .399 at .311. However the OIL forecast does not beat the benchmark as the Accuracy Ratio is slightly higher (.0951 vs .0868). Below are the plots of the VXX and OIL five day price forecasts.





Both the OIL and VXX plots show significant deviations between predicted and actual prices over the testing data time period. Possible reasons for this poor predictive performance include a narrow grid search space or inadequate model architecture.

Out of the four datasets, only one prediction beat the benchmark (VXX) and did so by a very narrow margin. The other 3 predictions performed poorly and did not beat the benchmark. Unfortunately, the current RNN model is not effective enough or robust enough to solve the ultimate problem of stock price prediction. Despite the well-known power of LSTM RNN models, this particular implementation simply cannot consistently beat the benchmark or perform adequately.

Reflection

One of the biggest challenges of implementing the RNN model is the lack of documentation and instructions. While the Keras[14] website provides pure API documentation, it does ***not*** provide many useful examples and gives little guidance on real world implementations or actual problem solving. As such, many external websites[19] were consulted for guidance on how to build an RNN model to make time series predictions.

Other challenges include installation and computational performance. Again, the documentation on these areas is very poor at the time of writing. More specifically, since the RNN is so computationally intensive, it practically requires installation of the G++ compiler to run in a reasonable time. Installing the G++ compiler is very difficult and not well documented. This requires lots of external web searching and trial and error.

The last challenge is fitting and tuning the model to the data. Many different network architectures (e.g., neuron size, layer size) were tested out to understand how the RNN behaves as these settings change.

Improvement

The RNN model could be improved in several ways but perhaps the most significant improvement is in the construction and implementation of the RNN model. Although using an LSTM RNN for predicting time series is widely accepted, the specifics of the model architecture in terms of layers, neurons, activation functions, learning rates schedules and more, could be further refined and improved. Further, the training algorithm could also be much further tuned in terms of the ideal batch size, number of training epochs, and more.

Additionally, other technical features may have more predictive ability. These may include oscillators, momentum indicators, trend lines, support and resistance points, etc.... Other features may even include different asset classes such as interest rates and commodity prices which may lead the price of the ETF being forecasted.

In addition to the model and features, the benchmark chosen could have been more sophisticated and potentially even harder to surpass.

However, regardless of the benchmark or technical features chosen, it appears the biggest void to fill is in the area of documentation. There simply is not yet enough available guidance on how to build and design RNN models to solve this type of time series prediction problem.

References

1. "Stock Market Trend Prediction Using Support Vector Machines", I. P. Marković, M. B. Stojanović, J. Z. Stanković, M. B. Božić, *http://casopisi.junis.ni.ac.rs/index.php/FUAutContRob/issue/view/378*
2. "Machine Learning in Stock Price Trend Forecasting",Yuqing Dai, Yuning Zhang, http://cs229.stanford.edu/proj2013/DaiZhang-MachineLearningInStockPriceTrendForecasting.pdf
3. "An Empirical Comparison of Machine Learning Models For Time Series Forecasting", Nesreen K. Ahmed, Amir F. Atiya, Neamat El Gayar, Hisham El-Shishiny, Econometric Reviews, 29(5–6):594–621, 2010
4. https://www.arpm.co/symmys-articles/4%20_%20Attilio%20Meucci%20-%20Risk%20and%20Asset%20Allocation%20-%20Quest%20for%20Invariance.pdf
5. "A Better Measure of Relative Prediction Accuracy for Model Selection and Model Estimation", Chris Tofallis, https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2635088
6. http://finance.yahoo.com
7. https://en.wikipedia.org/wiki/Technical_analysis
8. https://pandas-datareader.readthedocs.io/en/latest/
9. https://en.wikipedia.org/wiki/Forecasting#Na.C3.AFve_approach
10. https://en.wikipedia.org/wiki/Forecasting#Drift_method
11. https://www.otexts.org/fpp/2/3
12. http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
13. https://en.wikipedia.org/wiki/Recurrent_neural_network#Long_short-term_memory
14. https://keras.io/
15. http://www.investopedia.com/articles/trading/03/031203.asp
16. http://www.blastchart.com/Community/IndicatorGuide/Indicators/EaseOfMovement.aspx
17. http://www.johnwittenauer.net/a-simple-time-series-analysis-of-the-sp-500-index/
18. http://www.brameshtechanalysis.com/2015/05/understanding-on-balance-volume-indicator/
19. http://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/
20. http://scikitlearn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html#sklearn.model_selection.TimeSeriesSplit
21. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html
22. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html
23. https://pypi.python.org/pypi/Theano