Capstone Project: Predicting the adjusted close of Nasdaq

Hassan Sirelkhatim

August 15, 2017

Abstract

1 Definition

1.1 Project Overview

The stock market prediction problem is the problem of determining the future value of a certain stock. There are different ways of doing this, the most famous of which are fundamental analysis and technical analysis. Fundamental analysis is the method of assessing the value of the underlying company/companies who are behind the stock to determine its future value. Technical analysis concerns itself with charts and the patters that are observable in them. Another method which we will be concerning ourselves with in this project is the maching learning approach, which uses learning algorithms to predict the future value of a stock.

1.2 Problem Statement

In this project we will experiment with multiple models to observe the effect of using different strategies to predict the adjusted close. In the first half we will be using standard machine learning (ML) techniques (Random Forest, SVM, Linear regression) to predict the adjusted close based on the features Open, High, Low, and Volume of the stock. In the second half of the project we will test out three different deep learning architectures using Keras to benchmark with the standard

ML techniques. We will generally use Recurrent neural networks (RNNs) for the deep learning part. We will end the paper by concluding that RNNs/deep learning are generally a more powerful method when it comes to predicting stocks. RNN is a neural network that uses loops to encode sequences, which make these types of networks ideal for modelling time series. LSTM stands for Long Short Term Memory network and is a type of recurrent neural network that is good at encoding long term dependencies. Say for example I am writing a sentence in the second page of a report that is referring to a sentence in the first page. These networks aim to model these kind of dependencies. In this project I have downloaded data containing Nasdaq prices from Yahoo Finance for the years 1984-2017.

1.3 Metrics

In this project we will be using the root mean squared error (RMSE) to assess our models. The RMSE is defined as follows: $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$. Here y_i are the actual target values and $\hat{y}_i$ are the predicted values of our models. It indicates the absolute fit of the model to the data, and is commonly used when the type of problem we are dealing with is a prediction of continuous values (i.e. stock price) and we want to punish large deviations heavily, which makes it ideal for the purposes of our project.

2 Analysis

2.1 Exploratory Data Analysis

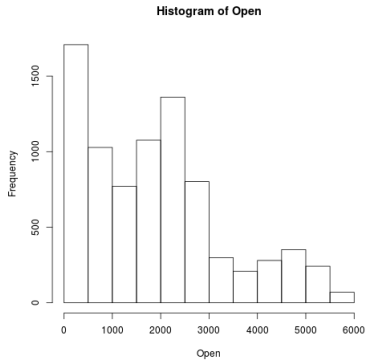
```
         Date         Open         High          Low        Close      Volume  \
0  2017-04-28  6072.870117  6074.040039  6040.709961  6047.609863  1995160000
1  2017-04-27  6038.470215  6050.700195  6031.589844  6048.939941  1876700000
2  2017-04-26  6028.120117  6040.890137  6021.720215  6025.229980  1894210000
3  2017-04-25  6004.160156  6036.020020  6002.649902  6025.490234  1895430000
4  2017-04-24  5979.959961  5989.919922  5970.250000  5983.819824  1836570000

     Adj Close
0  6047.609863
1  6048.939941
2  6025.229980
3  6025.490234
```

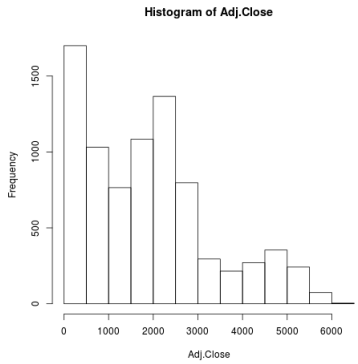
```
4  5983.819824

              Open         High          Low        Close        Volume  \
count  8205.000000  8205.000000  8205.000000  8205.000000  8.205000e+03
mean   1922.367570  1935.431815  1907.102335  1922.037527  1.164997e+09
std    1396.845658  1405.270890  1386.601750  1396.514087  8.571358e+08
min     238.000000   238.399994   237.699997   238.100006  3.884000e+07
25%     639.239990   644.919983   638.330017   642.599976  2.201500e+08
50%    1824.069946  1840.829956  1804.650024  1825.530029  1.384620e+09
75%    2595.449951  2612.699951  2568.439941  2593.379883  1.886200e+09
max    6072.870117  6074.040039  6040.709961  6048.939941  4.553600e+09

         Adj Close
count  8205.000000
mean   1922.037527
std    1396.514087
min     238.100006
25%     642.599976
50%    1825.530029
75%    2593.379883
max    6048.939941
```

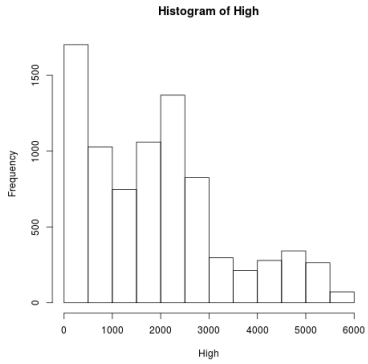
In our dataset we extracted the variables date, open, high, low, close, volume, and adjusted close. We find that open, high, low, adjusted close, and volume are all positively skewed (high frequency of relatively small magnitude numbers). Most of the outliers in Open, High, Low, and adjusted close lie in the range $4500-6000$. For volume the outliers lie in the range $3,500,000,000-4,500,000,000$. The volume has a large skew (difference between mean and median) of $219,623,000$. All the variables are numerical with the exception of date, which is a date variable. There are no missing values in our dataset, and other than a few outliers there are no abnormalities.

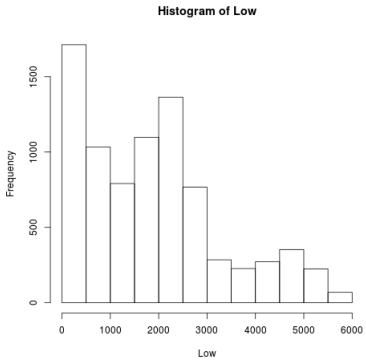
3 Exploratory Visualisation

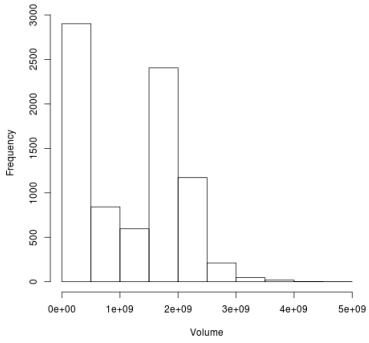
Here we see histograms created of the relevant variables to our problem statement. The significant thing to notice is that they are positively skewed. However, the volume variable does seem to have a bimodal shape. Open, close, high, low and adjusted close all seem to have fat tails, with the exception of volume.



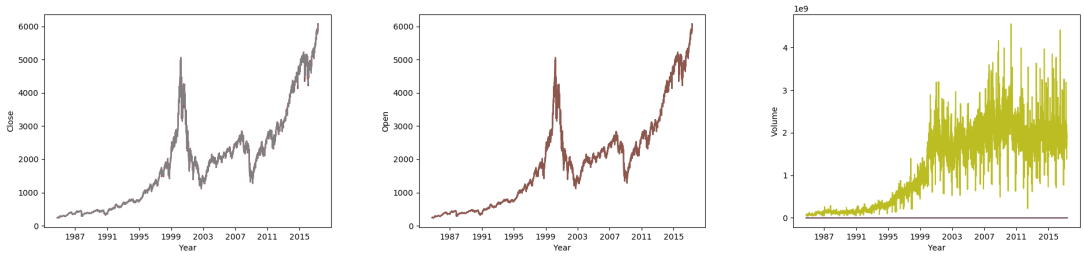




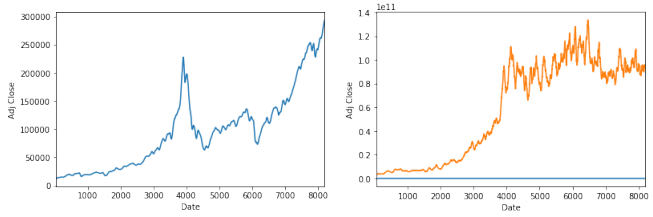




It would be good to visualise the yearly adjusted close to start with.



There price seems to be increasing over the years with occasional dips (2000-2003). The volume is also increasing with high variability.



Here we see a smoother graph of Volume and the closing price than the graph above by appling a rolling mean with a window size of 50.

3.1 Benchmark

The benchmark for this project is Jakob Aungiers/Siraj's LSTM model [4][5]. I have modified the parameters a bit, by changing the keep probability of the dropout layers and the activation unit as I thought it might make the model better at predicting the price. I will define the model again along with the parameters for convenience.

1. LSTM network (input dimensions: 1, output dimensions: 50, return sequences = True)
2. Dropout (keep probability: 0.5)
3. LSTM (100 units)
4. Dropout (keep probability: 0.5)
5. Relu activation unit
6. Fully connected dense layer (output dim: 1)

4 Methodology

4.0.1 Linear Regression

Linear regression is probably the simplest method that will be used in this report to predict the nasdaq prices. Mathematically the linear regression can be expressed as a finite weighted sum of independent variables used to predict a regressor variable (dependent variable). Formally we say $y = \alpha x_1 + \beta_1 x_2 + \ldots + \beta_2 x_n$ Here the x_i represents the open, High, Low, and volume.

4.0.2 Random Forest

Random forests are an ensemble learning method which aggregates the result of many decision trees to get a better decision tree. It randomly selects a sample with replacement of the training data and at each candidate split selects a random subset of features and fits a tree to it. It then predicts the unseen sample x_{test} by averaging all the predictions from the individual trees using the formula $\frac{1}{B}\sum_{b=1}^{B} f_b(x_{test})$.

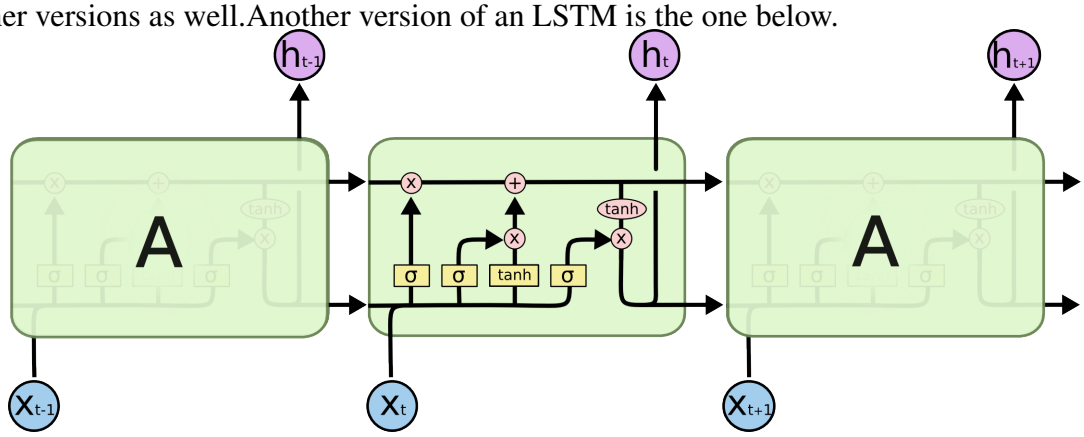
4.0.3 SVM

Support Vector Machines are a classification/regression algorithm that attempts to find the optimal margin of a line, to find the most optimal classification line/regression line. If the data is not linear one can use the kernel trick to transform the data into a linearly separable dataset.

4.0.4 LSTM

In this section we will discuss the theory behind the LSTM in more depth. A LSTM is an extended version of an RNN that seeks to model long term dependencies by having a long-term memory. In the first step of a normal LSTM, we would decide what information we're going to forget from the cell state. We use a sigmoid layer called the forget gate layer. It looks at h_{t-1} and x_t and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 would represent keeping the information completely, while a 0 would be to forget it completeley. This is modelled by the equation $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f$. The next step is to decide what new information is going to be encoded in the cell state. First a sigmoid layer called the "input gate layer" decides which values will be updated. Then a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that might be added to the state. This is modelled by the equations $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$ and then the

equation $\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_C)$. Now we update the old cell state, C_{t-1} into the new cell state C_t. We multiply by f_t, forgetting the things we decided. Then we add $i_t * \tilde{C}_t$. This gives us the equation $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$. Then we run a sigmoid layer which decides what we are going to output. Then we put the cell state through tanh, getting values between -1 and 1 and multiply by the output of the sigmoid gate. so we get $o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$ and $h_t = o_t * \tanh(C_t)$. Below we have a picture a LSTM. The reason I say normal is because there are other versions as well.Another version of an LSTM is the one below.



4.1 Preprocesseing

4.1.1 Classical ML algorithms

To run these three algoritms, I first shifted the adjusted close by 7 days ahead. So that the first row of open, high, low, volume corresponds with the adjusted close 7 days later. Furthermore, I standardised the data with the MinMaxScaler since the volume had a much higher range than the open, high, and low. To split the data into training, testing, and validation we use the train test split function from sklearn. We first split the data into training and testing with 20% for testing. We further split the training into validation and training set, where we use 20% for validation.

4.1.2 For LSTM

For preprocessing we started by normalising the prices using the equation:

$n_i = \left(\frac{p_i}{p_0}\right) - 1$

and to denormalise we use the equation:

$p_i = p_0(n_i + 1)$

where: n = normalised list [window] of price changes

p = raw list[window] of adjusted daily return prices

Here the window is the window size we use, which means instead of taking each price point we take average over a window to avoid training over the noise.

4.2 Model Architecture for LSTM

4.2.1 Model 1

I have used two modelsThe model I used was taken from Siraj's video on stock prediction ($https://www.youtube.com/watch?v=ftMq5ps503w$). The model is defined as follows:

1. LSTM network (input dimensions: 1, output dimensions: 50, return sequences = True)
2. Dropout (keep probability: 0.5)
3. LSTM (100 units)
4. Dropout (keep probability: 0.5)
5. Relu activation unit
6. Fully connected dense layer (output dim: 1)

4.2.2 Model 2

1. Fully connected layer (output dim: 1, input shape: (50,1)
2. LSTM (100 units)
3. Dropout (keep probability of 0.5)
4. fully connected layer (output dim: 1)

4.2.3 Model 3

1. Fully connected layer (output dim:1 , input shape: (50,1))
2. LSTM (100 units)
3. Dropout (keep probability: 0.5)
4. Fully connected layer (output dim: 50)
5. Dropout (keep probability 0.4)
6. Fully connected layer (output dim: 25)
7. Dropout (keep probability 0.3)
8. Fully connected layer (output dim: 10)
9. Dropout (keep probability 0.2)
10. Fully connected layer (output dim: 5)
11. Fully connected layer (ouptut dim: 1)

4.3 Implementation

4.3.1 Data

The data that has been used for this project was downloaded from Yahoo finance for the days 11/10/1984 to 28/04/2017

4.3.2 Framework and software

This project was coded in python 3.6.1 using keras and the python code/package lstm was taken from this blog on time series prediction http://www.jakob-aungiers.com/articles/a/LSTM-Neural-Network-for-Time-Series-Prediction. For data exploration R version 3.2.3 was used.

4.3.3 Process

To implement the algorithms above in part 1, we made use of scikit. To split the data we used the train test split function from sklearn, and then it was straightforward to implement the algorithms. To assess it we used the mean squared error function from sklearn.metrics. Overall there was no complications when implementing the algorithms. For the second part we used keras to implement the architectures above. to preprocess the data and split into training and testing we made use of lstm.py which was gotten from a blog. The complications arised when trying to experiment with different architectures. This was mostly resolved by experimenting profusely. The issue arised when trying to create compatible dimensions between the different layers.

4.4 Refinement

4.4.1 Part 1: Classical ML

Initially after we applied linear regression, random forests, and SVM, we got the results to be (from the RMSE): 6931.13669123,1318.74558589, 1122755.97522 respectively on the training data. Therefore we chose the random forest algorithm as our model and fine tuned it using grid search. The parameters that we fine tuned were maximum depth (the maxium recursion depth of the trees) and the number of estimators (number of trees in the forest). Scikit has a module for grid search which we applied to get the optimal parameters. We were able to lower the RMSE to 986.845124714 on the validation set. The final result of the random forest on the test set was 8849.18742758.

4.4.2 Part2: Deep learning architectures

For the deep learning architectures our benchmark architecture got a rmse of 0.00335285039019 on the test data. The other deep learning architectures that I came up with were largely due to experimentation and the idea that larger architectures were generally better at modelling complex datasets. We managed to get the rmse down to 0.0038389390613, 0.000770337886014 for model 1 and model 2 respectively.

5 Results

5.1 Model Evaluation and Validation

The final model that we got from part 1 (tuned random forest) is not a reasonable model for this time series problem. The training/validation error was relatively high and furthermore the training error was even higher. This shows that this model isn't really stable and not to be trusted with this type of data. However, the final model we got from the deep learning architectures seem to be pretty reasonable with a very low error on the testing data (unseen data), which indicates that it is a stable algorithm and doesn't change much with small perturbations, which makes it ideal for modelling this type of complex data. We can also see from the graphs below that model 1 (benchmark model) is doing a better job in modelling the price fluctuations of the model than model 2 and model 3, which seem to exaggerate the upward movements.

Figure 1: Model 1

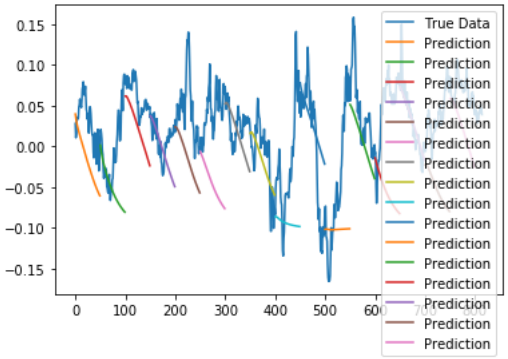


Figure 2: Model 2

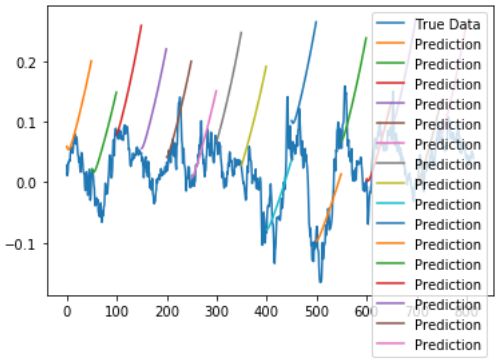
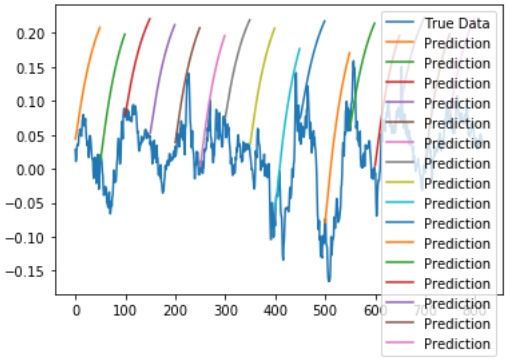


Figure 3: Model 3



5.2 Justification

For part 1, the benchmark model is clearly much better and so or model here as failed with an error of 2366492.94254 on the the testing data, which indicates that the model. As for model 3 architecture created above it does not give a much lower error than the benchmark model. The difference in errors is 0.000185887.

5.3 Conclusion

Using an LSTM network we are able to predict the adjusted close prices with a very low rms. In this project we attempted to predict the adjusted close of Nasdaq using classical ML algorithms as well as deep learning models. The project is split

into two parts: The first part tests ML algorithms and the second part deals with deep learning architectures used to predicted the adjusted close. Before training the models we first preprocess the data in different ways depending on whether we are using ML or deep learning architectures. After preprocessing the data we split the data into training and testing (also validation for the ML part). We end by concluding that deep learning architectures (specifically LSTM) are better at predicting when dealing with timeseries data. We test 3 different architectures, one which is a benchmark and two others which were derived through experimentation. The best architecture we get is from model 3 with a mean squared error (MSE) of 0.000525887298149, however the difference is not really significant enough since the difference is minute. In terms of the classical ML algorithms, they seem to be poorly suited for time series. The lowest MSE error we got was 2366492.94254, which is very far from our prediction accuracy with deep learning models. Model 3 was derived by experimenting with architectures. The basic idea is that the benchmark model did not have many layers and so I thought adding more layers, while still adding dropout layers, would add to the model's ability to deal with complex data. The results stated above, definitely agree with my intuition of deep learning architectures generally being better than classical ML algorithms at dealing with complex datasets. Furthermore, It does not however agree with my hypothesis that adding more layers to a deep learning network should give better results. Therefore this model should not be used as a general setting to solve these types of time series problems. However, it might not be a fair comparison since the deep learning model only had the previous adjusted close prices to predict from. Whereas the ML algorithms had to predict the adjusted close prices 1 days ahead using the features open, close, and volume, which could possible have introduced unnecesary noise, if they have no predictive value. For future investigation one might consider trying to predict the adjusted close using only the previous close prices. Also the random forest seems to be ill suited for these types of problems and more to overfit. One might consider using other ensemble methods or introducing more regularisation mechanisms to get better results

6 Sources

1. https://en.wikipedia.org/wiki/Random_forest
2. https://en.wikipedia.org/wiki/Support_vector_machine
3. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

4. http://www.jakob-aungiers.com/articles/a/LSTM-Neural-Network-for-Time-Series-
5. https://www.youtube.com/watch?v=ftMq5ps503w)
6. https://www.kaggle.com/wiki/RootMeanSquaredError
7. http://www.theanalysisfactor.com/assessing-the-fit-of-regression-models/
8. http://people.duke.edu/~rnau/compare.htm
9. https://en.wikipedia.org/wiki/Stock_market_prediction