Datascience Nanodegree

Capstone Project

Roald Brønstad
November 26, 2018

I. Definition

Project Overview

Predicting stock prices and market direction can surely be considered the holy grail within finance. Algorithmic trading is replacing humans and the finance sector is exploring how new technology can be utilized every day.

Traditionally the finance sector has relied on econometric tools such as linear regression to try and predict the market, today machine learning and deep learning networks are taking over the task.

The purpose of this project is to investigate the possibility of predicting tomorrows stock market direction based on historical data as a binary classification, up or down. Daily prices for the Norwegian stock index OSEBX is available online with variables such as open, high, low, closing price and volume. I have also expand the data-set with technical indicators such as momentum, relative strength index, moving average etc. The difference between closing price today and closing price tomorrow (prediction) will be either positive or negative, indicating direction.

The project will experiment with several machine learning algorithms as well as artificial neural network to analyze the possibility of predicting tomorrow direction

Problem Statement

A successful model will be able to predict whether the market will move up or down tomorrow better than random guess. It should output a 1 for predicting upwards movement and 0 for downward movement.

An important step before building this model will be to create the data-set. The data is in a time-series format and in order to use machine learning models in a useful way, technical indicators must be derived from the time series observations. The target label (at time t) will be defined as 1 if closing price for (t+1) - t is positive and 0 if its negative.

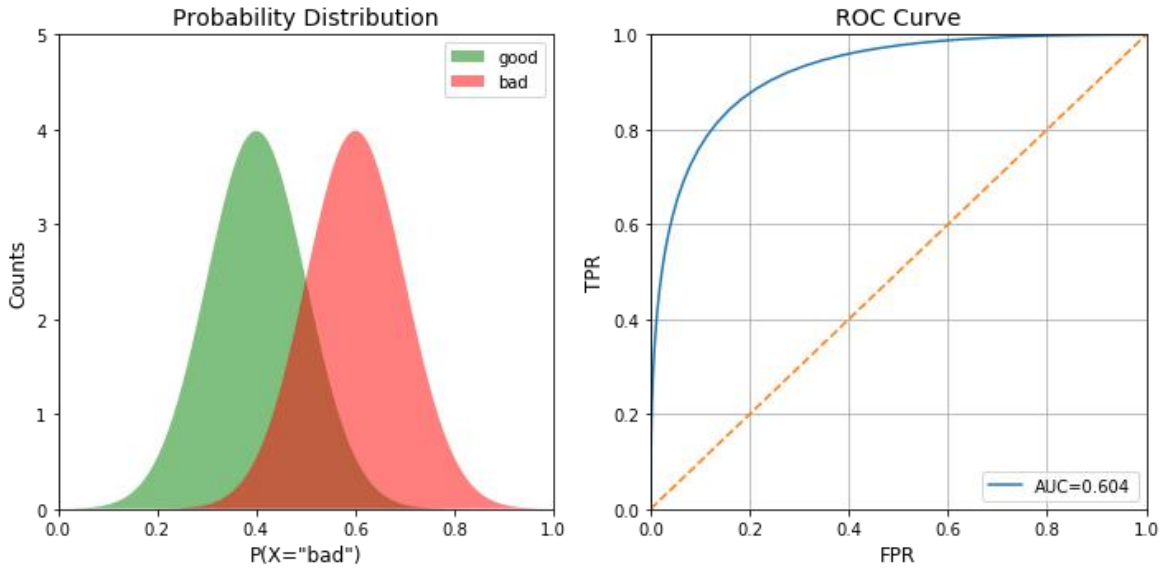
The data-set will be split into a training set and test set by a ratio of 70 - 30% and normalized to a range between 0 and 1.

I will train a simple logistic regression model to serve as a benchmark model to compare against more advanced and relevant machine learning models such as Random forest, Naive Bayes, Support vector machines and Adaboost. The models will be trained and hyper tuned by exhaustive grid-search and compared to each other and the benchmark model. I will also build a Recurrent neural network which has been developed especially to work well on sequence problems.

Metrics

I have used ROC curve and area under curve as an evaluation metric for this project.

A roc curve is derived from a confusion matrix of the test data (true/false positive and true/false negatives) A binary classifier will classify based on the distribution of probabilities. (see picture below) A threshold needs to be set unless the model can predict 100%. The ROC curve shows true positive rate (true positive / all positive) against a false positive rate (false positive / all negative) plotted for all discriminating threshold in the distributions. The diagonal line shows pure guessing between two classes. The ROC curve for a model with good predictive power will lie close to the upper left corner. Far away from the diagonal line.



II. Analysis

Data Exploration

The data-set I have chosen has been downloaded from Oslo stock exchange* and contains daily observations of high, low, closing price and volume from 2001. There are no missing values in the dataset which consisted of a total of 4388 rows and 4 features initially.

I have derived 14 extra features out of the original features. The complete list is:

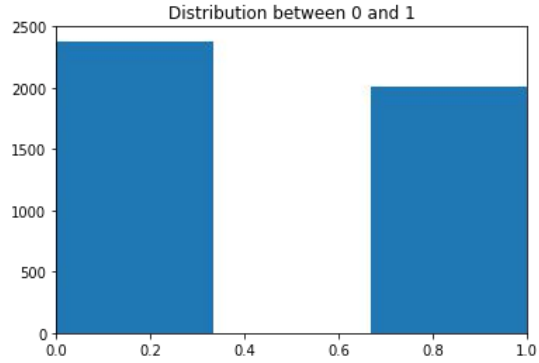
- Simple moving average (10 days)
- Simple moving average (30 days)
- Simple moving average (90 days)
- Exponential moving average (10 days)
- Exponential moving average (30 days)
- Exponential moving average (90 days)
- Stochastic Oscillator

*https://www.oslobors.no/ob_eng/markedsaktivitet/#/details/OSEBX.OSE/overview

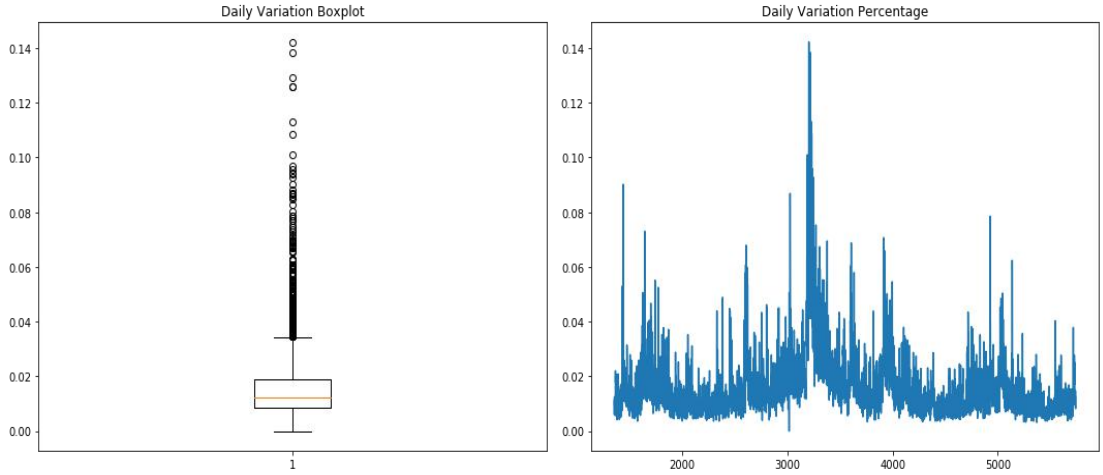
- Momentum (10 days)
- Momentum (5 days)
- Momentum (2 days)
- Standard deviation (from 10 day rolling mean)
- Daily variance
- Day of week
- Month
- Relative strength index*

The features derived from time, ie. Day of week and month was converted to dummy variables.

I have also created a target label with 1 for upwards movement next day or 0 for downward. The distribution between 1 and 0 in the target features looks like this:



Financial data is known to be very noisy which makes it very difficult to predict. I created a feature for daily variation in percentage in order to look at outliers/noisy data.

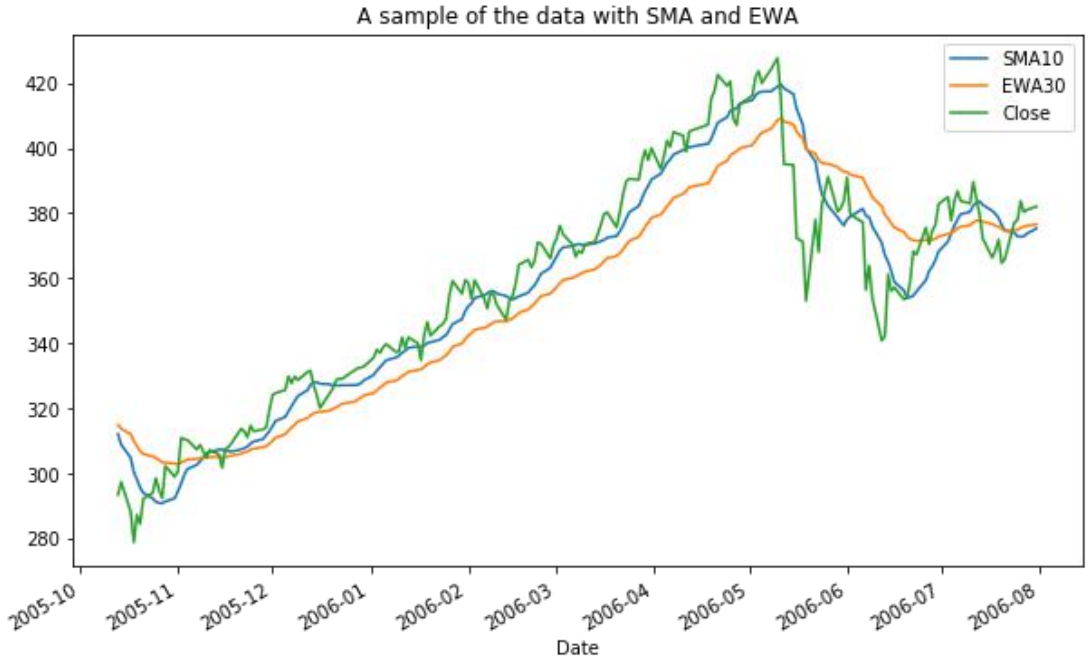


I tried removing data-rows with daily variation more than 5%, but it made no impact on the results. Since large daily variations are a part of the financial markets, I decided not to drop any extreme values.

*https://www.investopedia.com/terms/r/rsi.asp

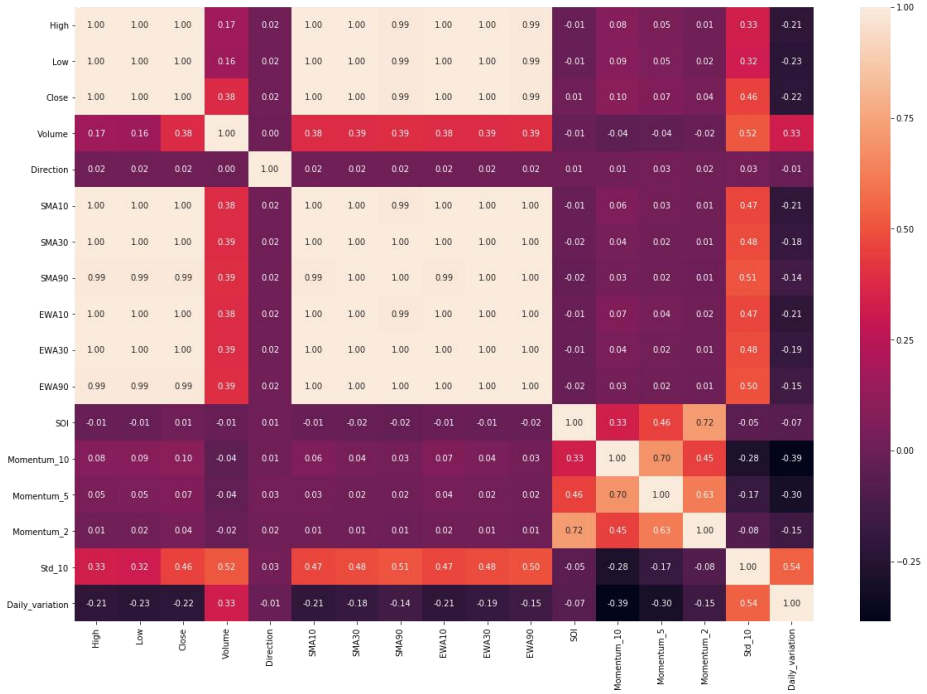
The size of my data-set (4388 rows) represent a concern due to its small size. I would have prefer to use data on a more detailed level such as hourly or even minute, but I was not able to get data on that level.

Exploratory Visualization



I made this visualization foremost as a sanity check to test the technical indicators I created in order to be sure they would appear as expected. I have plotted the closing price along with a Simple moving average and an exponential moving average. Features such as these two can hopefully contribute in predicting the future movement of the market.

I also made a heatmap in order to get some insight into what correlates with the target label. There are no huge correlations unfortunately, but the indicators seem to have a very small impact nonetheless.



Algorithms and Techniques

I experimented with several machine-learning models commonly used for binary classification for my project. The models I used was:

Support Vector machine:

An advanced algorithm for solving non-linear problems both within regression and classification. SVM use a technique called the kernel trick in order to take low dimensional input space and transform it to a higher dimensions space that in turn can transform a non separable problem into a separable one.

Random Forest Classifier:

This algorithm consists of an ensemble of decision trees. Decision trees can be compared to the game 21 questions, were each tree seeks to answer or vote on simple questions. A combination of hundreds or even thousands of these trees makes up a forest. The randomness comes from the fact that each tree is trained on a random subset of the data and a random subset of the features. This makes the algorithm more robust and less skewed by noisy data and outliers.

Naive Bayes:

Naive Bayes uses Bayes rule to calculate the probability of the features occurring in each class and return the most likely class. The algorithm assumes Independence between features and when this assumption holds, Naive Bayes is a fast and reliable algorithm that can be trained on little data.

AdaBoost:

AdaBoost is also based on decision trees. One decision tree is a very simple way to classify and AdaBoost combines many of these simple trees in order to make one strong algorithm. The only requirement is that the simple trees must provide slightly more precision than pure guess. Each tree is called a weak learner. AdaBoost will increase the weights on all points who were not correctly classified in each iteration thus the next weak learner will focus on misclassified points from the last iteration. At last AdaBoost will create one strong classifier out of all the weak learners and weigh them based on their error rate.

Recurrent neural network

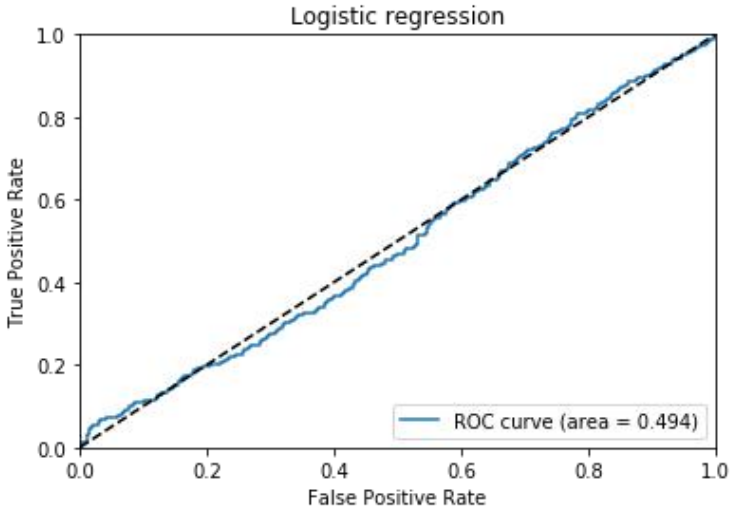
This form of artificial neural networks differs from the more simple convolutional network because it does not assume that the inputs and outputs are independent of each other. RNNs can be used for sequence prediction such as next word in a sentence or in this case sequence patterns in stock data. RNN uses its output from each node as an input for the next, thus creating a form of memory

Gridsearch Randomizedsearch

All the machinelearning models have been tuned with grid search (or randomized search to save time). This is a technique for automating experimentation with multiple different hypertunings in order to find the best settings.

Benchmark

I created a simple logistic regression model to serve as a benchmark for the more advanced machine learning models. All the machine learning models were compared to the benchmark model by ROC-curves. A good performing model needs to have an area under the curve significantly better than random guess --> 0.5. As we can see from the picture, the log model does not do a much better job than random guess.



III. Methodology

Data Preprocessing

Most of the features I have used have been derived out of a few initial features. Such as moving averages, momentum, variation day of week etc.

I chose to drop observations before 2001 as they did not contain data on volume. There were no further missing values in my dataset except for the first few lines of hte moving averages that could not be calculated before a certain amount of ebservations had been done. They were also dropped.

I analyzed outliers visually as described in the data exploration part. I experimented with creating a dataset excluding rows with more than 5% daily variation to check if this would increase accuracy. The result was negative. The financial markets are known to be noisy and containing extreme values, but that's the reality of that kind of data and I chose not to exclude any extreme values in the end.

All the features was normalized between 0 and 1 as a preprocess to increase accuracy and to be able to use Support vector machine who requires normalization. I used the MinMaxScaler in Sklearn to achieve this.

Implementation

After creating features and a target label based on closing price next day, The data was split into a training set and a test set with a split of 70 -30%. I used Sklearn to implement machine learning algorithms and Pytorch to build the RNN network. The machine learning models was tuned with GridSearch or RandomizedSearch with ROC-curve as scorer.

All the models was then evaluated by plotting their ROC-curves.

Building the RNN network was the biggest challenge since the amount of online resources on implementing rnn models is quite limited, especially with Pytorch. The data was fed into the model using pytorchs dataloader with batches of 30 observations at a time. I made a function to train the model with a periodical print out of accuracy on the given batch and an accuracy function to test on the testset when the training was done. I also implemented functions for saving and loading the model.

Refinement

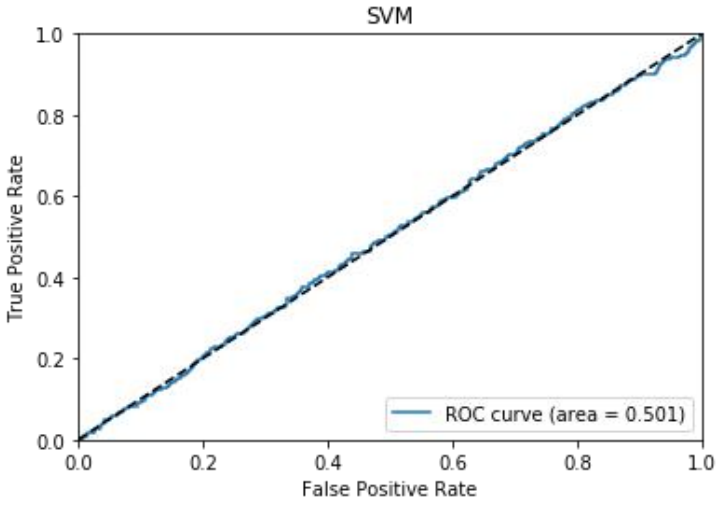
After the first training runs, results were not great. I started making more features (technical indicators) and experimented with standardizing the data instead of normalizing and to remove outliers. I also trained the RNN model for more epochs (20) in order to try and increase the accuracy. I also expanded the parameters in the GridSearch to test more hypertunings.

IV. Results

Model Evaluation and Validation

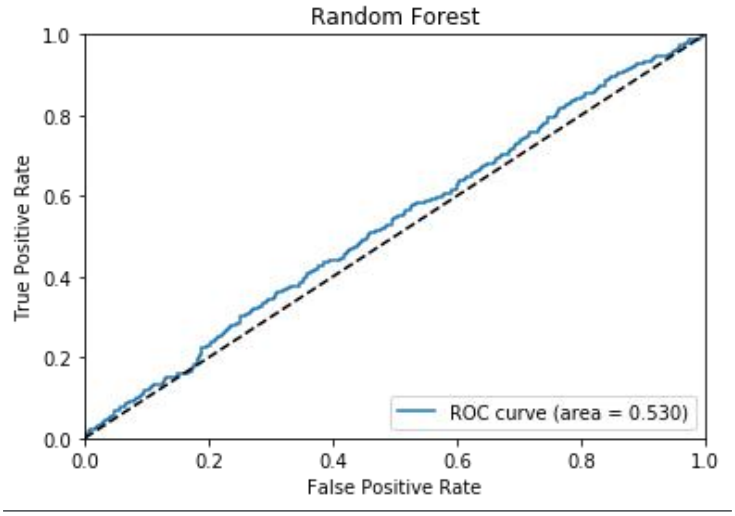
Visualizations and discussions of the performance of each of the models I developed:

Support vector machine:



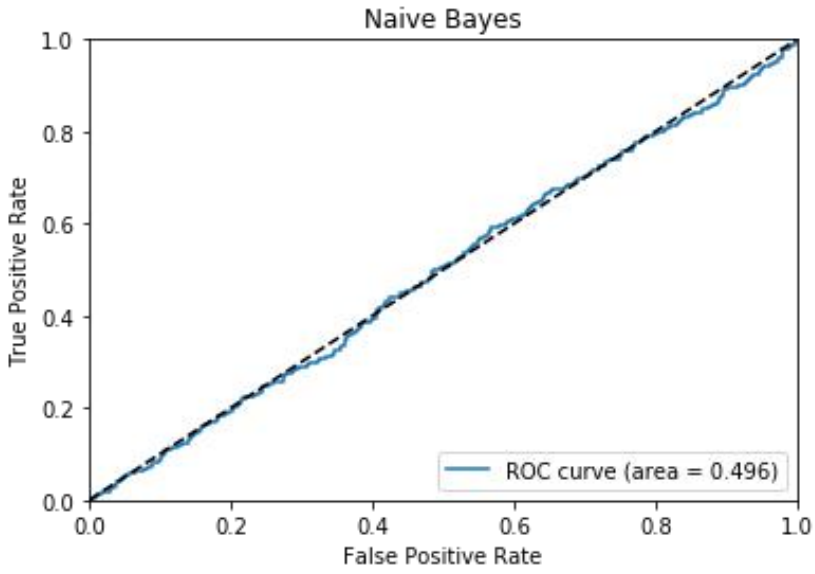
The result for SVM is pretty negative. No good increase in accuracy over random guess which is represented by the diagonal line.

Random forest classifier



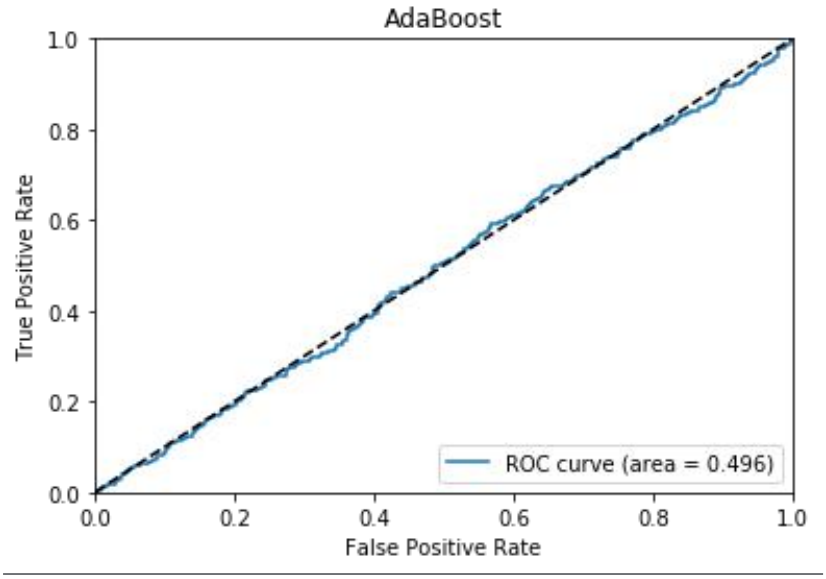
Random forest gave the best visual result out of the machine learning models, but its only marginally better than guessing. Due to the fact that the test set is quite small (only 1300) observations) I conclude that the performance is to marginal to be considered significant.

Naive Bayes:



The Naive Bayes model offers no better performance than guessing.

AdaBoost



Adaboost neither

RNN

The RNN network gave an accuracy 0f 55%. This is exactly the result a model that purely guessed down as a prediction would have achieved as the number of days down vs. Up was 55%.

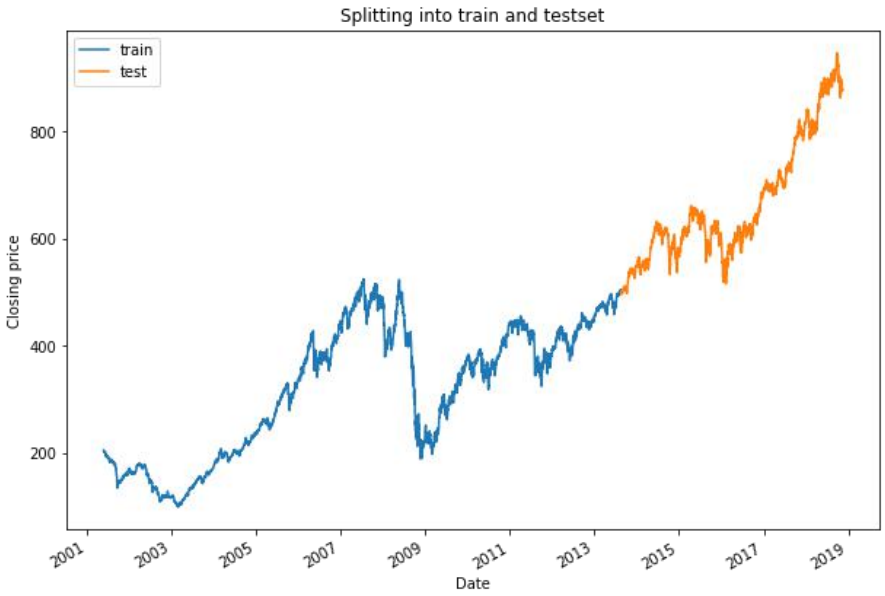
Justification

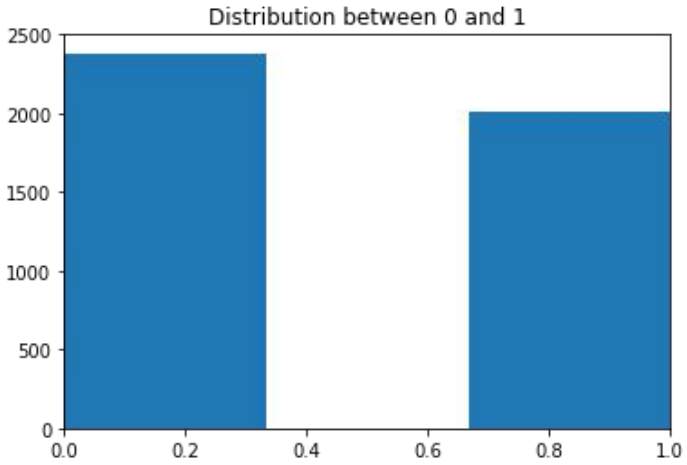
Unfortunately, none of the models I developed was able to predict the stock market direction efficiently. The benchmark and the machine learning models as well as the neural network cant do any better than random guessing.

V. Conclusion

Free-Form Visualization

The following visualization represents the closing price for the whole data-set plotted against time. It also shows how i split the data between training and test. In my proposal I argued that there must be more days with positive direction than downwad direction as the trend is clearly positive, but thats not actually the case. The next visualization shows that the distribution between positive and negative days are actually 55% down and 45% up. However the stock market has bigger moves on the good days than it looses on the bad days.





Reflection

This has been an interesting project involving a time series problem and the opportunity to implement a wide range of machine learning models as well as neural network. Engineering features was fun and I had to research which technical indicators are perceived to yield predictive power. When the data-set was completed I implemented several machine learning algorithms and tuned them with GridSearch.

The challenge of predicting financial markets turned out to be to big this time and I was not able to create a good model. This is not to surprising though as the financial markets are quite stochastic in nature and many argues that they are simply a random walk. I did learn a lot during the process, about time series and recurrent neural networks

It was quite difficult to dive into the realm of recurrent neural network and I spend a substantial amount of time to build a model and train it to predict something. For a long time it would only predict the label 0 (down movement).

Improvement

I believe one the of biggest issues with my implementation is the limited data-set. With daily observations I only had 4388 rows of data and this is too little, especially for neural networks. I would have been interesting to do this project on hourly or minute data, but I could not find such a data-set for free. However it could be that the stock market simply cannot be predicted based on historic data.

It would also be interesting to implement a LSTM - neural network on the data. I made an attempt to, but I was not able to implement it correctly. The network trained, but the loss only repeated itself in a pattern with no notable decrease over time.