Ruben Cubo
September, 2017

Learning the Stock Market in Depth

Definition

Background

The Stock Market has been one of the main drivers for creating and multiplying wealth historically, together with other investments such as real estate and entrepreneurship. This is reflected in historical market performances: investing in the S&P 500 has returned 7% on average after inflation, which is enough to double real value of investments in roughly 10 years according to the rule of 72.

However, there have been people that are capable of beating the market, i.e. indices such as the S&P 500, e.g. Warren Buffett. Technical and fundamental approaches have been used, although this work will be focused on the former. A myriad of indicators and methods exist, although in this work the focus will be on deep learning, and in particular, using recurrent neural networks (rNN).

Problem Statement

The main aim of this work is to estimate how a stock value will fluctuate with time by using rNNs. In particular, we want to focus on one step prediction, i.e. predicting what the exact value of the stock the next day. In particular, we are interested in the Close value, i.e. the value the stock will have at the end of the day. Hence, it will be a regression problem, where the solution would be the estimated value of the stock. There are many regression algorithms in the literature, but it is unclear how the stock prices will depend on the rest of the available data. Hence, a model capable of taking into consideration non-linear interactions between features such as a neuron network would be suitable.

With the predictions, one can then give orders to buy or sell accordingly, optimizing returns which will hopefully beat the market, although portfolio optimization is well beyond the scope of this work.

The main issue when trying to predict market prices is high volatility: prices can vary by a large amount during a day and in between days for unknown or unrelated reasons. However, the author believes that these volatilities can be predicted in a model to some extent, although they can be very complex.

Evaluation Metrics

Several evaluation metrics can be considered in these kinds of problems. Since we are dealing with a regression problem, some measurement of the deviation from the real value would be suitable.

First, for the value estimation, the metric would be the Mean Square Error (MSE):

$$MSE = \frac{1}{N}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

Where $\hat{Y}_i$ would be the estimated value of the stock price and Y_i would be the real value. N would be the total number of stocks and time instants considered. This was chosen since we are interested on the deviation with respect to the real value of the stock.

Another measurement would be the simple return:

$$r_s = \frac{P(t+1) - P(t)}{P(t)}$$

Where P(t) and P(t+1) would be the stock prices or moving averages at time t and t+1. Time here depends on the sampling: it can be one day, one week or even one year. When considering whether we can beat the market or not, we will want to buy if the predicted return is positive and sell when it is negative.

Analysis

Data exploration

In order to do the analysis, stock market data is needed. For this work, Quandl's freely available stock data will be used, that will include historical data of open, high, low, close and trading volumes for each day. Both nominal and adjusted were taken into consideration, with the latter being a more accurate view of the real value of the stock, since some factors such as dividends, splits and new offerings are taken into consideration.

It should be stressed that we want to predict the adjusted close value, that is, the value the stock will take at the end of the day after correction. Hence, we want to solve a regression problem, with adjusted close as labels and with the rest of the dataset as features.

Luckily, the data obtained from Quandl did not have any missing values. Since the stocks are a time sequence and it is assumed to be accurate, no outlier removal was considered.

For this project, the whole stock data available from Google was considered. The range in this case spans around 13 years from 2004 up to the current date, although it is possible to take other companies and/or subsets, but it is beyond the scope of this project.

In total, there are 3287 samples and 10 different features. An example of the 5 first rows of the data is depicted in Figure 1.

Date	Open	High	Low	Close	Volume	Ex-Dividend	Split Ratio	Adj. Open	Adj. High	Adj. Low	Adj. Close	Adj. Volume
2004-08-19	100.01	104.06	95.96	100.335	44659000.0	0.0	1.0	50.159839	52.191109	48.128568	50.322842	44659000.0
2004-08-20	101.01	109.08	100.50	108.310	22834300.0	0.0	1.0	50.661387	54.708881	50.405597	54.322689	22834300.0
2004-08-23	110.76	113.48	109.05	109.400	18256100.0	0.0	1.0	55.551482	56.915693	54.693835	54.869377	18256100.0
2004-08-24	111.24	111.60	103.57	104.870	15247300.0	0.0	1.0	55.792225	55.972783	51.945350	52.597363	15247300.0
2004-08-25	104.76	108.00	103.88	106.000	9188600.0	0.0	1.0	52.542193	54.167209	52.100830	53.164113	9188600.0

Figure 1: First 5 rows of the raw data, without any preprocessing

Exploratory Visualization

First, we see how the labels behave with time. For example, if we consider Google's adjusted close values between 2004 and 2017:

Figure 2: Adjusted Close values for Google (2004-2017)

We see that although the trend is clearly upwards, there are ups and downs, including a roughly 50% fall between 2008 and 2009 (the Great Recession).

Now, let's see how the features correlate with each other:

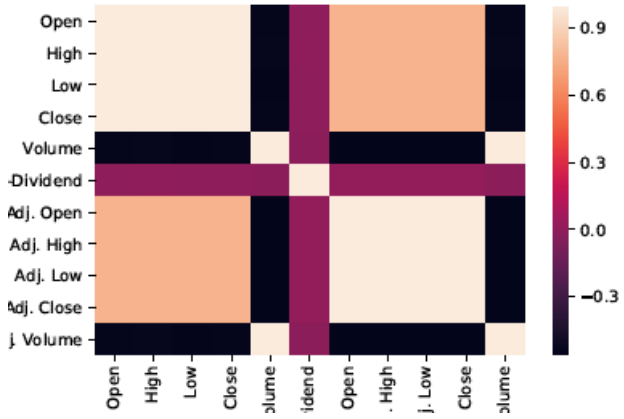

Figure 3: Correlation Matrix of the features for the Google stock

We can see that there are a lot of values that are mildly correlated with each other. The Split-Dividend feature has zero correlation, but it is because it has only a value of 1, so we dropped it.

Algorithms & Techniques

The work is divided into three sections:

- Data acquisition and preprocessing: Stock data will be taken from Quandl open databases using the API they have for Python. For now, we consider only Google and Microsoft, although this analysis can be extended to any company with the same or similar features. As usual, data are divided in a training set and a test set. In this case, 80% of the data are used for training and 20% for testing.
- Estimation: It will be done by using a Long Short-Term Memory (LSTM) rNN. The reason for it is that its recurrent nature makes it suitable for time series, and it is unclear how long the past will influence prices in the future. Some details about the architecture:
    - Since we are dealing with regression, some error measurement has to be chosen. In this case MSE was chosen as a loss metric when training. Random shuffling was disabled.
    - 30% of the training data was chosen as validation data in order to analyze overfitting. Dropout was also considered to avoid overfitting.
    - At the end of the recurrent layer(s), a Dense layer with one output was added which will yield the final result.
    - The weights of the lowest validation error will be saved, and then loaded for further usage.
- Evaluation: It will be divided in two parts:
    - Estimation error in the test data, by using MSE and comparing it to the benchmark, that will be described later.
    - Returns evaluation: We want to beat the market, so hopefully our algorithms will outperform a buy & hold strategy.

A flowchart of the project work is illustrated below for clarity

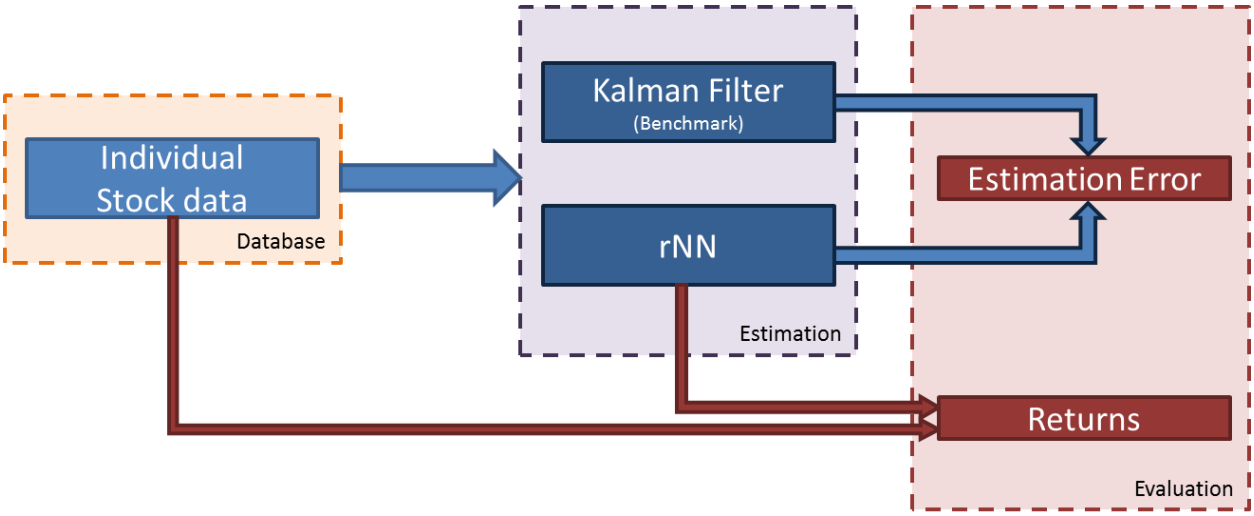

Figure 4: Flowchart of the whole project

Benchmarking

Two benchmarks will be considered for this project:

- A 3-step autoregressive (AR3) linear model for estimation, inspired by Martinelli & Rhoads work (see references). In this model, the prediction will be given by a linear combination of the stock prices up to three days before. However, due to the stochasticity inherent to the stock market, one must use strategies to account for it. In particular, the Kalman Filter was used, which is proved to be optimal as long as the stochasticity is Gaussian white noise. MSEs of both approaches will be compared.
- Buy & Hold strategy: In order to evaluate the returns with respect to a vanilla buy & hold strategy, to see whether our proposal would work to make more money.

Methodology

Data Preprocessing

In addition to dropping the Split-Dividend feature, some preprocessing is needed in order to use rNN. Since they use stationary data, one possibility would be to use the difference between two consecutive time steps. However, due to how values can change greatly over time, this is not a suitable approach.

Instead, the simple return mentioned above will be used as labels for the rNN. After the transformation, min-max normalization will be performed, in particular between -1 and +1.

Implementation

The estimation part was conducted in four steps:

- Preprocessing as mentioned above, i.e., transforming the features, splitting the sets… also, a random number seed was defined with numPy to keep control over random numbers, which are used in e.g. the Dropout layers.

- The rNN was implemented using Keras with TensorFlow backend.
    - It should be stressed that we will only consider one-step prediction, i.e. when predicting, we will load the data up to time t-1 and then use the weights to predict the stock value at time t, and so on. This is considered to be realistic, since new stock data are available when the markets close. The rNN will not be trained when incorporating the new values, although it might be useful to do that if we want to use it for a long time (say, one or two years).
- The Kalman Filter was implemented using the regular formalism with the matrices built as in Martinelli & Rhoads work, which yield a system with three states and one output. The considered system would be:

$$x(t) = Ax(t-1) + w(t)$$
$$y(t) = Cx(t) + e(t)$$

Where:

$$x(t) = \begin{bmatrix} x(t) \\ x(t-1) \\ x(t-2) \end{bmatrix} \quad A = \begin{bmatrix} 3 & -3 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad C = [1 \quad 0 \quad 0]$$
$$w(t) \sim N(0, Q); e(t) \sim N(0, R)$$

w(t) and e(t) are assumed to be white noise with covariance matrices Q and R, which are unknown.

Following Martinelli & Rhoads, both Q and R will be estimated by minimizing the innovation variance, i.e., how consistent are the estimates from the filter with respect to new measurements.

The quantity of interest is the one-step prediction $x_{t|t-1}$, i.e. the value of x at time t with measurements up to t-1, of the first state. This will yield our prediction of the stock value the next day.

After that, one can incorporate the measurement at time t and give a better estimate of the state. This is the essence of the Kalman filter. Said estimate, denoted by $x_{t|t}$ is computed by:

$$x_{t|t} = x_{t|t-1} + G_k(y(t) - x_{t|t-1})$$

which is the one that will be used in the model above. The quantity G_k is the Kalman gain. A more thorough explanation of the Kalman filter lies beyond the scope of this report, and there is a comprehensive literature available for the interested reader.

- Finally, both estimates will be compared to the true stock values, giving a final MSE value for both.

On the other hand, since these estimations are computed to increase returns with respect to a buy & hold strategy, it is interesting to see how they can be used in practice. To do that, the following was done:

- We start with some amount of money, say, 1000 monetary units (could be dollars, euros…)
- For a buy & hold strategy, we just invest all the money and wait, and see how much we get at the end.

- For the other two, part of the money is invested and part is kept in cash. Then:
    - Depending on whether the stock price is predicted to increase or decrease, we issue buy or sell actions accordingly.
    - To avoid running out of money, we buy or sell a set percentage of the cash money or the invested money, respectively. So for example, if we have 500 in the market and 500 in cash and it is predicted to fall the next day, we sell a fraction, say, 30% of it. So we will have at the beginning of the next day 350 in the market and 650 in cash.
    - We update then the money in the market with the real value from the dataset, and we repeat the process. If it worked, we would have sold when the stock was high and bought when the stock was low.

Refinement

Different architectures were tested for the rNN. It was found out quickly that overfitting was going to be an issue.

First, a simple LSTM network was tested, with just 32 nodes. It was not enough to catch the variability of the stocks so it was made more complex. A value of 128 nodes was found to be sufficient. The batch size considered was fixed at 32, which gave a good balance between accuracy and computing speed.

However, when trying to go to multi-level (deep) architectures, it did not improve the results. In Figure 5 and Figure 6 some learning curves are plotted for the different architectures. Surprisingly, in most cases, the MSE for the validation set was lower than the training set.

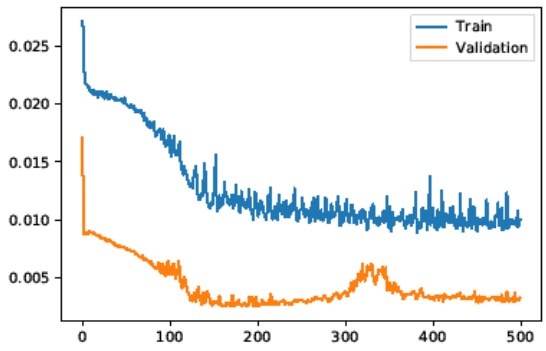

Figure 5: Learning curves for a single-layer architecture

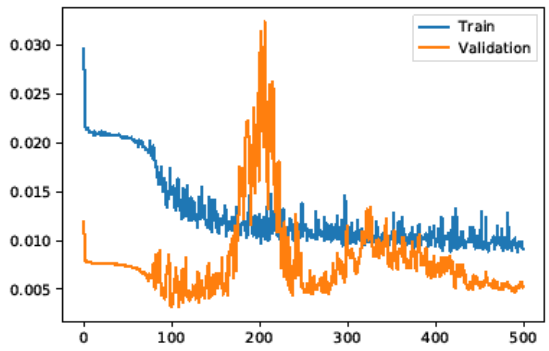

Figure 6: Learning curves for a four layer network. Note the overfitting happening at 200 and 325 epochs

However, a test with a double layer LSTM with 128 nodes and 50% dropout gave good results and was able to predict better over longer horizons.

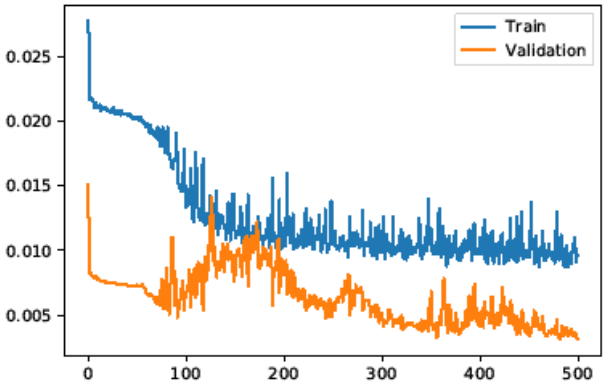

Figure 7: Learning curves for a double-layer architecture

For the Kalman Filter, the algorithm itself is quite straightforward and deterministic. However, the parameters Q and R must be estimated in order to compute the Kalman gain. As mentioned before, they will be estimated using the innovation variance.

R was assumed to be fixed at 0.1 and Q was assumed to be a diagonal matrix with the same value in all diagonals, i.e. $Q = qI_{3x3}$, where I is the identity matrix. A grid search was performed in Q, using values between 10^{-5} and 1 in a logarithmic scale, keeping the value that minimizes the innovation variance.

Results

Estimation: rNN

The final architecture took roughly 5 minutes to train in an Intel Core i7. It was tested in a GeForce GTX 1060 GPU but due to the small size of the dataset, large overheads were found and not a significant speed-up was seen.

The comparison between the real returns and the predicted normalized returns for the *training* set is depicted in Figure 8 below.

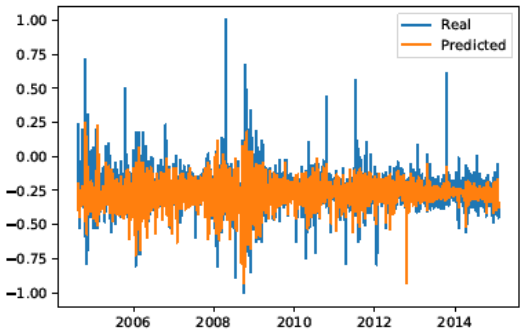

Figure 8: Real and predicted returns for the training set. Returns are normalized between -1 to 1

It can be seen that it captures the returns rather well, although the extreme returns are not so well captured, probably because they are quite rare.

The weights saved from the rNN were tested in the remaining part of the dataset, the testing set. Then they were de-normalized and translated to the stock values for better visibility. They are plotted in Figure 9. It can be seen a good matching between both, although there are some parts that are mismatched, which seem to be “extreme” changes. As seen in Figure 8, said extreme changes are not well captured by the network.

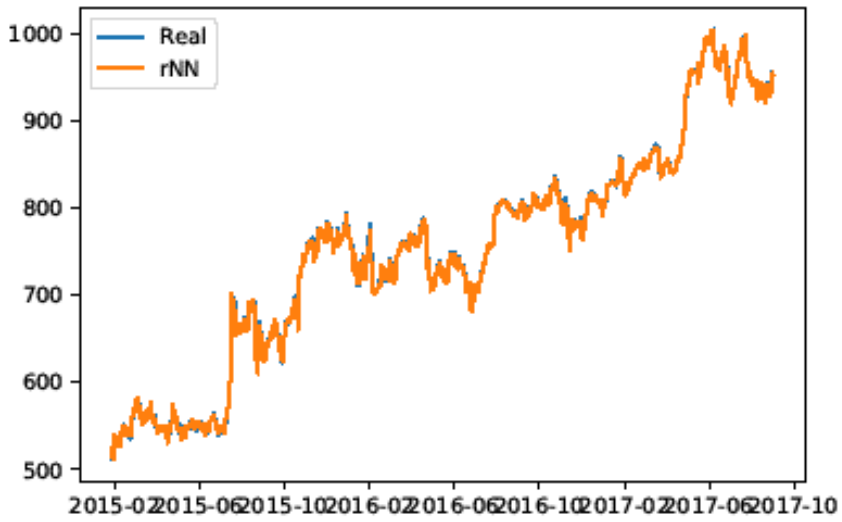

Figure 9: Adjusted close values for both real data and data predicted from the rNN for the test set.

Estimation: Kalman

The procedure described above was performed for the Kalman filter. A value of 7.49×10^{-4} was found for Q, which is 133 times smaller than R. This means that the model noise is much smaller than the measurement noise. This is indicative of a good model.

Adjusted closed values are plotted in Figure 10. A good matching can be seen, but the Kalman filter seems to overestimate the changes in stock prices, which means that the estimates are seemingly not as accurate as the rNN

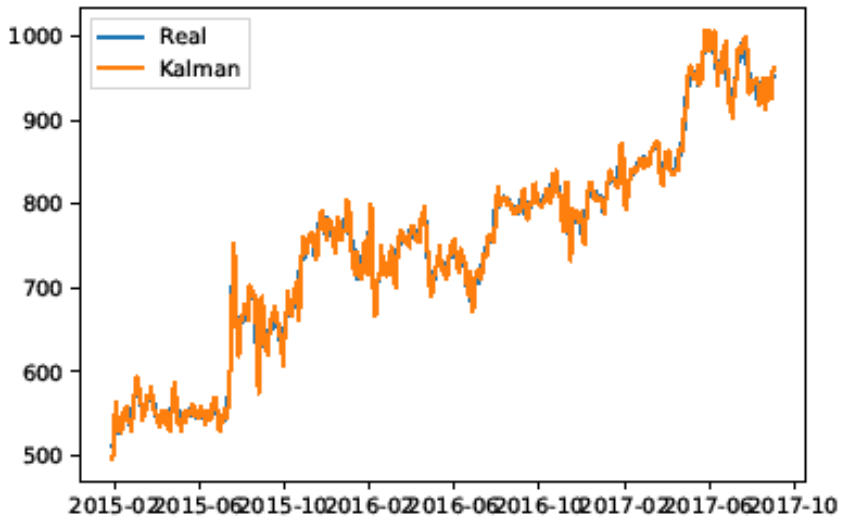

Figure 10: Adjusted close values for both real data and data predicted from the Kalman Filter for the test set.

Estimation: Comparison

Both approaches can be easily compared. Looking at their MSE:

MSE	
rNN	*Kalman*
7.04	194.45

and putting their predictions together:

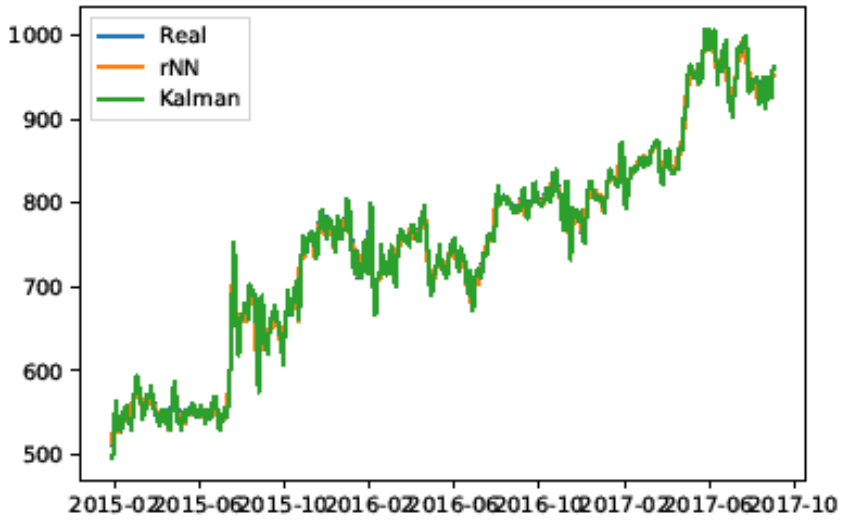

Figure 11: Adjusted close values for the test set, including real and predicted.

It seems that the rNN predicts better the stock prices in general. It should be noted however that the rNN has a random component, and this MSE value can vary.

Using predictions in practice: The stock game

While predicting the value of stocks is nice, the main purpose of it is to earn money in the stock market. Hence, it is worthwhile to see how they would work in practice.

The procedure was described above. The main parameters to adjust are the initial investment as a fraction of the total money and the fractions that we sell and/or buy at each time instant. Of course, these are relevant when trying to predict the stock prices, not in the buy & hold strategy!

We start with 80% initial investment. First, we assume a conservative strategy, with 10% of the money being moved (so either we sell 10% of our stocks or we buy stocks with 10% of our remaining money). Figure 12 shows us that predicting prices with this strategy perform roughly in the same way as the buy & hold strategy.

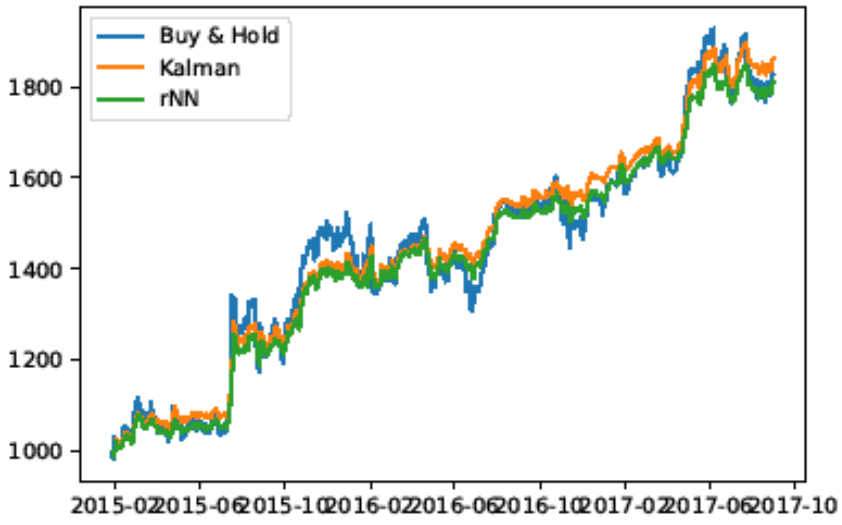

Figure 12: Money obtained by following our three strategies for 10% movements

However, if we are more aggressive, results could be much different. For 40% money movements, we get the results of Figure 13 which are much better.

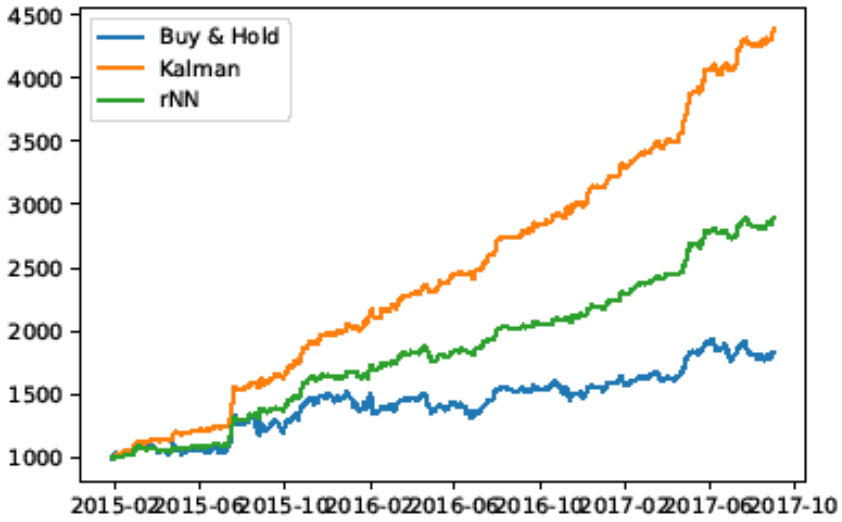

Figure 13: Money obtained by following our three strategies for 40% movements

However it is quite clear that for this game, the Kalman Filter performs better than rNN, and both significantly better than Buy & Hold.

Conclusions

The stock market has been proven to be a traditionally good investment and it is has been enough to just follow the market to achieve a decent return. However, higher returns are of great interest to investors worldwide. To do so, it is imperative to predict stock market prices in some way, or at least to know whether the trend is upwards or downwards. Here, one-day prediction was considered, although it could be potentially extended to other time frames.

Different approaches to how to predict stock prices using rNNs and Kalman filtering have been discussed. It was abstracted into a regression problem, with the adjusted close prices as labels.

In particular, it was seen that a double-layer rNN with 128 nodes and a dropout of 50% seemed to give acceptable results for estimation, with an MSE of 7.04. This result was compared to a linear AR3 model with Kalman filtering, and its MSE was 27 times larger for the latter. This showed good promise for the rNN when considering stock price estimation.

On the other hand, the Kalman filter was better at issuing buy and sell signals than the rNN, which limits the latter's potential for investing. Nevertheless, both of them were better than a regular buy & hold strategy.

As an interesting addition to the project, it has also been seen that a more aggressive strategy, i.e. using significant fractions of money, is better in order to obtain a better return on investments.

Ruben Cubo
September, 2017

Reflections and Limitations

Although the workflow has been engineered to simulate a realistic situation as much as possible, it is unclear whether this approach will work in real life.

In the estimation side, only a few features have been considered, in particular the ones that were freely available and simple to obtain. The analysis would be likely improved and better results obtained if more features were present, such as P/E ratio and other fundamentals.

In the "stock game" side, a few assumptions were made. First, we only considered adjusted close prices. Phenomena such as arbitrage and intraday variations were not considered, which can make stock values fluctuate and could ruin the strategy. In addition, buy and sell fees were not considered, which are one of the reasons why trading fails when frequent buy and sell orders are issued. Taxes and/or dividends were not considered either, which could make a buy & hold strategy more appealing.

Lastly, only one stock was considered, Google. Different stocks may behave differently, which can be looked into as future work.

References

R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems", Journal of Basic Engineering, vol. 82, no. 35, 1960

R. Martinelli, and N. Rhoads, "Predicting Market Data Using The Kalman Filter", Stocks & Commodities, vol. 28, no. 1, pp. 44-47, 2010

R. Martinelli, and N. Rhoads, "Predicting Market Data Using The Kalman Filter, Pt. 2", Stocks & Commodities, vol. 28, no. 2, pp. 46-51, 2010

https://www.quandl.com

S. Hochreiter, J. Schmidhuber, "Long Short-Term Memory", Neural Computation, vol. 9, no. 8, pp. 1735-1780, 1997