Stock Market Prediction - Buy or Sell?

Koren Maliniak (ID. 200319572), Ido Markovitz (ID. 305490104)

Submitted as final project report for Deep Learning, IDC, 2019

1 Introduction

The subject we chose to research is a reduction of the popular problem of stock price prediction from the finance field. The chosen problem is that of a trader who needs to decide when to buy a stock (long) and when to sell (short). It's a reduction of stock price prediction because we don't output a specific stock price, we just have to decide whether to buy or sell. We chose this problem because we learned that stock price prediction is a very difficult problem, and we wanted to challenge ourselves. We did look at other tasks, such as machine translation or classification of music, but we saw a lot of online implementations for it and that would've made things a little too easy. We had several goals in mind when approaching this project. First and foremost, we wanted to establish an end-to-end solution for a deep learning problem from scratch. That means: harvesting the data, shaping it, extracting features, choosing and building the model, training the model, and measuring results. Second, we wanted to see if we can "crack" this problem, as it has very obvious and distinct real-life applications.

1.1 Related Works

There are some examples online regarding stock price prediction (which you could then reduce into our buy/sell problem). None of the examples seemed to solve the problem. It makes sense that there are no published results that solve this problem because if anyone found a solution to this problem they would probably turn it into a hedge fund, and not publish it online. The following notebook is an example of a project aimed at solving stock price prediction. It managed to guess a trend, sometimes:

Using a Keras Long Short-Term Memory (LSTM) Model to Predict Stock Prices

2 Problem Description

We are trying to build a strategy for the financial stock market, that at the end of any given day, will propose whether to buy a stock, or sell it. We have in our hands a statistical, momentum-based financial model that for our purposes is a block-box. The input to the algorithm is the stock end of month close price, and its output are two series A and B.

The algorithm works well when back-tested on a monthly basis. Meaning inputting the current stock price of today, computing A and B and checking the yield. But when testing the algorithm ”day-by-day” as if it ”lives” in the past, the results are not that good. The reason is a lot of false positive signals. E.g. on the 15 of the month the algorithm outputs a signal to buy, but when reaching end of month it falls back. those 15 days are usually a loss in yield. Our goal in this project is to predict when a signal is false positive and when it is a true positive. Figure 1 depicts the output of the algorithm tested for a specific day in regard to some stock. Series A and B are depicted as the green and red lines. When the series intersect it means that we should either go into a buy (long) position or into a sell (short) position. This example is the wrong type of back testing because A and B are calculated with respect to the current present day. and not as if you are currently at each signal day as it was predicted in ”real life situation”.

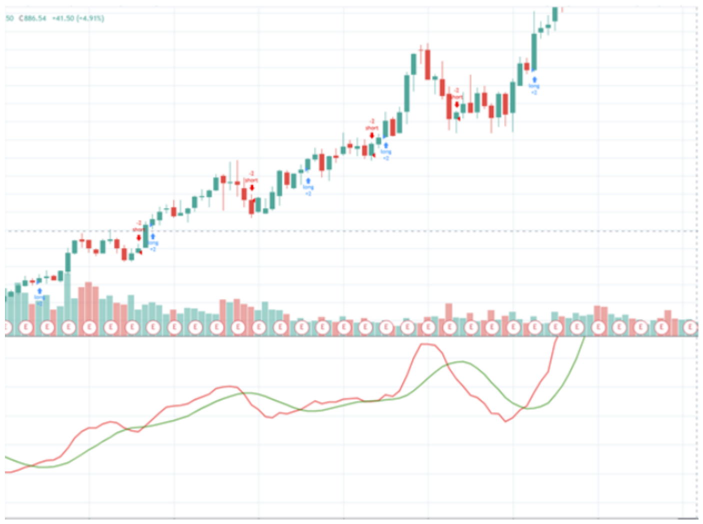

Figure 1: Series A (green) and B (red) intersecting to indicate when to buy or sell a stock

As mentioned above, the algorithm works well when its input is the stock price at the end of the month (and all previous historical data), but when tested on a specific day in the month we may receive false positive signal. A false positive signal occurs when the algorithm shows series A and B as intersecting at some day in the month, but as we gather more data towards the end of the month we learn that the series did not in fact intersect on those days. That means that the buy (or sell) operation offered by the algorithm was wrong, and we end up with a bad decision. The goal of this paper is to minimize these false positive signals so that we can improve our buy and sell strategy. Figure 2 depicts a false positive signal. On the first row in the 5th picture we can see the lines intersect which suggests a buy. Towards the end of the month, as we get more data about the stock price, we learn that the lines never actually intersected and that the buy was a false signal.

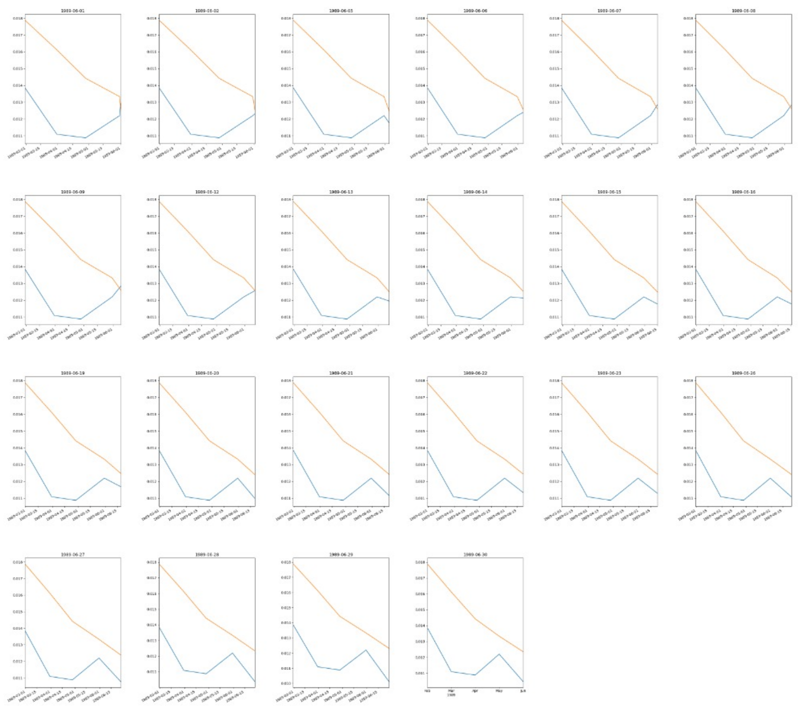

Figure 2: Algorithm outputs, each figure depicts the output of the algorithm given that day's closing stock price

3 Solution

3.1 General approach

As mentioned above, the goal of our project was to design, build and train a model that will allow us to identify our black-box algorithm's false positive signals and therefore improve its buy/sell trading strategy. Our solution outline is as follows:

1. Procuring raw stock price data for processing
2. Running the data through our black-box algorithm (creating the tags)
3. Creating features for the tagged data
4. Designing, creating and training our model
5. Testing the model

Our model results will then be tested against classic models using the same input data. We will now go into further detail for each aspect of the solution.

3.2 Procuring raw stock price data for processing

We used raw stock price data for the S&P 500 index. Later on we decided to expand our universe by using the Russel index (3000 stocks).

3.3 Running the data through our black-box algorithm (creating the tags)

We started off by running a few for loops in order to run through the raw data and feed it to the black-box algorithm. ("day-by-day") This turned up to be slower than we thought, so we improved it by changing the architecture to a vectorized one using pandas and numpy. This gave exponential improvement in speed. We also looked into Python green threads (a user-space multi threaded approach), but decided not to implement the solution since the vectorized approach was working well enough for our needs.

3.4 Creating features for the tagged data

The created features are composed of different families.

1. Raw Stock price
2. Time series values as outputted by the black-box statistical algorithm
3. Pre-engineered features that we thought would help such as:
    (a) The angle between the series at a certain point

(b) The distance between the lines

(c) The linear equation of each line

(d) Momentum of the difference between A and B (total of ups and downs)

(e) The distance to the point of intersection between the two lines

3.5 Designing, creating and training our model

Throughout the project we tried several different architectures as well as several models from different learning families. We added more and more things to the model to try and improve our results. The following list outlines the process:

i. Started with a simple LSTM to which we fed the data as input (monthly data)

ii. Added batch normalization and dropout

iii. Added several dense layers

iv. Using methods like over samplling or class weights because our classes were 66% vs. 33%

v. We realized we should create and add daily and weekly data, as we explored and visualized our data we saw that coming to a signal the daily and weekly re-sampled data can also have predictive value

vi. Adding daily and weekly data caused us to look at the Keras functional API as it required low level operations

vii. Ended up with 3 sub-models each composed of several LSTM layers whose outputs we connected to a final layer that calculated the result

viii. Tried a bi-directional LSTM

Figure 3 depicts the model as output by Keras.

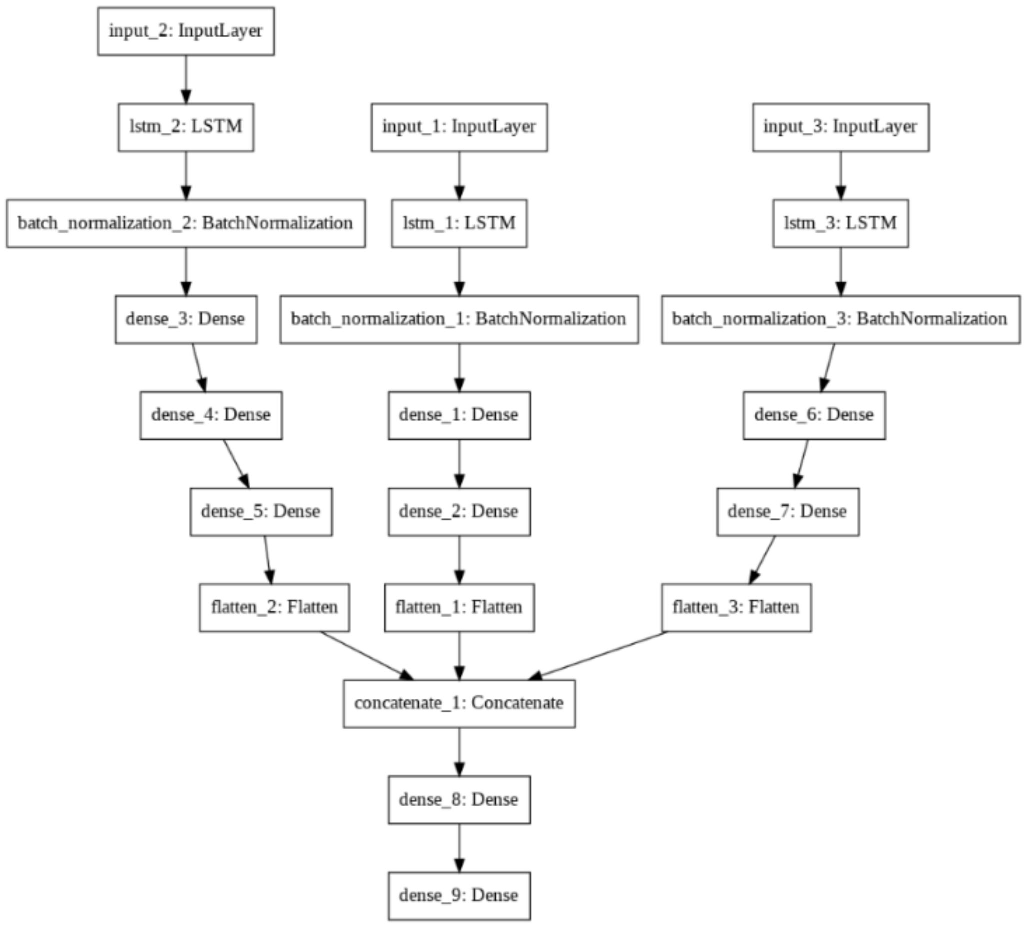

Figure 3: The final model

3.6 Testing the model

The loss function we used is a standard binary cross entropy. The score for prediction is classification score. While training the model we encountered a problem where the loss function is not identical to what we are trying to measure, and using the naive greater than 0.5 probability function didn't suit our cause as well. We therefore created custom accuracy functions, with different threshold. SO discussion regarding loss vs. score.

4 Experimental results

We didn't manage to improve our false positive rate and predict a buy/sell strategy. Either the model over-fit the training data and predicted a random chance on the test or it always predicted the majority class. Figure 4 depicts loss and accuracy.

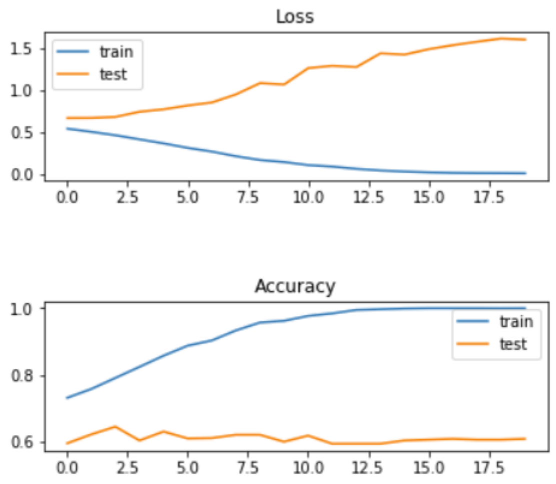

Figure 4: Loss and accuracy

We tried using several classic models as well to compare, and we ended up getting similar results. The classical model implementatoins can be found in the colab and include SVM, Random forest, Decision tree, KNN regressor and Gradient Boosting.

5 Future work

There are several things we can do in the future to try and improve our results:

1. Log scaling the information of the stocks in order to normalize the data
2. "Playing" around with the hyper parameters of our learning network as well as the architecture itself (there our automated libraries in python for that)
3. Extracting PCA and using it
4. Running the network using only our extracted features
5. Running the network only using raw data
6. Randomly combining features in order to empirically improve our results

7. Implement and use Attention Layers to try and improve our results
8. One algorithmic direction can be to create a heuristic regarding when to buy. E.g. not to buy at the first day of the signal, but wait for 2 consecutive days with signal. When researching a bit in that direction we found that this simple rule dropped the false positives rate by 20%, but also effected the yield of the entire strategy. So an interesting direction can be to combine a heuristic with some deep learning model probability.

6 Code & design

Short description of the github code:

1. Digestors - wraps the logic of structuring the data and creating the features
2. Signals, features - the logic
3. Data - handles getting the data via API
4. Main - runs the code

https://github.com/koren88i/DeepLearning-ClassificationOfFinAlgo
https://colab.research.google.com/drive/1yO3sqYQ77ZDAArc8GVKQBmghUFKrrQj6

7 Discussion & summary

It seems that the problem we chose, that is eliminating our false positives and achieving a better buy/sell strategy is not very far from prediction stock price. It's not very far from predicting stock price in the sense that it's a difficult problem that's not easily solvable. We thought that there would be a difference between artificially created features in order to classify a signal vs. end to end learning for price prediction, but at least with regards to our problem and solution there was no difference (both produced the same results). Classical models achieved the same results as our model. In some of our experimental results it appears that we over-fit the model. It is therefore possible that getting more data could have helped us achieve better results. We could potentially use all U.S stock market stocks data, but that would've only tripled our data, which wouldn't have made much of a difference for a deep learning model (still same scale of the number of examples).

To conclude, we managed to tackle a learning problem and implement all stages in the research and development cycle. From data gathering, through feature engineering, to model building, optimization and testing. Even though we were not able to reach our goal, we learned a lot in the process.