**School of Media Arts and Technology**

BSc (Hons) **Software Engineering**

**Academic Year 2020-2021**

**João Leite**

**Software Development Project**

Assignment 1

**Machine Learning**

**Tutor: Dr Olufemi Isiaq 16/01/2021 ­**

# Introduction

This report is about the development of an Intelligent Stock Trader software, designed for SOLFINTECH, with the purpose of aiding customers make smart decisions when investing in the stock market. Customers should be able to enter how much profit or the time interval they want to see the prediction for, and the software would return the best stocks to buy in order to make that profit, it should also provide a visual representation (graph), comparing the historical data and the system prediction, so users can evaluate the risks further before buying.

This application was done with Python and several packages (TKinter, Sklearn, Keras, etc.) that helped the overall development of the solution.

# Problem Definition

Every year there are tons of new people getting into the stock market, and with artificial intelligence being increasingly popular, we have seen a range of adaptions of algorithms to try to find the patterns of stock prices. Of course, stocks have a lot of external characteristics that influence their prices such as inflation, trends, news articles, and others; those are examples of the features that we have no data for, on the other hand, we are using publicly available data so we can get a range of values (as well as lots of information about the company) for stock prices.

The application goal is to make accurate predictions for the near future close price, for a desired time interval of a stock, based on its historic metrics (open, close, high, low, adjusted close), these predictions can then be used to aid users to make decisions when choosing when to buy/sell and which stock to invest and when.

I am aiming for a prediction success rate of at least 51%, which in a real case scenario it would allow you to profit from the system.

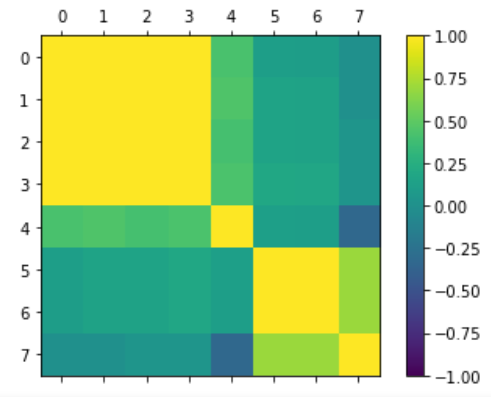
# Data Collection

In order to get the data, I am using YahooFinance API along with the yahoo\_fin library and several of its modules. These allow me to get the daily prices in a time interval of a stock. I decided to train the module using different datasets to understand how different lengths and data could influence the learning of the algorithm.

The API returns large datasets (at least 2000 rows for open, close, high, low, volume, and adjusted close prices), so with the use of data exploration techniques and tools we can understand our data better, learning about feature relations and relevance which helps determining which features will be appropriate to the prediction.

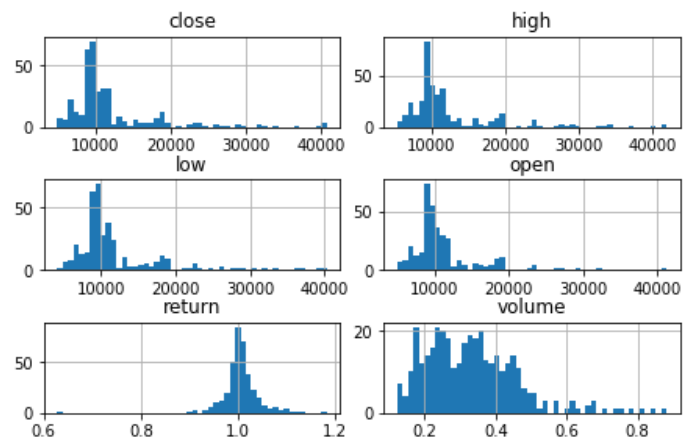
I also added a Return column which is the percentage of growth from the previous day adjusted close to the present. When exploring the data with these features the results were way more interesting as the values showed a lot of variety.

On the original dataset, I am using a correlation matrix to find relationships between the columns, the graph shows all relations, except for volume (number 4 in figure 1) and return (number 5), to be very high, which is quite normal as the daily values of a stock usually do not have drastic changes. On the other hand, the volume seemed to have around a 50% correlation in all features, this shows that the number of trades can have somewhat an influence on price, the return column shows even less effect at around 0% correlation, so I have decided to remove it completely from the training data as it would only contribute with randomness.



1 - Correlation Matrix of Tested Features (0-4 are stock price default values, the rest are the extracted features)

With a histogram and a density plot we can look individually for skewness in the features. The plots tell us that again, all columns except volume show positive skewness rates (~2.5), which means the mean value is higher than the median.



2 - Histogram of data collected features.

I have explored the data of 5 companies and found the results to be very similar. Although stock prices vary a lot, the relation between features and distribution of data seem to follow a pattern across stocks. I have also produced a scatter matrix plot where we can also observe the interaction of two variables. This helps support the previous findings that the features used have high correlation levels.

I am getting a total of 20 stocks in order to provide a good range of investing opportunities.

# Model Training

Because we have a sequence prediction problem, I decided to use the Long short-term memory architecture (LSTM). It is an architecture adaptation of recurrent neural networks, which works with multiple layers of functions (cells) with results that must be learned, it uses the results as input in order to learn from them and distributes their weights across repetitions, the final output will be the pattern the model learned from the data. LSTM was created to be a combination of Backpropagation Through Time and Recurrent Real Time Learning algorithms and improve on its problems, LSTM can hold its long-term memory for much longer without suffering the optimization problems the previous algorithms have. Despite being a good algorithm to use when training the model with several features, compared to inputting only one of the fields (adjusted close price), it showed slightly worse performance, when using only one it gave somewhat more accurate predictions and made the model less complicated.

I am using a sequential model from the Keras package and configuring it with two LSTM layers, one with 50 units (dimensionality of output) which I set to return sequences in order to get the hidden output of each iteration and define the input shape to the training set shape; the other to 50 units and a dropout of 30% of the outputs.

I have tried training the model with several other configurations, but these were the ones that showed the best results across multiple datasets. I have noticed that we can easily over or underfit our model by changing the neural network settings, my current model shows signs of overfitting, which means the algorithm is partially learning from noise in the data.

For the model compile settings, I have set the loss function to reduce the mean squared error, but still, the value is way higher than desired, I also tried to reduce the mean absolute error but I could not find any significant difference in the results. As for the optimizer I am using the Adam algorithm to update the neural network weights while iterating through the training data, in contrary to the well known stochastic gradient descent method which maintains the learning rate while training, Adam implements the benefits of both AdaGrad and RMSProp algorithms and is one of the most efficient optimizers available.

Finally, I am training the model using my x and y datasets and setting it to iterate twice on the dataset and to load 32 sample per weights update, the verbose mode is set to 1 which shows a progression bar for each iteration (epoch) when training the model.

With the model ready, we need to store it for later use, I am using Keras save function to save the model as a H5 format, used to store and manage large amounts of numeric data, in our case is saving the model architecture, the weights, and the compile algorithms. I can then load the model in the application using the load\_model function and use it to predict the prices using the data sets.

Another model I considered was ARIMA, which in contrary to LSTM is not a neural network algorithm, but it can also be used to find trends in time series data, these trends can then be analyzed in order to get a further understanding of the data patterns or be used to predict future values. ARIMA can be split into three main components and takes three parameters respectively: Autoregression where we can set the value to be the terms that will produce values based on previous output, same as LSTM (p); integration which subtracts the output from the current time step to make the data stationary (d); and the moving average that portraits the error of the model based on the combination of errors from previous terms (q), the value we insert is the number of moving average term to be included (Shawl 2020).

# While researching the models I found that ARIMA models, besides being faster and easier to implement, are great to predict a single time series, however, when used on a different dataset its performance is likely to be worse. This is where recurrent neural networks make a difference, despite being more complex to implement and slower to train, as we are working with large data sets, LSTM is optimized to break them into batches, is also particularly useful to predict into longer time series by adapting itself to learn feature relevancy while training the model.

# Model Evaluation

Since we are working with historical data, I am evaluating the model performance by predicting the past and comparing it with the actual stock price at that time. Instead of using the conventional holdout method, I choose to evaluate the model performance on datasets of different stocks than the one used to train it. This gave a way to compare results across several companies to make sure the model was predicting accurately. My metrics for the evaluation were MAE, MSE, RMSE and R2 to calculate the variance between the prediction and the historical data.

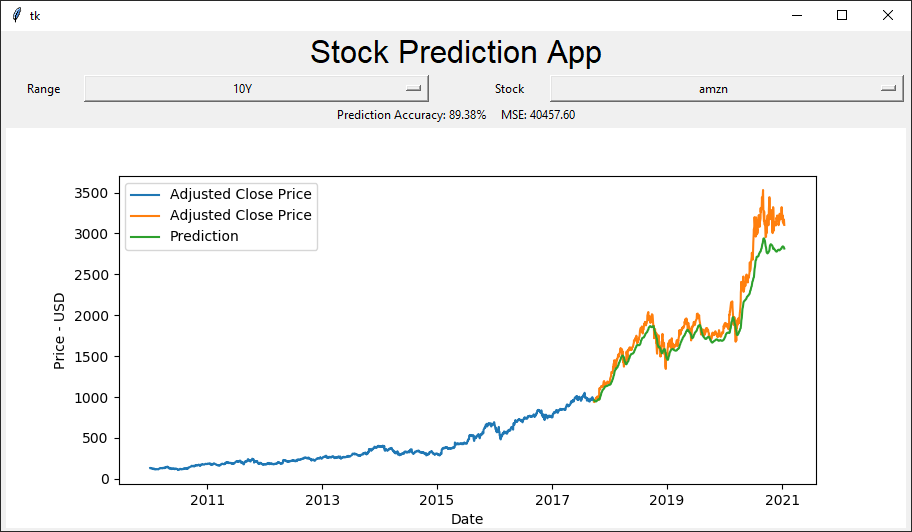
While trying to enhance the model, I have discovered that training it with larger dataset was beneficial to the prediction score, and that certain stocks seemed to result in better effects, although I am not entirely sure, it seems the algorithm works better when the datasets show different patterns instead of a stable growth pattern. On the other hand, giving it small amounts of data (datasets with less than about 500 rows) gives very inaccurate predictions most of the time.

Despite my effort, the model still shows signs of overfitting as the prediction’s accuracy vary a lot from stock to stock, still, the final application predicts accurately on almost all datasets, showing R2 accuracy levels between 50 and 100 percent on most, but falling drastically on others.

# System

The system interface was done with the Tkinter library, my focus was to make its use as straight forward as possible. Being only the second time using it, I struggled a bit with making the UI appellative, and still think it could look better but I ended up going for simplicity and make it responsive. With more time I would have like to use a different tool to build the interface. The PySimpleGUI library looks to be a great option or even a front end with html to allow interactivity with the graphs which unfortunately is not available in Tkinter.

The final application allows you to choose, from two dropdown lists, a list of 20 stock companies, and three different time ranges. It displays a graph of the historical financial data of the stock chosen, for the time interval selected, as well as a prediction for the last 30% of data available. It allows for a better understanding of the stock trends and facilitates the analysis of the model performance and it also displays the current plot evaluation metrics.



3 - Final Interface

# Conclusion

I learned a lot about concepts from the finance, math, and mainly machine learning fields. Although it made it harder, developing this application with python was great to get me more familiar with the language.

My research on the algorithms exposed the depth of machine learning for me, and specifically how neural networks are being used to solve complex problems.

Unfortunately, I am struggling to understand how I would use my particular model to predict future values instead of historical data, so the system is only adapted for the latter.

Further improvements to this application could include a better tuned or even different model, to increase prediction accuracy, using the model to predict prices in the near future, and using a different library for the interface. Although Tkinter was great to use, it is fairly restricting on what type of interface you can make.

All the code for the data preprocessing and model training, as well as several attempts with differently tuned models, are included in the project directory to provide further insight into my development process.

# Bibliography

BROWNLEE, J., 2013. How to Define Your Machine Learning Problem, Available from: <https://machinelearningmastery.com/how-to-define-your-machine-learning-problem/>

BROWNLEE, J., 2017. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning, Available from: <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>

CHOUHBI, K., 2020. Preprocessing Data: Feature Scaling, Available from: <https://towardsdatascience.com/preprocessing-data-feature-scaling-cc28c508e8af>

DIVA DUGAR, 2018. Skew and Kurtosis Available from: <https://codeburst.io/2-important-statistics-terms-you-need-to-know-in-data-science-skewness-and-kurtosis-388fef94eeaa>

JORDAN, J., 2017. Hyperparameter tuning for machine learning models., Available from: <https://www.jeremyjordan.me/hyperparameter-tuning/>

MITCHELL, C., 2019. How to Use a Moving Average to Buy Stocks, Available from: <https://www.investopedia.com/articles/active-trading/052014/how-use-moving-average-buy-stocks.asp>

NETHALA, A., 2018. Layout Managers in Python GUI — Pack, Grid and Place, Available from: <https://medium.com/programminginpython-com/layout-managers-in-python-gui-pack-grid-and-place-af0f14303178>

SHAWL, S., 2020. Python | ARIMA Model for Time Series Forecasting, Available from: <https://www.geeksforgeeks.org/python-arima-model-for-time-series-forecasting/>

SUNIL RAY, 2015. Beginners Guide To Learn Dimensionality Reduction Techniques, Available from: [https://www.analyticsvidhya.com/blog/2015/07/dimension-reduction-methods/](%20https://www.analyticsvidhya.com/blog/2015/07/dimension-reduction-methods/)

‌

‌

‌

‌

‌

‌

‌