Advanced Data Analysis Project

Google Stock Dataset

Week 5

LUT University

Alejandro Lopez Vargas
Sina Saeedi
Pau Ventura Rodríguez

Visit our GitHub to gain access to the script.

Introduction

In this project, we aim to analyze the historical data of Google's (GOOGL) stock price and create a predictive model for it. The dataset includes the following variables: *Date, Open, High, Low, Close, Volume,* and *Name*. The *'Name'* variable is constant and represents the company's name, which is Google in this case, making it unnecessary for our analysis.

Here are brief explanations about each of the variables:

1. **Date**: This variable represents the specific dates corresponding to each data point in the dataset. It serves as the timeline for the stock's historical performance.

2. **Open:** The opening price of the Google stock on a particular date. This is the price at which the stock begins trading at the beginning of the trading day.

3. **High:** The highest price at which Google stock traded during a given day. It provides insights into the highest level the stock reached within a trading session.

4. **Low:** The lowest price at which Google stock traded during a given day. It indicates the lowest level the stock reached within a trading session.

5. **Close:** The closing price of Google stock on a specific date. It is the final price at which the stock is traded for the day, representing the market sentiment at the close of the trading session.

6. **Volume:** The total number of shares of Google stock traded on a given day. Volume is a crucial indicator of market activity, providing insights into the level of interest and participation in the stock.

Data Preprocessing:

1. **Data Formatting:** Ensure that the date format is the correct one and build an appropriate dataframe indexed by date. Ensure as well that data has the same frequency between observations (in our case we need to have all business days). We have converted the Date column on the dataframe to format Datetime in order to give it an ordinal value. The stock market is only open on business days, which means we do not have data for all days of the year. Furthermore, some holidays did also not register data like Christmas, which we will create in order to have an equally spaced dataframe. Pandas allows us to create these extra samples by using the command: *df.asfreq('b')*, which will fill their values with NaN.

2. **Data Cleaning:** Dropping unnecessary columns. In our case the column “Name” only has one unique value, which does not give us any information on the target and will then be dropped. Check for missing values in the dataset and handle them appropriately. We have missing values as we created them when we added the extra days to keep the frequency. Unfortunately, we have 3 samples that have consecutive missing values. As all those samples consist of only clusters of 2 consecutive samples, we will use Frontfilling on the first instance and Backfilling on the last instance. To fill up NaN values, we can use Backfilling (setting up the values with the next datapoint’s values), Frontfilling (setting up the values with the previous datapoint’s values),

Meanfilling (assigning the mean value of that feature, which in this case will not yield good results due to the increasing tendency). In this case, we will use Backfilling. In addition, check for outliers in the data and consider whether to remove or transform them.

3. **Feature Engineering:** Create lag features for Open, High, Low, Close, and Volume. However, if we want to predict the Closing price, we will not have the High, Low, Volume nor Closing variables. Instead, what we will have is the values for those variables on the previous timestamp, so we can use on each sample the lagged version of Closing, Low, High, and Volume instead of the original one. This means including previous time steps as features. The number of lags is a hyperparameter that can be tuned. The samples will then be, given current day (cd) and previous day (pd): Open(cd), close(pd), Low(pd), High(pd), Volume(pd). So we will try to predict next sample's Close price (which will be the current day's closing price).

4. **Train, Test, and Validation Split:** Divide the dataset into training, testing, and validation sets, ensuring that the test and validation data capture the trend and seasonality patterns present in the training data. Employ a split ratio of (80-10-10) for the dataset allocation. The training split will be used to train our model. The validation will be used to know when to stop training the model and avoiding overfitting. Finally, the test split will be used to measure the performance of the model on unseen data. We can also perform cross validation to ensure that our data partitions are not biased and our performance results are as rigorous as possible. The type of cross validation used in that case would be Time Series Cross-Validation, which divides the training set into two folds at each iteration on condition that the validation set is always ahead of the training set

5. **Normalization:** It's common to normalize the data, especially when using neural networks, to improve convergence and training speed. For our analysis, we will use the Z-score normalization method on train data

and with the mean and standard deviation that we found from the train data, we will normalize the test and validation data.

LSTM Model

To train a LSTM model, the data requires to be Normalized and properly processed.

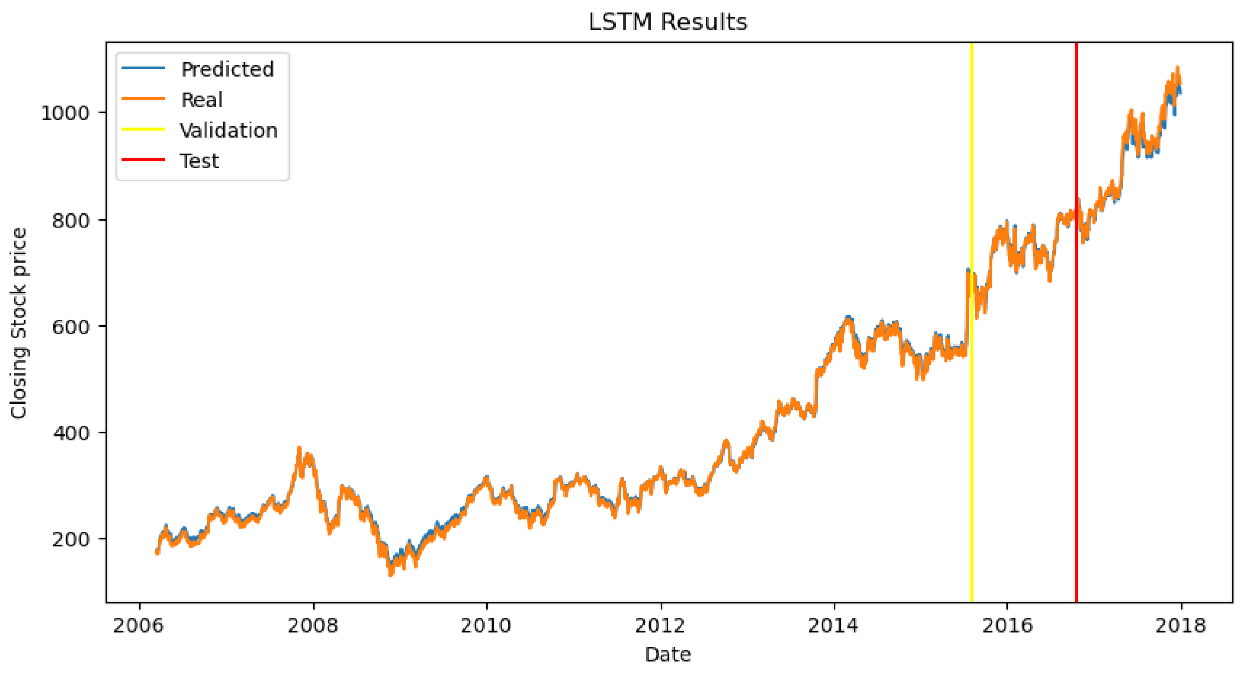
Normalization has been done training the scaler with the training set (80% of the dataset), which was then applied to the validation set (10%) and the testing set (10%).

LSTM needs the data to be fed in the shape [n_training_samples, n_timesteps_per_sample, n_features]. In our case, after trying different timesteps, we have decided to add 50 timesteps into our model, which results in information about 50 days before the current day. Also keep in mind that we had to include the current day's Open feature, so we had to drop last day's Opening feature. Although including 50 lagged variables drastically increases execution time, it adds a good amount of information to reach good predictions and, considering that each sample only has 5 features, adding 50 lagged variables only results in having 250 more features (which for a Machine Learning model is not that much). The chosen prediction window was 1 day, as we will simulate a short-period trader that wants to know if it is worth to buy stock when the day opens.

After the samples were reshaped, they were ready to be fed into the model. The parameters that needed to be considered are: the amount of neurons on the Hidden layer inside LSTM, the batch size, number of epochs and learning rate. The number of epochs was set into an arbitrary number (100) and then EarlyStopping was applied. This would allow the model to avoid overfitting and not rely on the number of epochs trained. After trying different combinations of neurons in the Hidden layer, we decided to set the number to 500, as our data has many underlying patterns and it's hard to model. Same for the learning rate, after different trial and error, we decided to set it up to 0.0005 to ensure a slow but converging phase. The batch size was set to 32, as larger batch sizes result in faster training times, but less accuracy and even overfitting. In the model we also added a Dense layer at the end to combine the output of the LSTM into the final output without any activation function. When compiling the model, we have used MSE as the loss function and the Adam Optimizer.

Training the LSTM model took almost 9 minutes, which is a decent value considering that we have 2500 training samples and 250 features.

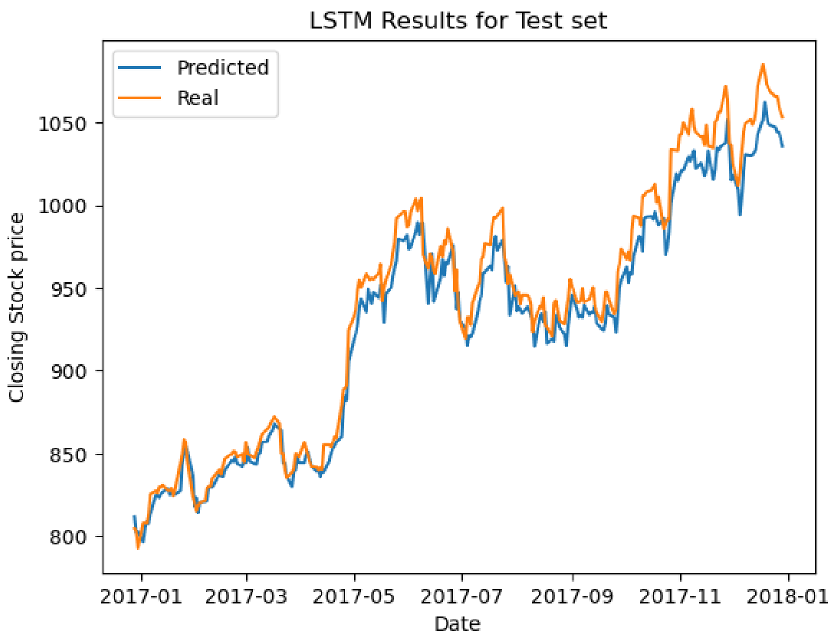
The results on training, validation and testing data are the following:



The model seems to have learned properly the trends and patterns of our data. On the training data specially, the prediction closely fits the real values, achieving a MAE of 5.70 and a RMSE of 7.34, which means that on average, we will fail the closing prediction in 5.7 units.

On the validation set, the prediction is not as good as in the training set, but still closely matches the real value, achieving a MAE of 6.00 and a RMSE of 8.17, which means that on average we will fail the closing prediction in 6 units.

On the testing set, the prediction is the worst of the three, as the model did not see the data and was not trained depending on it. Still, as we can see on the chart it performed quite good, with a MAE of 11.63 and a RMSE of 14.10. Here we can see a zoomed version of the testing set to avoid being mislead by the large scale of the dataset:



The model seems to have a good understanding of the patterns, as the predicted and real values have a very similar behavior.

Overall, the predictions are really good, but is it enough?

Well, considering that in the validation set the average difference between the Opening and Closing values is 0.28, we could set Closing = Opening + 0.28 and we get a MAE of 5.83, which is much lower than our current MAE. So a mean regresor with only one variable (Opening) performs much better than our model. Getting really close to our model's performance on the training set.

RNN Model

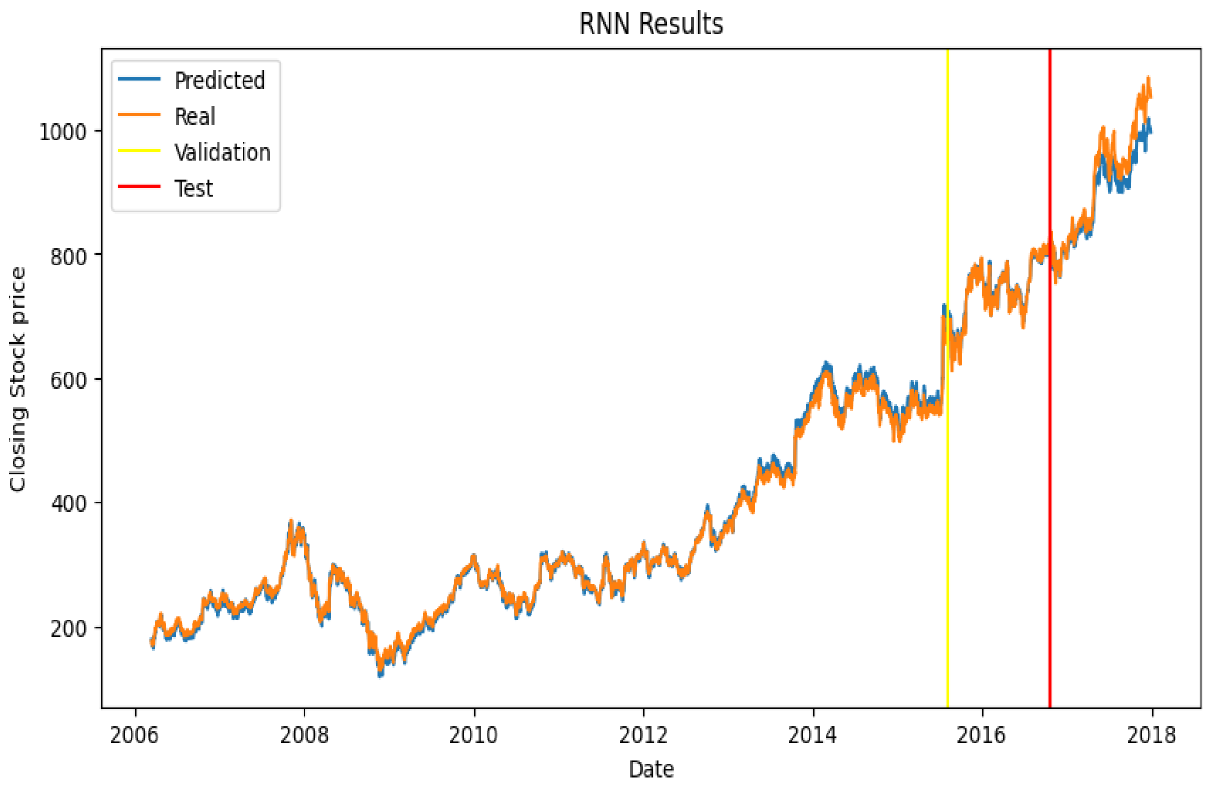
Recurrent Neural Networks (RNNs) are a type of neural network architecture designed for sequence data, making them well-suited for tasks such as natural language processing, speech recognition, and time series analysis. Unlike traditional feedforward neural networks, RNNs have connections that form a directed cycle, allowing them to maintain a hidden state that captures information about previous inputs in the sequence.

Here's a brief overview of the main components of a simple RNN:

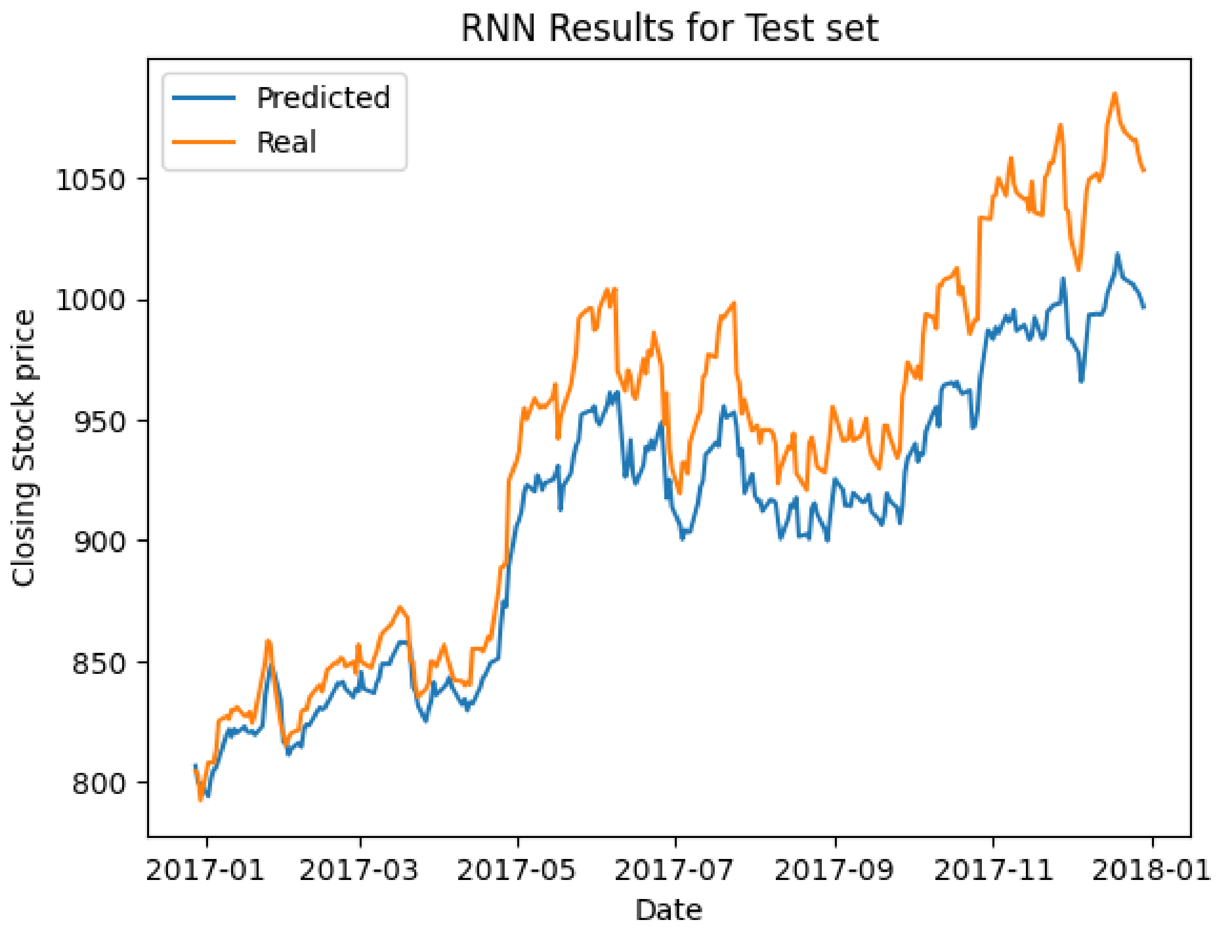
1. Hidden State: The hidden state at time step "t" is a representation of the network's memory or context, capturing information from previous time steps.
2. Input: The input at time step "t" represents the current element in the input sequence.
3. Output: The output at time step "t" is the prediction or representation generated by the network based on the current input and the hidden state.

In our scenario, our goal is to forecast the Close price of Google stock using the RNN model. Following the required preprocessing steps and the addition of lagged features, we establish our simple RNN model consisting of only one simple recurrent layer. The number of neurons in the hidden layer is set at 700 which is a parameter that can significantly impact the model's accuracy. Another crucial parameter is the learning rate, which is chosen as 0.0005.

In the following figure, we can observe the performance of our RNN model for train, test, and validation data.



The model exhibited satisfactory performance on the training and validation datasets. In addition, Initially, it performed well on the test data, but over time, it struggled to closely track the target Close price. The following plot provides a more detailed visualization of the test data predictions.



Furthermore, for the performance matrices, we have chosen RMSE and MAE for this model.

```
Test RMSE: 34.73758067021133

Test MAE: 29.6833082075808

Validation RMSE: 8.957153526603104

Validation MAE: 6.573888052553414

Train RMSE: 8.79016334167736

Train MAE: 6.854271679404635
```

Examining the matrices, it's clear that the model performed well during both training and validation. However, the model faced some difficulties when attempting to predict the test data. Yet, the satisfaction level with the Root Mean Square Error (RMSE) could vary depending on our particular criteria for accuracy and precision in the model's predictions.

Also, regarding the execution time, it can be stated that the RNN model is approximately 4 to 5 times faster than the LSTM model, as would be naturally anticipated.