Advanced Data Analysis Project

Google Stock Dataset

Week 6

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.

Transformer Model

Transformers, a groundbreaking machine learning architecture, revolutionized time series analysis by introducing parallel processing through self-attention mechanisms. Originally designed for natural language tasks, Transformers excel in capturing temporal dependencies, making them ideal for modeling complex patterns in sequential data. This architecture, introduced by Vaswani et al. in 2017, has transformed the landscape of sequential data processing, offering remarkable capabilities in tasks such as forecasting, anomaly detection, and pattern recognition.

To train a Transformer model and ensure that it converges, 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%).

The transformer model used has been adapted from the template provided in Transformer's exercise, as some changes have been made. For example, in this case we do not want to predict all the features but only the Closing price. Also the normalization has been fixed to make sure that the statistics are only calculated using the training set.

In terms of features, we are using 50 lagged variables to make sure that the model is able to capture the most amount of information (and as well because that was the optimal number for LSTM).

After the samples were reshaped and sequenced, they were ready to be fed into the model.

The parameters that needed to be considered are:

- Learning Rate: Dictates the phase at which the model will learn. A learning rate too large might cause the model to not converge, and a

Learning rate too small might cause the model to take too much time to converge.

- Number of epochs: Backpropagation uses Gradient Descent algorithm to find the optimum value of the weights. This involves making a certain number of steps, which might not be reached only going through all the data once. Thus, the number of epoch is the number of times the data will be sent to train the model.
- Batch size: The data might be too large to be handled all at the same time. For this reason, DataLoaders are used, which contains a certain amount of data on each batch.
- Number of Layers: Determines the number of layers of type transformer inside the model.
- D: Represents the dimensionality of the model, also known as the dimensionality of the hidden state.
- H: Number of attention heads inside the transformer.
- Hidden mlp dim: Dimensionality of the Hidden Multilayer Perceptron inside the transformer layer.
- Dropout: Defines the percentage of neurons that are removed from the architecture to ensure the network does not overfit.

Epoch, Learning Rate, Batch size and Dropout have shown to have the least impact on the model, so we have chosen to determine a default value of 300 epochs, 0.00001 learning rate, 32 batch size and 0.1 dropout. This was made to focus on the most impactful parameters. Those parameters then have been deeply analyzed along different ranges of values.

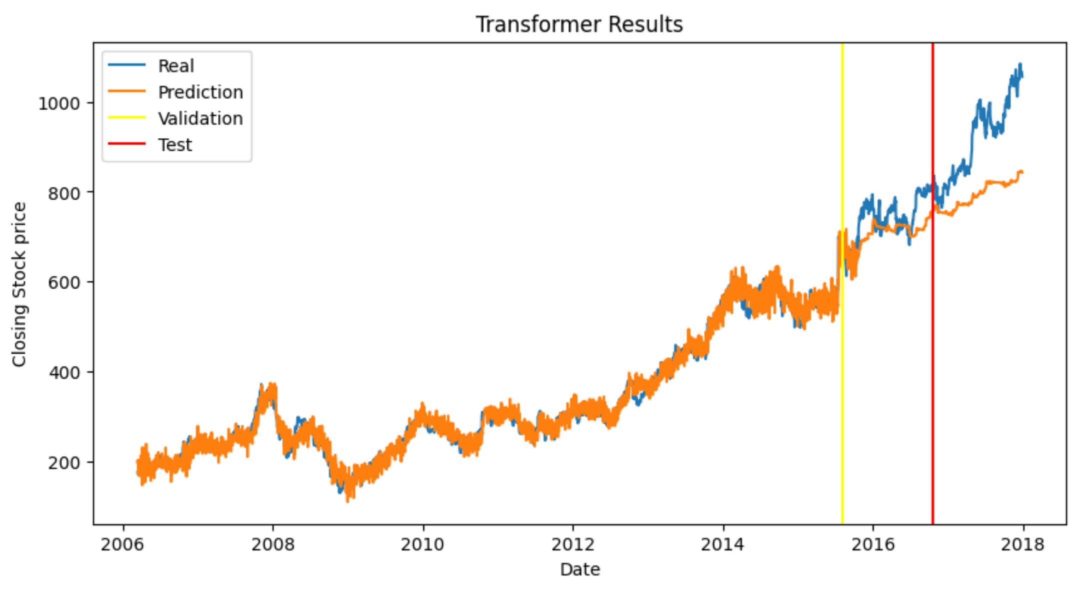
Trying all the possible combinations of these parameters and hyperparameters on the model would be a time-consuming task. To solve this problem, we will try random combinations of the values, the so-called Random Grid Search and compare the performances of the models.

The best hyperparameter found are:

Number of layers: 1, D: 256, H:4, Hidden layer dim: 96.

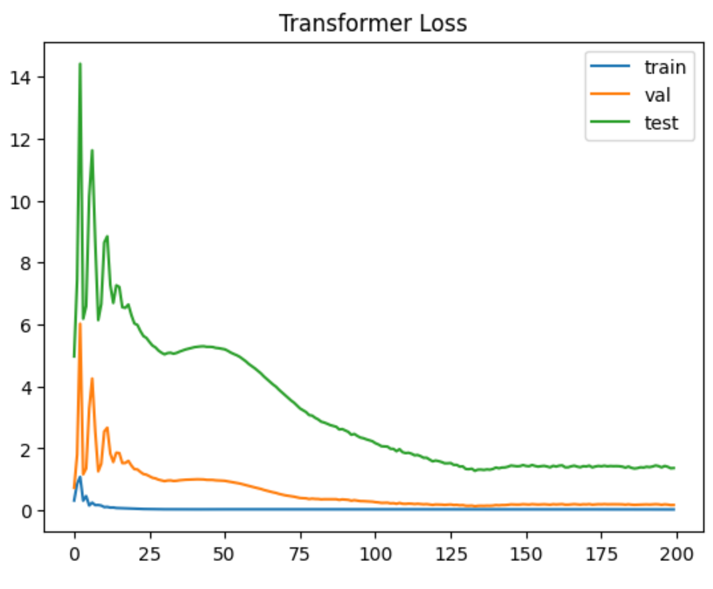
Training the transformer model took almost 3 minutes.

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



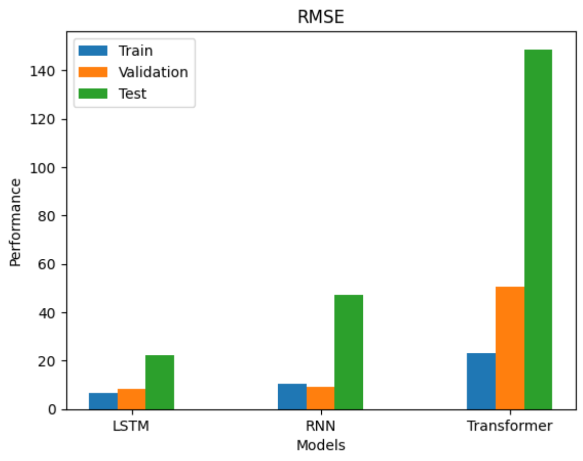
As it can be seen, the Transformer model did not perform properly, as the predictions are really far away from the real values. Even in the training data the predictions have a lot of variation and do not fully fit to the real values.

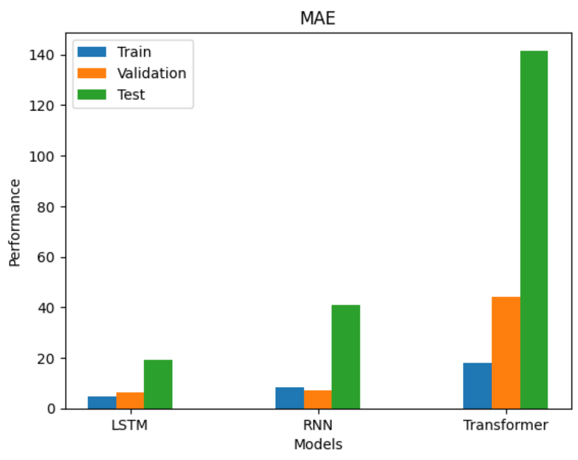
The validation and testing split also do not properly converge to the real value, and mostly yield really bad underestimations.



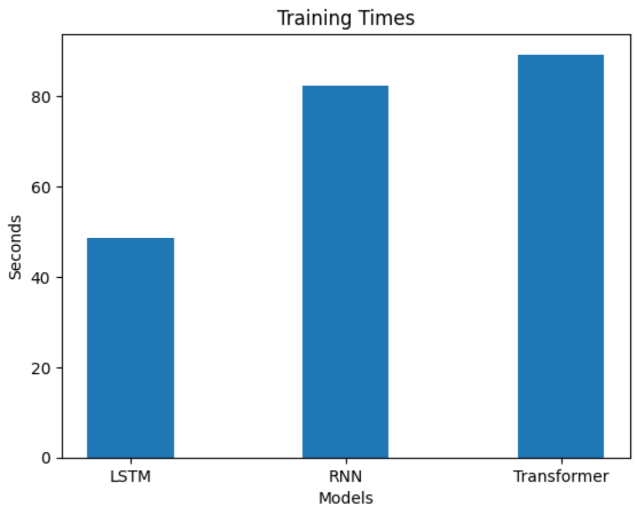
It can be seen from the loss that the test partition is very far away from the validation and training losses. This might be an indicator of overfitting to the data. However, increasing the dropout rate does not help improving this effect.

Comparing the models





Using both metrics to compare the models, we can conclude that the best model is LSTM, followed closely by RNN. The worst model has shown to be Transformer by far. Although all models have much higher testing error than training and validation, the LSTM seems to be the one that overfits the less and thus achieving the minimum error on unseen data. Note than, given the results, using LSTM ensures an average absolute error of around 20 on each prediction.



If we compare the training time, LSTM seems to still yield the best results, with a training time of 50 seconds, being the lowest of the three models, followed by RNN and Transformer.

Conclusion

Based on the metrics and different tests done, the best performing model seems to be LSTM by far, achieving an average error of around 20 dollars on the stock price, while keeping the lowest training time among all other models.

Overall, the predictions from LSTM 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.