ProgressSoft Assignment

Problem 5 – Time Series Prediction

Approach:

To predict the stock's most recent 20% prices, I used two approaches, one using Machine Learning and the other using Deep Learning.

The solution includes four files:

- ***TimeSeries_with_ML.ipynb***: Jupyter notebook to solve the problem with Machine Learning
- ***TimeSeries_with_DL.ipynb***: Jupyter notebook to solve the problem with Deep Learning
- ***prices.txt***: the data file which has the dates and the prices for the stock in txt format.
- ***prices.csv***: the data in CSV, the way of converting happened in the ***TimeSeries_with_ML.ipynb*** file.

The Dataset:

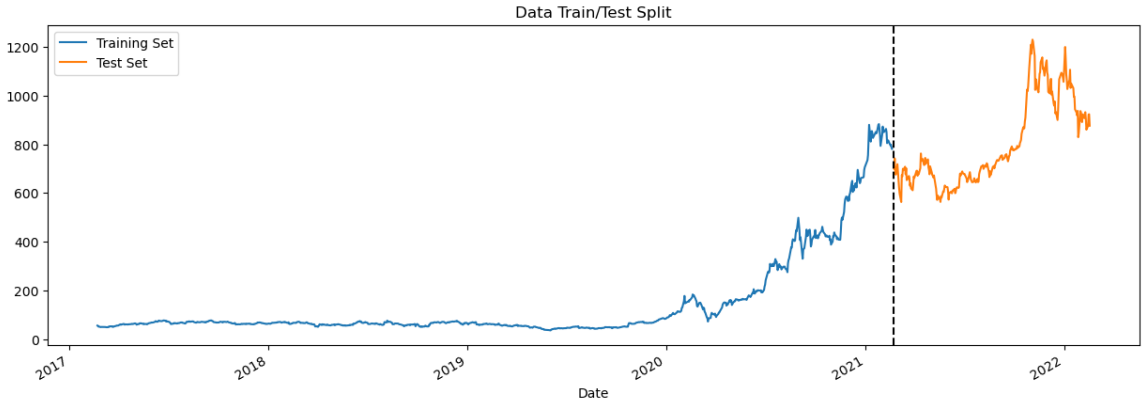
The Data come in .txt format which has values in a comma-separated way. It has 2 features, the **date**, and the **stock prices**. It has **1259** rows (instances) of data. The data has values for each stock for each day, and it starts from **2017-02-21** to **2022-02-17**, which is approximately **5 years**.

What I did is read this file in python, extract the values, and make them as Pandas data frame, make the data type for the date feature as datetime through pandas (for manipulating the data as a date data type), make the date as the index for this data frame, and save this data frame a CSV file using pandas ***.to_csv*** method.

Machine Learning Approach:

I first read the data from the prices.txt file, then extract the values needed and make a data frame from it, then save it as a CSV file. Before everything, I changed the date datatype to datetime, for working with it as a date, not as an object (string).

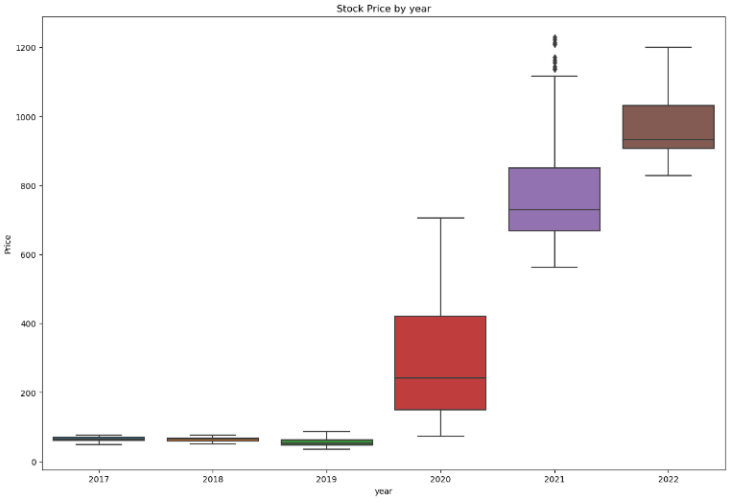
After that, I prepare the data for splitting. The splitting happened as the first **80%** of the data was the training dataset and the final **20%** was the test dataset. This give that we have **1,007** instances for the training data, and **252** instances for the testing data. This means that the data is **1259** instances (stock prices).



Train / Test Split for the Data

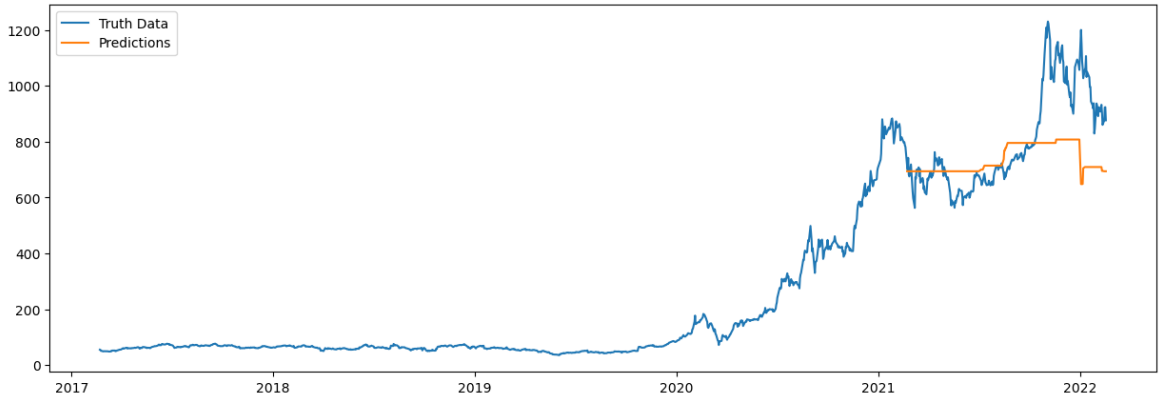
Then, I created a function called ***create_features*** and take data frame as input. This function creates features for the data frame, the features are related to the data and extract information from it, such as the day of the week, quarter, month, year, etc.

Then, do EDA (Exploratory Data Analysis) through some plots of the data. After this step, I split the data into features, and the target (X, y) just like it is a supervised learning approach.



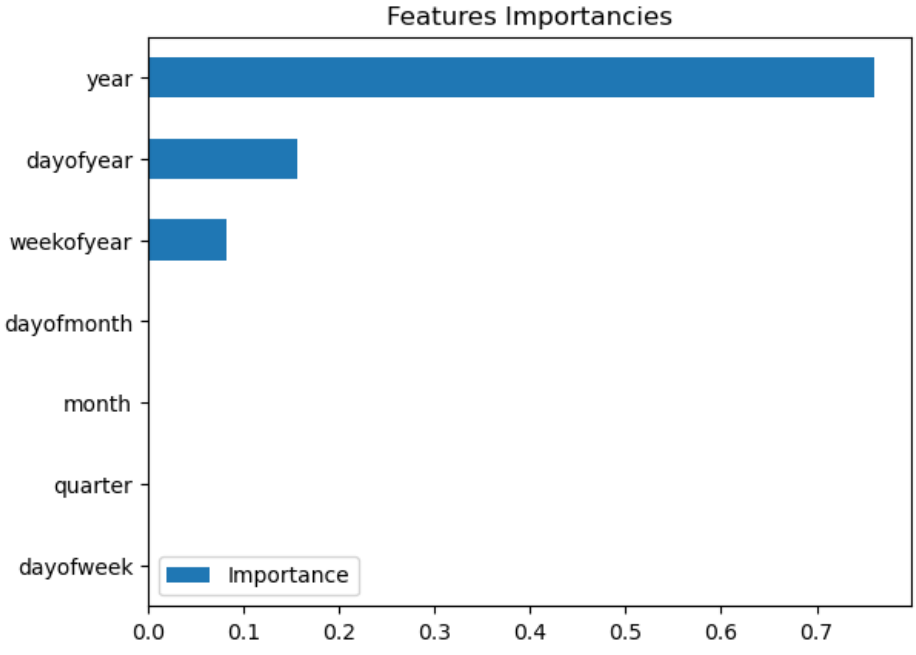
Stock Prices by Year

After that, build the model and train it on the data. The model I used is an ***XGBoost Regressor*** with a 1,000 estimator, 0.01 learning rate, max depth of 2, and evaluation metric as 'MAE' (Mean Absolute Error). Then, evaluate the model with the MAE (Mean Absolute Error), and RMSE (Root Mean Squared Error).



Model Performance on the Test Data

Finally, plot the feature importances for the model and horizontal bar chart, and we conclude from it that the ***year*** feature is the most important feature for the model.



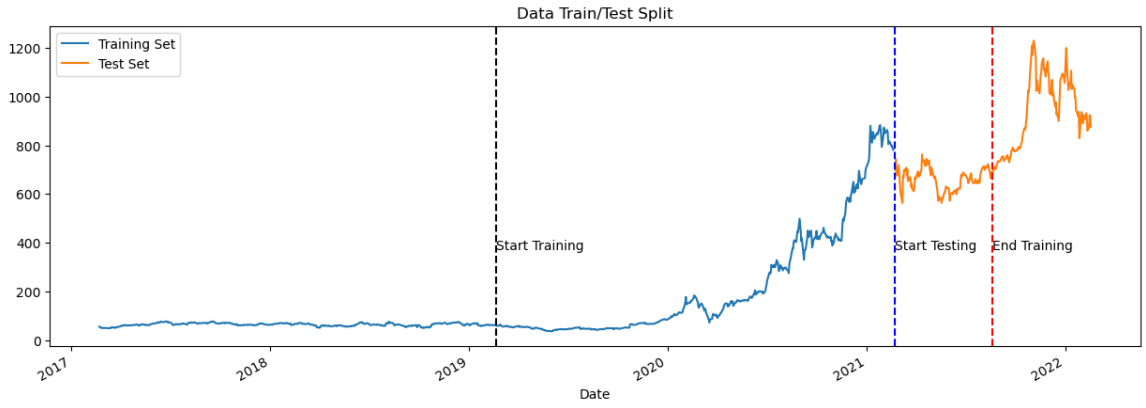
Feature Importances for the Model

The Deep Learning Approach:

I used a window method to preprocess the data, which involved creating windows of fixed size and predicting the next value in each window. I then defined 3 main functions I will use in the notebook, ***plot_series*** which visualizes time series data, ***windowed_dataset*** which generated dataset windows, and ***model_forecast*** which used an input model to generate predictions on data windows.

I then split the data into training and testing sets. That training data is in the range of 40% to 90% of the data, and the testing set has the last 20% of the data. I did this for the training set because the first 40% of the data as almost similar to each other and the range of the price for them is 20 – 100, and recent years have prices more than 500, so there is no benefit to train the model on a data its values kind of disappeared. Note that there will be 10% will be overlapping with the training and testing sets, which will not give us a real evaluation number in the evaluation phase.

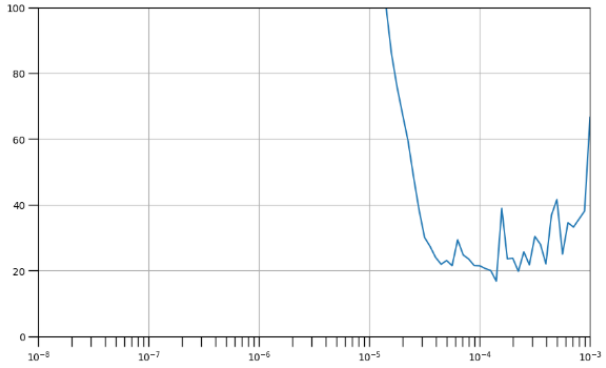
I used the window method because it is a common technique to preprocess time series data for deep learning models.



Train / Test Split for the Data on the DL Model

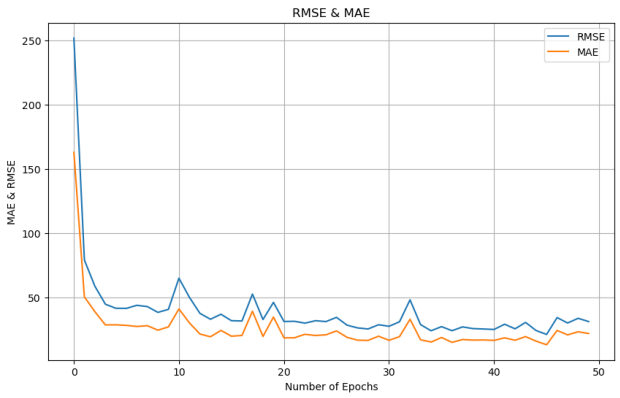
After that, I used a deep learning model with two LSTM layers and multiple Dense layers to train the model on the training set and then tested it on the testing set. I used an LSTM-based model because LSTMs are known to perform well on time series data and can capture the temporal dependencies in the data.

After building the model I trained it on the training data just to pick the best value for the learning rate for this model on this data. This happened with the ***callbacks.LearningRateScheduler***, and then pick the best value of the learning rate and train the model with this value.

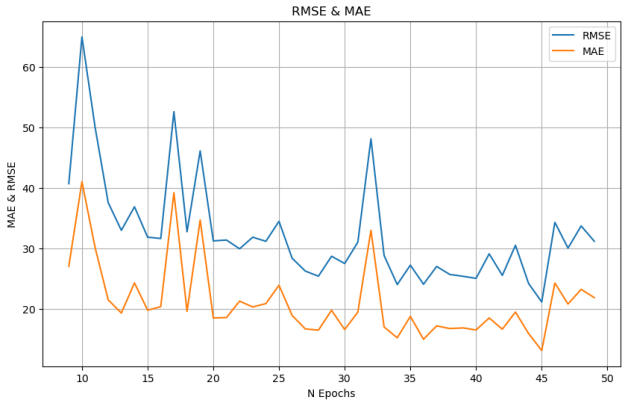


Learning Rate with the Loss

I chose MAE as the loss function, and the RMSE as the metrics. I choose them because they are popular metrics in time-series problems, which can show the model performance in a good way.

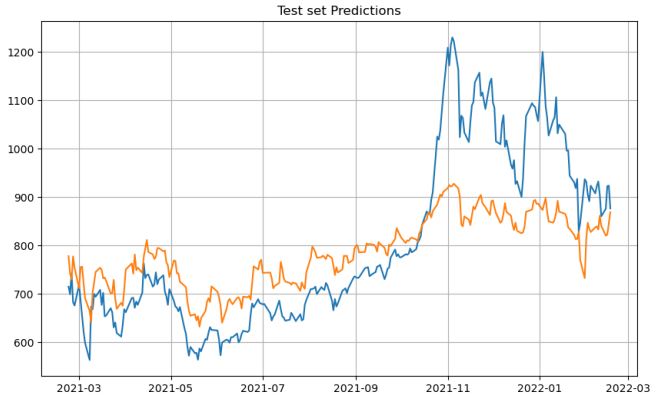


MAE & RMSE while Training

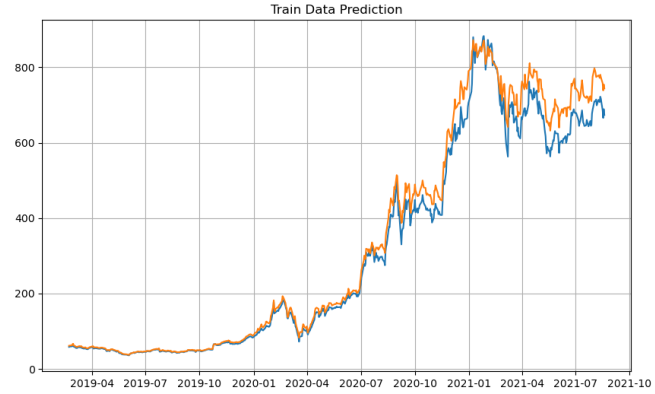


Zoom in to the MAE & RMSE while Training

Then, I made predictions on the train, and the test data, and plot the results.



Model Performance on the Test set



Model Performance on the Training set

Clearly, we can see that the model is overfitted on the training data, I tried the change the model architectures multiple times, but I couldn't prevent the overfitting, I tried to change some values for parameters like window size, batch size, and number of units (neurons) inside the layer, and many other parameters.

Evaluation:

The performance of the models was evaluated on the test dataset using MAE (Mean Absolute Error), and RMSE (Root Mean Square Error) metrics. After applying the evaluation step to our models, I got the following values for the metrics.

Model \ Metrics	MAE	RMSE
Machine Learning	118.24	166.59
Deep Learning	93.81	115.28

As we can see that the deep learning model performed better than the machine learning model, maybe that is because of the complexity of this model, which has multiple LSTMs and Dense layers. The DL model has **150,751 parameters.**

Python Version:

The script was developed using Python ***3.9.13***

How to Run the Scripts:

For both the ML and DL notebooks, *write* ***pip install -r requirements.txt*** in **CMD**, then open them and execute the cells from above to bottom, and if you want to explore different values for the parameters of the models, notice that the prices.txt file should be in the same directory of the notebooks.

NOTE: I only tested it on Windows.

Future Enhancement:

For the machine learning approach, future enhancements could include trying out more regression models and hyperparameter tuning to improve the model's performance. For the deep learning approach, enhancements could include trying out different architectures and hyperparameter tuning to improve the model's performance. Additionally, feature engineering could also be explored to add more relevant features to the dataset.