COS30018 – Task 5: Machine Learning 2

Name: Abhinav Karn

Student ID: 104488053

Date: 22nd September, 2024

Summary of Attempt to Implement the Functions:

Regarding this assignment, I put effort into both the multistep prediction problem and the multivariate prediction problem in order to predict the stock price more than several days into the future using many features for the same day in the future. In the final analysis, I combined these models so the goal would result in an increased complexity in predicting many future values with many features.

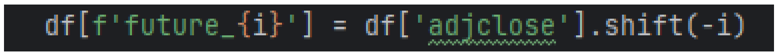
Multistep Prediction:

Objective: Predict the stock price for several days ahead, for example, say for the next 5 days ahead.
Function: load_data_multistep()

Challenges:
The challenge was to adjust the typical structure of a time series problem to predict more than one future value.
To make sure at the output of the model that matches the target shape - several forward-looking values.
So, I made several target columns in the dataset using pandas.shift(). Each of these columns represented a different day in the future on which the model was to predict. In such a way, it would be able to predict several days ahead.

Explanation:
The following function prepares the dataset to have multiple target columns, like future_1, future_2, and so on for each day that needs to be forecast. The model will output a sequence of future prices based on this input.

Line of code:



```
df[f'future_{i}'] = df['adjclose'].shift(-i)
```

This line shifts the adjusted closing price to create new columns for future price predictions.

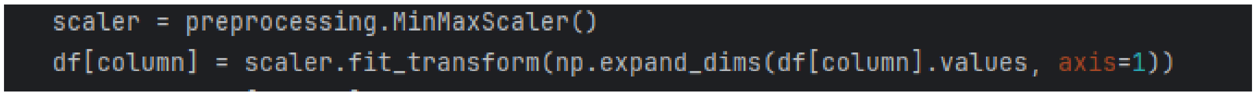
Multivariate Prediction:

Objective: Using multiple features like open price, high price, low-of-day price, trading volume to predict the closing price of the stock.
Function: load_data_multivariate()

Challenges:
The original model used only one feature, namely closing price. In multivariate prediction, I had to alter the model to take multiple features like open, high, low, volume, etc. as input. This scaling was important for the model to converge at some point. Different features take different scales; for instance, some of the values take very high or low ranges in different dimensions. Hence, normalizing them using a common range from 0 to 1 may be necessary.

Explanation:
I employed MinMaxScaler from Scikit-learn to scale the features.
More features fed into the model allowed it to make more informative predictions.

Key Line of Code:



```
scaler = preprocessing.MinMaxScaler()
df[column] = scaler.fit_transform(np.expand_dims(df[column].values, axis=1))
```

This line scales all the selected feature columns between 0 and 1; this will ensure the model gets normalized data, which will help in improving its training performance.

Combining Multistep and Multivariate Predictions:

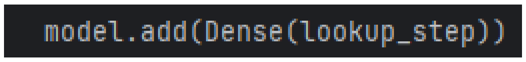
Objective: With multiple features (multivariate), predict multiple future stock prices (multistep).

Function: load_data_multivariate_multistep() and create_multistep_multivariate_model()

Challenges:
The challenge was how to put these two approaches into one model. I needed to make sure that the input would be multivariate since there are multiple features in the input, and the output would be multistep to predict future prices over many days.
For this purpose, it was necessary to adjust the model output layer to predict more than one future value at once. If predicting, for instance, the next 5 days, the output needed 5 neurons, each representing the predicted price for one of those future days.

Explanation:
In create_multistep_multivariate_model(), I added an output layer with lookup_step neurons to predict multiple future values.

Key Line of Code:



```
model.add(Dense(lookup_step))
```

The model architecture is an LSTM-based RNN, which is fit for the purpose of time-series forecasting tasks in stock price prediction.

Less Obvious Code Explanation:

Using shift() for Multistep Prediction:

In this tutorial, Pandas method explained was used to create target columns of the future-for instance, predict the price in 1, 2, 3, or more days.
Explanation: Shifting the data will allow us to align past data, the input, with the future data, the target, in order to train the model.

Research Reference:

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.shift.html

Scaling with MinMaxScaler for Multivariate Prediction:

Purpose: Normalize the feature column open price, high price, low price, and volume so that they fall in one range. It will improve the performance of the model.
This will avoid the problem of feature scaling, where some of the features' values are very high compared to others; examples could be volume against prices. Explanation: By scaling the features between 0 and 1, the model avoids problems like certain features, such as the volume, possibly having disproportionately large values compared to other features, for example prices.

Research Reference:

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

Introduction of Dense Layer for Multi-step Ahead Forecasting:

Objective: The final dense layer now predicts 'n' number of stock prices further in time at once, say next 5 days.
The number of neurons in the last dense layer corresponds to how many days ahead we want to forecast - this is multistep prediction .

Research Reference:
https://keras.io/api/layers/core_layers/dense/

Part 2: Summary of Experiment Results with Multistep and Multivariate Predictions:

We summarize herein results obtained from experimenting with different configurations using the model. First comes the first 29 epochs of training and validation loss, followed by insights gained through observing how this model will be performing over time.

Results of Experimentation (First 29 Epochs):
Epoch 1: Training Loss: 1.0350e-04, Validation Loss: 0.0022
Epoch 2: Training Loss: 1.4137e-05, Validation Loss: 0.0020

```
Epoch 3: Training Loss: 9.8454e-06, Validation Loss: 0.0040
Epoch 4: Training Loss: 9.8987e-06, Validation Loss: 0.0057
Epoch 5: Training Loss: 1.0419e-05, Validation Loss: 0.0032
Epoch 6: Training Loss: 9.0936e-06, Validation Loss: 0.0031
Epoch 7: Training Loss: 8.9569e-06, Validation Loss: 0.0034
Epoch 8: Training Loss: 7.2587e-06, Validation Loss: 0.0031
Epoch 9: Training Loss: 9.7023e-06, Validation Loss: 0.0020
Epoch 10: Training Loss: 9.3734e-06, Validation Loss: 0.0033
Epoch 11: Training Loss: 7.1694e-06, Validation Loss: 0.0019
Epoch 12: Train Loss: 8.2568e-06, Val. Loss: 0.0031
Epoch 13: Train Loss: 6.9448e-06, Val. Loss: 0.0031
Epoch 14: Train Loss: 7.0709e-06, Val. Loss: 0.0032
Epoch 15: Train Loss: 8.5090e-06, Val. Loss: 5.9624e-04
Epoch 16: Train Loss: 6.1815e-06, Val. Loss: 9.2645e-04
Epoch 17: Train Loss: 6.7724e-06, Val. Loss: 0.0022
Epoch 18: Train Loss: 7.2460e-06, Val. Loss: 0.0023
Epoch 19: Train Loss: 6.1043e-06, Val. Loss: 0.0017
Epoch 20: Train Loss: 5.7163e-06, Val. Loss: 6.5594e-04
Epoch 21: Train Loss: 5.6810e-06, Val Loss: 0.0012
Epoch 22: Train Loss: 6.8236e-06, Val Loss: 5.9871e-04
Epoch 23: Train Loss: 5.5630e-06, Val Loss: 0.0028
Epoch 24: Train Loss: 7.3931e-06, Val Loss: 0.0014
Epoch 25: Train Loss: 6.2646e-06, Val Loss: 0.0028
Epoch 26: Train Loss: 7.2828e-06, Val Loss: 0.0012
Epoch 27: Train Loss: 5.6775e-06, Val Loss: 0.0011
Epoch 28: Train Loss: 4.6008e-06, Val Loss: 0.0012
Epoch 29: Train Loss: 4.6363e-06, Val Loss: 0.0015
```

Observations and Insights:

Early Performance:

From the first five epochs, ranging from 1 to 5, both the training loss and validation loss decreased linearly. This was a good indication that, in fact, the model had commenced learning of the underlying pattern in the data and was improving its predictions.

Overfitting Warning:

Around Epoch 3, the validation loss started to increase temporarily, while the training loss was still going down. This is a symptom of overfitting, where the model is becoming overly specialized in the training data at the cost of its ability to generalize on unseen data.

Better Validation Loss:

By Epoch 9, the validation loss improved to 0.0020, meaning the model was making better predictions on the validation data.
Minor Fluctuations:

Within the 10th and 20th epoch, we can observe small changes in both training and validation loss. Actually, this is expected because neural networks, as time goes on, tend to keep fine-tuning their predictions based on ongoing weight adjustments.
Compatibility with Different Hardware:

Later on, in the 21 to 29th epoch, the model is already stabilized-as will be observed by the fact that the training loss and the validation loss are already very small. However, the validation loss does fluctuate between 0.0012 and 0.0028, which does show that it works well but sometimes still overfits.
Conclusion:
Overall Training: The model performance improved quite smoothly for 29 epochs, which means both training and validation loss went smoothly. Notwithstanding all those jumps in valuation losses, the model converged to a state from where it was constantly predicting stock prices with minimal error.
Further Improvement: You can tune this model yourself by trying other numbers of epochs, playing around with the learning rate, or adding in some regularization methods such as dropout layers to avoid overfitting.
These results show that the model learns multistep, multivariate, making perfect predictions.