Student Name : Matheesha Ileperuma  
Student ID : 104659053

C2 Stock Prediction System – Solo Report

Summary

For Task C.2, I worked on analyzing, understanding, and extending the stock prediction system provided in the original stock.py (v0.1) script. The original version laid the foundation for downloading stock data and using an LSTM model for price prediction, but it lacked several important elements like proper data preprocessing, sequence generation, model evaluation, and missing value handling. These were crucial to make the prediction pipeline more reliable and modular.

Explanation of Additions and Key Concepts I Had to Learn

Here are a few areas where I had to put in extra effort to understand and implement:

* Handling NaNs in the dataset: I realized that some downloaded stock data might contain NaN values. These were removed using dropna() before scaling.
* Creating sequences for time-series: This was tricky at first. I had to figure out how to generate sliding windows of historical stock prices and match each sequence with the appropriate label for training. I added comments in create\_sequences() to reflect my understanding.
* Splitting the dataset: I created a split\_data() function that can split data either by time (chronologically) or randomly. This was important to avoid leakage during training.
* Saving and loading models and scalers: I added logic to save the trained model (.h5) and scaler (.pkl) so they can be reused during evaluation instead of retraining from scratch.

# Overview of Major Code Files

1. dProcess.py

* This file handles all preprocessing tasks:
* Downloads data via Yahoo Finance (yf.download).
* Scales it using MinMaxScaler.
* Saves the processed data and scaler locally.
* Generates sequences using a fixed sequence\_length (50 steps by default).

1. train.py

* Loads processed data.
* Splits into training/testing sets.
* Builds an LSTM model and trains it.
* Saves the trained model to models/.

1. evaluate.py

* Loads test data and the saved model.
* Recreates test sequences.
* Uses the model to make predictions.
* Plots the prediction vs. actual prices.

# Improvement from the original code

* The original stock.py file provided a solid foundation for this project but lacked some key aspects that are essential for scalability, reusability, and clarity. To enhance its functionality and align it with the expectations of a modular and maintainable machine learning project, I made several important additions and changes.
* Firstly, NaN handling was completely missing in the original script. I addressed this by introducing the dropna() method to clean the dataset before any processing occurred. This ensured we didn’t train the model on incomplete or corrupted data.
* Secondly, scaling of features and saving the scaler were not implemented at all. I added these steps so that the same transformation used during training could be consistently applied during evaluation and future predictions. This also improves model reliability.
* The original code also lacked a formal train/test split function — everything was done on a single dataset. I introduced a proper data-splitting function which allows us to specify how we want to divide the data, either by date or randomly, ensuring a more realistic and testable model evaluation.
* Sequence generation in the original was done manually and embedded within the logic. I replaced that with a reusable function called create\_sequences() and included detailed comments explaining how sequences and labels are formed from historical stock data.
* There was also no mechanism to save or reload models in the original script. I added functionality to save models as .h5 files and scalers as .pkl files, which makes it easier to reuse trained models or perform inference without retraining.
* For visualisation, the original code had very basic graph output. I enhanced it by integrating matplotlib to generate clear and labeled comparison graphs between actual and predicted values, which helps in interpreting model performance.
* Lastly, I split the entire logic into multiple separate modules instead of keeping everything in one script. This modular approach improves readability, debugging, and collaboration, as each script now has a single responsibility (e.g., training, preprocessing, evaluation).

# Conclusion

This task taught me a lot about structuring machine learning projects. I was able to debug multiple issues, like sequence shape mismatches, IndexError from wrong loop ranges, and scaler-related bugs. Most importantly, I learned how to interpret older code written by someone else and improve it while staying consistent