COS30018 – Intelligent System

Task B.6 Report

# ARIMA and Ensemble Methods

ARIMA is a machine learning model specifically developed for forecasting time series data, which makes it suitable for stock prediction applications. However, its implementation and functionality differ from those of an LSTM model. The initial step involves importing the ARIMA library into the program. This also requires the installation of the `statsmodels` package using pip, as well as adding this package to the `requirements.txt` file.

from statsmodels.tsa.arima.model import ARIMA

My initial effort to implement a predictive model involved developing a new ARIMA model utilizing the test dataset and attempting to execute the `forecast()` method, specifying the length of the test data as the number of steps. However, this approach did not yield successful results, despite my investigations into the underlying issues and attempts to rectify them. Ultimately, I discovered that the `forecast()` method tends to perform more reliably with smaller step counts, prompting me to devise a new strategy for generating predictions.

To enable predictions across the entire test dataset, I needed to establish a loop that iterated through the length of the test data, effectively creating and fitting a new ARIMA model during each iteration. In each cycle, the model would be generated using the training data, supplemented by the actual value from the previous iteration's test data. The forecast would then be executed for the subsequent day, with the result incorporated into the final prediction array before advancing to the next loop iteration.

predictions = list()

for t in range(len(test\_data)):

    arima\_model = ARIMA(history, order=(5,1,0))

    model\_fit = arima\_model.fit()

    output = model\_fit.forecast()

    forcast = output[0]

    predictions.append(forcast)

    obs = test\_data[t]

    history.append(obs)

    print('%f/%f, predicted=%f, expected=%f' % (t,len(test\_data), forcast, obs))

To then create an ensemble prediction using the average between the ARIMA and LSTM results, firstly the arima results needed to be inverse scaled to match the actual results, and then reshaped to the same shape as the LSTM prediction. Then a simple averaging calculation is run on the two prediction arrays to create the ensemble prediction.

arima\_predictions\_scaled = data["column\_scaler"][prediction\_column].inverse\_transform(np.array(predictions).reshape(-1,1)).reshape(-1)

ensemble\_prediction = (predicted\_close\_prices + arima\_predictions\_scaled) / 2

To obtain predictions for the upcoming k days, it was straightforward to create a new ARIMA model. This involved fitting the model and then utilizing the forecast function to specify the desired number of future steps. Similarly, the ensemble prediction can be computed using the same methodology as previously described.

arima\_model = ARIMA(data['train\_data'][prediction\_column], order=(5,1,0))

model\_fit = arima\_model.fit()

arima\_prediction = model\_fit.forecast(steps=N\_STEPS)

#arima\_prediction = arima\_prediction[0]

arima\_prediction = data["column\_scaler"][prediction\_column].inverse\_transform(np.array(arima\_prediction).reshape(-1,1)).ravel()

ensemble\_prediction = (prediction + arima\_prediction) / 2

# Loop over the prediction and print each day's predicted price

for i, price in enumerate(prediction):

    print(f"LTSM Prediction for day {i+1}: {price}")

for i, price in enumerate(arima\_prediction):

    print(f"ARIMA Prediction for day {i+1}: {price}")

for i, price in enumerate(ensemble\_prediction):

    print(f"Ensemble Prediction for day {i+1}: {price}")

# Random Forrest Ensemble

To explore ensemble predictions using different model configurations, I incorporated a Random Forest model for additional predictions. This process entailed establishing a new Random Forest model with 300 estimators (or "trees") and setting the random state to 42. Prior to fitting the model, the data was adjusted to conform to the required input shape. The ensemble prediction was subsequently computed using the same methodology as previously outlined.

# Create a Random Forest Regressor

rf = RandomForestRegressor(n\_estimators=300, random\_state=42)

#X\_train\_reshaped =  data["X\_train"][:, :, closing\_price\_index]

X\_train\_flattened = data["X\_train"][:, :, closing\_price\_index].reshape(data["X\_train"].shape[0], -1)

#y\_train\_reshaped = data["y\_train"].reshape(-1, 1)  # shape will be (1507, 100\*6)

# Fit the model on your training data

y\_train\_reshaped = data["y\_train"][:, :, closing\_price\_index]  # shape will be (1507, 5)

print(X\_train\_flattened.shape)

print(y\_train\_reshaped.shape)

rf.fit(X\_train\_flattened, y\_train\_reshaped)

# Make predictions on the test data

X\_test\_2d = data["X\_test"].reshape((data["X\_test"].shape[0], -1))

rf\_predictions = rf.predict(data["X\_test"][:, :, closing\_price\_index])

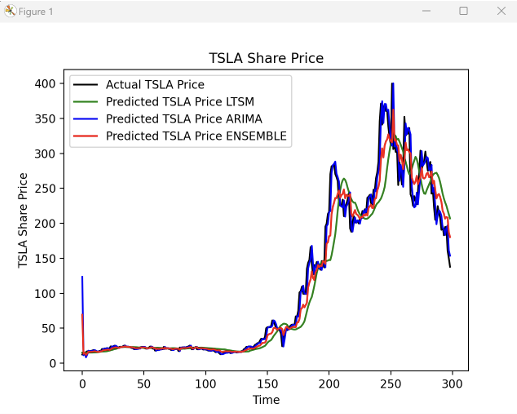
rf\_predictions = data["column\_scaler"][prediction\_column].inverse\_transform(np.array(rf\_predictions).reshape(-1,1)).reshape(-1)

rf\_predictions = rf\_predictions[-len(predicted\_close\_prices):]

ensemble\_prediction = (predicted\_close\_prices + rf\_predictions) / 2

# Result

As illustrated by the results presented below, the ARIMA forecasting method demonstrates a high level of accuracy. However, this precision may stem from its approach of incorporating the actual test value in each iteration, rather than solely relying on prior predictions. Conversely, the LSTM model's lower accuracy adversely affects the reliability of the ensemble prediction that follows. This trend is further evident in the subsequent predictions for the upcoming days.



A screenshot of a computer

Description automatically generated

The results obtained from the random forest indicate that there may have been an error in either the model itself or the data preparation process, as the output appears to be entirely inaccurate. However, it does illustrate that an ensemble average can indeed be calculated from various configurations and models.

A graph of different colored lines

Description automatically generated