**Task 5: Machine Learning 2**

**Muhammad Ali, 103960437, COS30018, 08/09/2024, Tutor: Ru Jia.**

**Report on Implementing Multivariate and Multistep Stock Price Prediction**

**Implementation Summary**

**1. Multistep Prediction Function**

To enable the model to predict multiple future time steps, we needed to adjust how the data is prepared and how the model outputs predictions.

**Function: prepare\_multistep\_data**

We introduced a new function, prepare\_multistep\_data, in data\_processing.py:

def prepare\_multistep\_data(X, y, timesteps, future\_steps):  
 *"""  
 Prepares data for time series models for multistep predictions.  
  
 Parameters:  
 - X (np.ndarray): Input feature matrix.  
 - y (np.ndarray): Target vector.  
 - timesteps (int): Number of timesteps for each sample.  
 - future\_steps (int): Number of future steps to predict.  
  
 Returns:  
 - X\_reshaped (np.ndarray): Reshaped input data for the model.  
 - y\_reshaped (np.ndarray): Corresponding reshaped target data (multistep).  
 """* n\_samples = X.shape[0] - timesteps - future\_steps + 1  
  
 X\_reshaped = np.array([X[i:i + timesteps] for i in range(n\_samples)])  
 y\_reshaped = np.array([y[i + timesteps:i + timesteps + future\_steps] for i in range(n\_samples)])  
  
 return X\_reshaped, y\_reshaped

**Explanation:**

* **Calculating n\_samples:** We calculate the number of samples considering both the input timesteps and the future steps we want to predict.
* **Creating X\_reshaped:** For each sample, we take a window of timesteps from X.
* **Creating y\_reshaped:** For the corresponding target values, we take the next future\_steps from y after the current window.

**Research Reference:**

* *Stack Overflow discussion on preparing data for multistep time series forecasting*: [Link](https://stackoverflow.com/questions/42992889/how-to-prepare-multistep-time-series-data-for-lstm-in-keras)

**2. Multivariate Prediction Function**

The original program used only the 'Close' price as the feature. To implement multivariate prediction, we modified the data loading function to include all relevant features.

**Modification in load\_and\_process\_data\_with\_gap function:**

# Determine features and target  
if feature\_columns is None:  
 feature\_columns = df.columns.tolist()  
 # Include 'Close' as a feature if desired  
 # If you want to exclude the target column from features, uncomment the next line  
 # feature\_columns.remove(target\_column)  
  
x = df[feature\_columns].values  
y = df[target\_column].values

**Explanation:**

* **Including Multiple Features:** By default, we set feature\_columns to include all columns from the DataFrame.
* **Target Column:** We specify the target column (defaulting to 'Close') to use for predictions.

**3. Combining Multivariate and Multistep Predictions**

To address both multivariate inputs and multistep outputs, we adjusted the data scaling, model creation, training, and prediction processes.

**Scaling Functions for Multistep Data**

We needed to ensure that scaling was correctly applied to multidimensional data.

def scale\_X(X\_train, X\_test):  
 n\_samples\_train, timesteps, n\_features = X\_train.shape  
 n\_samples\_test = X\_test.shape[0]  
 X\_train\_flat = X\_train.reshape(-1, n\_features)  
 X\_test\_flat = X\_test.reshape(-1, n\_features)  
 scaler = MinMaxScaler()  
 X\_train\_scaled = scaler.fit\_transform(X\_train\_flat).reshape(n\_samples\_train, timesteps, n\_features)  
 X\_test\_scaled = scaler.transform(X\_test\_flat).reshape(n\_samples\_test, timesteps, n\_features)  
 return X\_train\_scaled, X\_test\_scaled, scaler  
  
def scale\_y(y\_train, y\_test):  
 n\_samples\_train, future\_steps = y\_train.shape  
 n\_samples\_test = y\_test.shape[0]  
 y\_train\_flat = y\_train.reshape(-1, 1)  
 y\_test\_flat = y\_test.reshape(-1, 1)  
 scaler = MinMaxScaler()  
 y\_train\_scaled = scaler.fit\_transform(y\_train\_flat).reshape(n\_samples\_train, future\_steps)  
 y\_test\_scaled = scaler.transform(y\_test\_flat).reshape(n\_samples\_test, future\_steps)  
 return y\_train\_scaled, y\_test\_scaled, scaler

**Explanation:**

* **Flattening and Reshaping:** We flatten the data before scaling and then reshape it back to its original dimensions.
* **Ensuring Correct Dimensions:** Careful attention was needed to maintain the correct shapes during scaling, especially when dealing with 3D arrays (samples, timesteps, features).

**Research Reference:**

* *Scaling multivariate time series data for LSTM networks*: Machine Learning Mastery

**Adjusting the Model Architecture**

We updated the model to produce multiple outputs corresponding to future steps.

output\_size = FUTURE\_STEPS # Set output size to number of future steps

input\_shape = (PREDICTION\_DAYS, X\_train\_scaled.shape[2])  
layer\_types = ['LSTM', 'GRU', 'Dense']  
layer\_sizes = [150, 100, 50]  
dropout\_rates = [0.3, 0.3, 0.2]  
output\_size = FUTURE\_STEPS  
return\_sequences = [True, False, False]  
activation\_functions = ['tanh', 'tanh', 'relu']

**Explanation:**

* **Output Layer:** The output layer now has FUTURE\_STEPS units to predict multiple future time steps.

**Modifying Prediction and Evaluation**

We adjusted the prediction and evaluation code to handle multistep outputs.

# Predict prices using the test set  
predicted\_prices = model.predict(X\_test\_scaled)

# Reshape to (-1, 1) for inverse transform  
predicted\_prices\_flat = predicted\_prices.reshape(-1, 1)  
actual\_prices\_flat = y\_test\_scaled.reshape(-1, 1)

# Inverse transform  
predicted\_prices\_inv = y\_scaler.inverse\_transform(predicted\_prices\_flat).reshape(-1, FUTURE\_STEPS)  
actual\_prices\_inv = y\_scaler.inverse\_transform(actual\_prices\_flat).reshape(-1, FUTURE\_STEPS)

# Flatten for evaluation  
predicted\_prices\_flat = predicted\_prices\_inv.flatten()  
actual\_prices\_flat = actual\_prices\_inv.flatten()

**Explanation:**

* **Flattening Multistep Outputs:** We flatten the multistep outputs to compute evaluation metrics across all predicted values.
* **Inverse Transformation:** We apply the inverse scaling to bring predictions back to the original scale.

**Explanation of Less Straightforward Code Segments**

**1. Calculating the Number of Samples in prepare\_multistep\_data**

n\_samples = X.shape[0] - timesteps - future\_steps + 1

**Explanation:**

* **Understanding Time Steps:** When preparing data for time series forecasting, each sample consists of timesteps input observations followed by future\_steps target observations.
* **Adjusting for Future Steps:** We subtract both timesteps and future\_steps from the total number of observations to avoid indexing beyond the array bounds.

**Research Reference:**

* *Understanding time series windowing for LSTM input*: Towards Data Science

**2. Reshaping and Scaling Multidimensional Arrays**

X\_train\_flat = X\_train.reshape(-1, n\_features)

X\_train\_scaled = scaler.fit\_transform(X\_train\_flat).reshape(n\_samples\_train, timesteps, n\_features)

**Explanation:**

* **Flattening for Scaling:** MinMaxScaler operates on 2D arrays, so we flatten the 3D array (samples, timesteps, features) into a 2D array (total\_samples, features).
* **Reshaping Back:** After scaling, we reshape the data back to its original 3D shape for input into the LSTM model.

**Research Reference:**

* *Scaling 3D arrays for LSTM input*: [Stack Overflow](https://stackoverflow.com/questions/50125844/reshape-3d-array-to-2d-array-in-python)

**3. Handling Multistep Predictions in Evaluation**

# Flatten for evaluation  
predicted\_prices\_flat = predicted\_prices\_inv.flatten()  
actual\_prices\_flat = actual\_prices\_inv.flatten()

mae = mean\_absolute\_error(actual\_prices\_flat, predicted\_prices\_flat)

**Explanation:**

* **Flattening Multistep Predictions:** Evaluation metrics like MAE and MSE require 1D arrays. By flattening the predictions and actual values, we compute the error across all future steps collectively.
* **Comparative Analysis:** This approach allows us to assess the overall performance of the model across all predicted time steps.

**Experimental Results**

**Data and Experimental Setup**

* **Stock Ticker:** KO (The Coca-Cola Company)
* **Training Period:** August 1, 2016, to August 31, 2024
* **Prediction Days (Input Timesteps):** 60
* **Future Steps (Output Timesteps):** 3
* **Features Used:** Open, High, Low, Close, Adj Close, Volume

**Model Architecture**

* **Layer Types:** ['LSTM', 'GRU', 'Dense']
* **Layer Sizes:** [150, 100, 50]
* **Dropout Rates:** [0.3, 0.3, 0.2]
* **Activation Functions:** ['tanh', 'tanh', 'relu']
* **Loss Function:** Huber Loss
* **Optimizer:** Adam

**Training Performance**

* **Epochs:** Up to 100 (Early stopping applied)
* **Batch Size:** 32
* **Validation Split:** 20%

**Training and Validation Loss Plot:**

A graph with blue line and orange line

Description automatically generated

**Evaluation Metrics**

**Mean Absolute Error (MAE): 0.89**

**Mean Squared Error (MSE): 1.19**

**Root Mean Squared Error (RMSE): 1.09**

**Prediction Results**

**Plot of Actual vs. Predicted Prices:**

A graph showing a price prediction

Description automatically generated with medium confidence

**Next Days Prediction**

* **Predicted Closing Prices for the Next 3 Days:**

Next 3 Days Prediction: [71.125595 70.98358 70.88369 ]

**Conclusion**

The implementation of multivariate and multistep prediction functions successfully enhanced the stock price prediction model. By incorporating multiple features and predicting multiple future time steps, the model provides a more comprehensive forecasting tool. The experimental results demonstrate that the model can effectively learn from historical data to predict future stock prices with reasonable accuracy.

**Key Takeaways:**

* **Data Preparation is Crucial:** Properly reshaping and scaling the data is essential when dealing with multivariate and multistep time series data.
* **Model Complexity:** Adding more features and predicting multiple steps increases the model's complexity, necessitating careful tuning and validation to prevent overfitting.
* **Evaluation Metrics:** Using appropriate evaluation metrics and visualization helps in assessing the model's performance comprehensively.