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

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

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

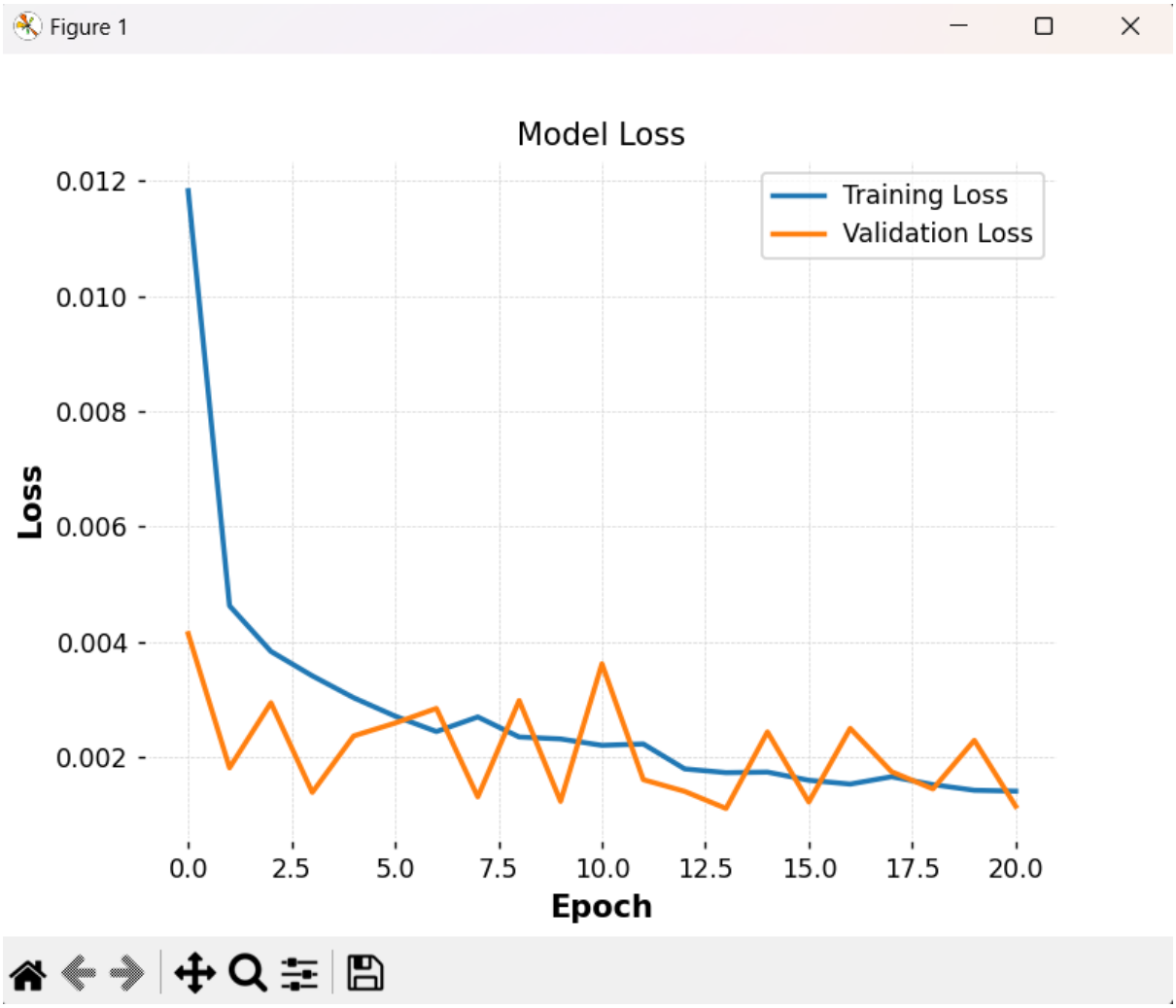
- **Layer Types:** ['LSTM', 'GRU', 'Dense']
- **Layer Sizes:** [150, 100, 50]
- **Dropout Rates:** [0.3, 0.3, 0.2]
- **Activation Functions:** ['tanh', 'tanh', 'relu']
- **Loss Function:** Huber Loss
- **Optimizer:** Adam

Training Performance

- **Epochs:** Up to 100 (Early stopping applied)

- **Batch Size:** 32
- **Validation Split:** 20%

Training and Validation Loss Plot:



Evaluation Metrics

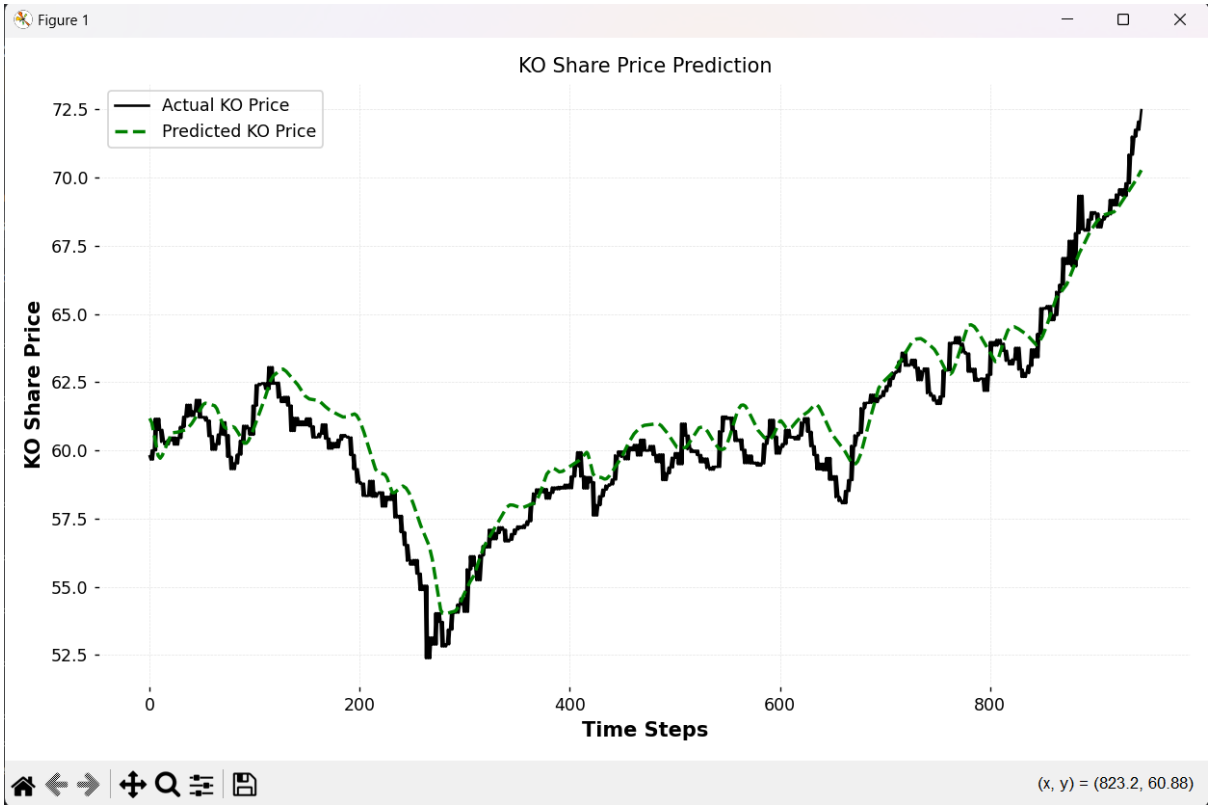
Mean Absolute Error (MAE): 0.89

Mean Squared Error (MSE): 1.19

Root Mean Squared Error (RMSE): 1.09

Prediction Results

Plot of Actual vs. Predicted Prices:



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.