




COS30049 - Intelligent System

MACHINE LEARNING

TRAN DUC ANH DANG

Student ID: 103995439



Requirements	2
Virtual Environment	2
Dependencies	2
Installation	2
Anaconda/Virtual Environment	2
Dependencies	2
Machine Learning 1	3
create_dynamic_model	3
plot_metric	4
Model's Performance	5
Layer Configs 1 - Mix LSTM, GRU, RNN, Different Activations	5
Layer Configs 2 - Simple LSTM Model	7
Layer Configs 3 - Simple GRU Model (Most Accurate)	9
Layer Configs 4 - More complex GRU Model	11
Layer Configs 5 - Mixed LSTM and GRU Model	13
Layer Configs 6 - Deep LSTM with more Units Model	15
Layer Configs 7 - LSTM Reduced Dropout Model	17
Layer Configs 8 - Complex Bidirectional LSTM Configuration Model	19
Layer Configs 9 - Complex Bidirectional GRU Configuration Model	21
Layer Configs 10 - Complex Bidirectional GRU Configuration - More Layers Model	23
Conclusion:	24

Requirements

Virtual Environment

- Google Colab
- Jupyter Notebook

Dependencies

- numpy
- matplotlib
- mplfinance
- pandas
- scikit-learn
- pandas-datareader
- yfinance
- pandas_ta
- joblib

Installation

*Note: Anaconda is required unless Google Colab is being used

Anaconda/Virtual Environment

1. Download Anaconda: Go to the Anaconda website (<https://www.anaconda.com/products/distribution>) and download the appropriate version for your operating system.
2. Install Anaconda: Follow the installation instructions for the OS from the Anaconda website.
3. Open Anaconda Navigator: Launch Anaconda Navigator from your installed applications.
4. Create a New Environment (Required): Create a new environment to isolate Jupyter installation on each project. Click on "Environments" in Navigator and then "Create" to make a new environment.
5. Install Jupyter Notebook: In the selected environment, click on the environment name and select "Open Terminal". In the terminal, type: `conda install jupyter`.

Dependencies

In Google Colab or Jupyter Notebook, it can directly install the required dependencies using the `!pip` command in code cells. Here's an example of how to install the dependencies:

`!pip install <package> or !pip install -r <text file>`

Machine Learning 2

create_predict_datasets

```
def create_predict_datasets(start_predict, end_predict, tick, k):

    # Download or Load Raw Data
    print(f"Fetching data from {start_predict} to {end_predict}")
    data = load_data(start_predict, end_predict, tick)

    print(f"Raw data shape: {data.shape}")
    print(data.head())

    # Data Validation
    df = data_validation(start_predict, end_predict, tick)

    print(f"Processed data shape: {df.shape}")
    print(df.head())

    # Define features and target
    feature_columns = ['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume', 'RSI', 'EMAF', 'EMAM', 'EMAS']
    target_column = 'TargetNextClose'

    # Preparing Datasets
    # Scaler for features
    scaled_data, train_feature_scaler = scaler_features(df[feature_columns])
    print("Scaled data shape:", scaled_data.shape)

    # Scaler for target
    scaled_target_train, scaler = scaler_features(df[target_column].values.reshape(-1, 1))

    x_test, y_test = [], []
    for i in range(step_size, len(scaled_data)):
        x_test.append(scaled_data[i-step_size:i])
        y_test.append(scaled_target_train[i])

    x_test, y_test = np.array(x_test), np.array(y_test)
    print("x_test shape:", x_test.shape)

    # For data
    if not isinstance(data.index, pd.DatetimeIndex):
        if "Date" in data.columns:
            data['Date'] = pd.to_datetime(data['Date'])
            data.set_index('Date', inplace=True)

    # For df
    if not isinstance(df.index, pd.DatetimeIndex):
        if "Date" in df.columns:
            df['Date'] = pd.to_datetime(df['Date'])
            df.set_index('Date', inplace=True)

    return df, scaled_data, scaler, x_test, y_test
```

The function `create_predict_datasets` takes the following parameters:

1. `start_predict`: A string representing the starting date for fetching the stock data in the format 'YYYY-MM-DD'.
2. `end_predict`: A string representing the ending date for fetching the stock data in the format 'YYYY-MM-DD'.
3. `tick`: A string representing the stock ticker symbol for which the data is to be fetched.
4. `k`: An integer parameter (the significance of which is not described in the provided context).

This function performs the following operations:

1. Data Fetching: Retrieves stock data for the specified `tick` between `start_predict` and `end_predict` dates using the `load_data` function.
2. Data Validation: Applies data validation and preprocessing using the `data_validation` function. This includes feature engineering and data transformation tasks.

3. Feature and Target Definition: Specifies the features and target columns for prediction. The features include stock data attributes like 'Open', 'High', 'Low', etc., and the target is the 'TargetNextClose' column.
4. Scaling Data: Scales the feature data to bring it within a normalized range, typically [0,1], using the `scaler_features` function. This ensures better performance during the prediction phase.
5. Prepare Test Data: Constructs the test datasets by segmenting the scaled data based on the `step_size` and the length of the scaled data.
6. Datetime Index Check: Ensures the dataframes `data` and `df` have a datetime index. If not, it converts the 'Date' column to a datetime index.
7. Return: Returns the processed dataframe `df`, the scaled feature data `scaled_data`, the scaler object `scaler` for inverse transformations, and the test datasets `x_test` and `y_test`.

single_day_multivariate_prediction

```
def single_day_multivariate_prediction(model, tick, predict_date):
    # Convert the string date to a datetime object
    predict_date = datetime.strptime(predict_date, '%Y-%m-%d')
    start_date = predict_date - timedelta(days=730)

    print(f"Date: {start_date.strftime('%Y-%m-%d')} to {predict_date.strftime('%Y-%m-%d')}")

    # Create a dataset from the start date to the day before the specified prediction date
    _, scaled_data, scaler, x_test, _ = create_predict_datasets(start_date.strftime('%Y-%m-%d'), predict_date.strftime('%Y-%m-%d'))

    # Check if there's enough data for prediction
    if len(x_test) == 0 or x_test[-1].shape[0] == 0:
        raise ValueError("Insufficient data for the specified prediction date.")

    # Use the model to make a prediction using the last day's multivariate data
    prediction = model.predict(x_test[-1].reshape(1, -1, x_test.shape[-1]))

    # Inversely scale the prediction to get the actual predicted price
    predicted_price = scaler.inverse_transform(prediction)

    print(f"Date: {predict_date.strftime('%Y-%m-%d')}, Predicted Price: {predicted_price[0][0]}")

    return predicted_price
```

The function `single_day_multivariate_prediction` takes the following parameters:

1. `model`: A pre-trained machine learning or deep learning model that will be used for prediction.
2. `tick`: A string representing the stock ticker symbol for which the prediction is to be made.
3. `predict_date`: A string representing the date for which the prediction is to be made in the format 'YYYY-MM-DD'.

This function performs the following operations:

1. Date Conversion: Converts the `predict_date` string into a datetime object for further operations.
2. Lookback Calculation: Determines the starting date (2 years prior to `predict_date`) for fetching historical stock data.
3. Data Preparation: Calls the `create_predict_datasets` function to fetch and prepare the necessary datasets for prediction. The data fetched starts from the calculated lookback date and ends a day before the specified `predict_date`.

4. Data Availability Check: Verifies if there's sufficient data available to make a prediction. If not, a `ValueError` is raised.
5. Prediction: Uses the provided `model` to make a prediction using the last day's multivariate data fetched.
6. Inverse Scaling: The raw prediction output from the model is scaled back to its original range using the inverse transformation process to get the actual predicted price.
7. Output: The function returns the predicted closing price for the specified `predict_date`.

predictions

```
def predictions(model, tick, start_predict, end_predict, k=10):

    print(f'Date: {start_predict} to {end_predict}')

    # Preparing Datasets
    predict_df, scaled_data, scaler, x_test, y_test = create_predict_datasets(start_predict, end_predict, tick, k)

    # Actual Prices
    actual_prices = predict_df['Close'].values

    # Past predictions
    past_predictions = model.predict(x_test)
    past_predictions = scaler.inverse_transform(past_predictions)

    # Convert to numpy
    past_predictions = np.array(past_predictions)

    # Placeholder for future predictions of company
    future_days = k
    future_predictions = []

    input_data = x_test[-1]

    # Getting the last known date from the dataset and generating future dates
    last_known_date = datetime.strptime(end_predict, '%Y-%m-%d')
    future_dates = [last_known_date + timedelta(days=i) for i in range(1, future_days + 1)]

    for i in range(future_days):
        pred = model.predict(input_data.reshape(1, -1, x_test.shape[-1]))

        # Inversely scaling
        predicted_price = scaler.inverse_transform(pred)

        future_predictions.append(predicted_price)

    # Print predictions
    current_date = future_dates[i]
    print(f'Date: {current_date.strftime('%Y-%m-%d')}, Predicted Price: {predicted_price}')

    # Updating the last element with the new prediction
    input_data = np.roll(input_data, -1, axis=0)
    input_data[-1] = pred

    # Convert to numpy
    future_predictions = np.array(future_predictions)

    return predict_df, actual_prices, past_predictions, future_predictions
```

The function `predictions` takes the following parameters:

1. `model`: A pre-trained machine learning model used for making predictions.
2. `tick`: A string representing the ticker symbol of the company's stock.
3. `start_predict`: A string representing the starting date for the dataset.

4. `end_predict`: A string representing the ending date for the dataset.
5. `k`: An integer representing the number of days into the future for which predictions need to be made (default value is 10).

This function performs the following operations:

1. Prepare Datasets: Calls `create_predict_datasets` to prepare the required datasets for prediction.
2. Past Predictions: Uses the model to predict past closing prices based on the test dataset.
3. Future Predictions: Predicts the closing prices for the next (k) days using the model.
4. Date Handling: Generates future dates based on the last known date in the dataset.
5. Prediction Printing: Prints each future prediction along with its corresponding date.

Conclusion

1. Multistep Prediction Problem:
 - The function `predictions` attempts to address the multistep prediction problem. Specifically, it aims to forecast the closing prices of a stock for (k) days into the future. It uses a loop to generate predictions for each day in the future, updating its input data with the new prediction at each iteration.
2. Simple Multivariate Prediction Problem:
 - The function `single_day_multivariate_prediction` is designed to solve the simple multivariate prediction problem. It takes into account multiple features of a company such as opening price, highest price, lowest price, closing price, adjusted closing price, and trading volume for predicting the closing price of the company for a specified day in the future. It uses the `create_predict_datasets` function to fetch and prepare the required multivariate data.
3. Multivariate, Multistep Prediction Problem:
 - The `predictions` function effectively combines the capabilities of the previous two functions to solve the multivariate, multistep prediction problem. It not only considers multiple features for prediction but also forecasts the closing prices for multiple future days.
4. Data Preparation and Validation:
 - The `create_predict_datasets` function is responsible for data fetching, validation, and preparation. It defines the set of features to consider and scales them appropriately. It also ensures that the dataset indices are of the type `DatetimeIndex`, which is essential for time series analysis.

In conclusion, based on the provided functions and their descriptions, I have made significant efforts to address the three main requirements:

1. Solving the multistep prediction problem.
2. Solving the simple multivariate prediction problem.

3. Combining the solutions of the above two problems to address the multivariate, multistep prediction challenge.

I have met the stated requirements. However, the effectiveness of the solutions would ultimately be determined by evaluating the model's predictions against actual stock prices and assessing its accuracy in real-world scenarios.