Stock Price Prediction using LSTM

by Lokesh senapathi , gitam university vishakapatnmam

Overview

This project aims to predict stock prices using a Long Short-Term Memory (LSTM) model. It utilizes historical stock price data for training and generates predictions for future stock prices.

This project utilizes a Long Short-Term Memory (LSTM) model to predict stock prices. It requires the following dependencies: numpy, matplotlib, pandas, scikit-learn, and Keras.

The code reads training and testing data from CSV files. The training data is preprocessed by scaling it using MinMaxScaler and reshaping it to match the LSTM input shape.

The LSTM model is built using Keras, with the specified number of units and activation function. It is compiled with the Adam optimizer and mean squared error loss.

The model is trained on the training data, and then used to predict stock prices for the testing data. The predictions are inverse transformed to obtain the actual stock prices.

The real and predicted stock prices are visualized using matplotlib. The Root Mean Squared Error (RMSE) is calculated to evaluate the model's performance.

Additionally, the code also predicts stock prices for the training data and visualizes the real and predicted stock prices for analysis purposes.

It's important to customize the file paths and adjust hyperparameters as needed for your specific project.

Requirements

To run this code, you need the following dependencies:

- numpy
- matplotlib
- pandas
- scikit-learn
- Keras

You can install these dependencies using **pip** or any other package manager.

Data

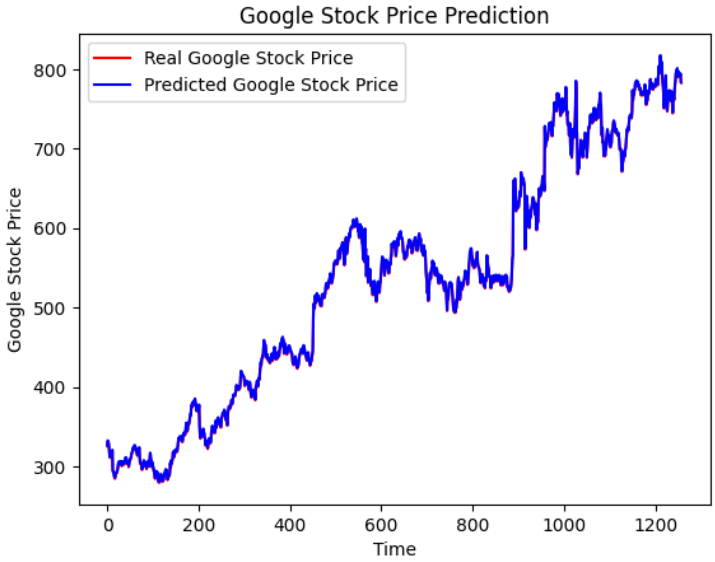
The project uses two CSV files for training and testing:

- **Google_Stock_Price_Train.csv**: Contains historical stock price data for training the model.
- **Google_Stock_Price_Test.csv**: Contains stock price data for testing the model's predictions.

Make sure to provide the correct file paths in the code.

Implementation

1. Import the necessary libraries:
    - **numpy** for numerical operations.
    - **matplotlib** for data visualization.
    - **pandas** for data manipulation.
2. Load the training data:
    - Use **pd.read_csv()** to load the training data from the CSV file.
    - Extract the relevant column containing the stock prices.
    - Convert the data to a numpy array for further processing.
3. Data Preprocessing:
    - Scale the data using **MinMaxScaler** from **sklearn.preprocessing**.
    - Reshape the input data to match the LSTM input shape.
4. Build the LSTM model:
    - Import the necessary Keras modules: **Sequential**, **Dense**, and **LSTM**.
    - Initialize a sequential model (**regressor**).
    - Add the LSTM layer with the desired number of units and activation function.
    - Add the output layer with one unit.
    - Compile the model using the Adam optimizer and mean squared error loss.
5. Train the LSTM model:
    - Fit the model to the training data with the specified batch size and number of epochs.
6. Make predictions:
    - Load the test data.
    - Scale the test data using the same scaler as the training data.
    - Reshape the input data for prediction.
    - Use the trained model to predict the stock prices.
    - Inverse transform the predictions to get the actual stock prices.
7. Visualization:



- Plot the real stock prices and the predicted stock prices using **matplotlib**.

8. Evaluation:
    - Calculate the Root Mean Squared Error (RMSE) between the real stock prices and the predicted stock prices using **mean_squared_error** from **sklearn.metrics**.
9. Further analysis:
    - Load the training data again.
    - Predict the stock prices for the training data.
    - Inverse transform the predictions.
    - Plot the real and predicted stock prices for the training data.

Conclusion

This project demonstrates the use of LSTM for stock price prediction. The model is trained on historical stock price data and provides predictions for both the test and training data. The predictions are evaluated using RMSE. The visualizations help compare the real and predicted stock prices, providing insights into the model's performance.

Remember to customize the file paths and adjust hyperparameters as needed for your specific project.