MLOps Training - DataScientest

Stock Investigations - Report

Model Development, API Implementation & Specifications

1. Introduction

Stock price prediction is a critical task that involves forecasting future stock prices based on historical data. This report outlines the process of developing a stock price prediction model using Long Short-Term Memory (LSTM) neural networks. It includes details about data preprocessing, model selection, evaluation metrics, and a comprehensive description of the FastAPI-based API developed for model deployment.

2. Data Preprocessing

2.1 Data Collection

The initial step in the project was to gather historical stock price data, which serves as the foundation for training the predictive model. The data was collected using the `yfinance` library, which allows users to download stock data from Yahoo Finance. The dataset includes crucial features such as:

- **Open**: The price at which the stock opens for the trading day.
- **High**: The highest price the stock reached during the trading day.
- **Low**: The lowest price the stock reached during the trading day.
- **Close**: The price at which the stock closes for the trading day.
- **Adjusted Close**: The closing price adjusted for dividends and stock splits, providing a more accurate reflection of the stock's value over time.
- **Volume**: The total number of shares traded during the day.

	Date	Open	High	Low	Close	Adj Close	Volume
0	2014-09-12	25.302500	25.547501	25.270000	25.415001	22.566164	250504400
1	2014-09-15	25.702499	25.762501	25.360001	25.407499	22.559500	245266000
2	2014-09-16	24.950001	25.315001	24.722500	25.215000	22.388578	267632400
3	2014-09-17	25.317499	25.450001	25.147499	25.395000	22.548403	243706000
4	2014-09-18	25.482500	25.587500	25.389999	25.447500	22.595026	149197600

2.2 Data Cleaning

Data cleaning is an essential step to ensure the integrity of the dataset. In this project, we focused on the following aspects:

- **Handling Missing Values**: Any missing data points were either imputed or removed to maintain the quality of the dataset.
- **Removing Duplicates**: Duplicate entries were identified and eliminated to prevent skewing the analysis.
- **Correcting Inconsistencies**: Any discrepancies in data formats were resolved to ensure uniformity across the dataset.

2.3 Normalization/Scaling

Normalization is crucial when working with neural networks. We used Min-Max scaling to scale the features to a range between 0 and 1. This transformation helps in speeding up the convergence of the model during training.

2.4 Data Splitting

The preprocessed dataset was split into training and testing sets. The training set was used to train the LSTM model, while the testing set was reserved for evaluating its performance. The split ratio was set to 80% for training and 20% for testing.

3. Choosing the Model

For this project, we selected the Long Short-Term Memory (LSTM) model, a specialized type of Recurrent Neural Network (RNN).

3.1 Why LSTM?

LSTMs are particularly effective for time series data for several reasons:

- **Long-Term Dependencies**: LSTMs can learn dependencies over long sequences of data. They achieve this through memory cells that can maintain information over time, making them well-suited for capturing trends and patterns in stock price movements.
- **Avoiding Vanishing Gradients**: Traditional RNNs often face issues with vanishing gradients, which can hinder training. LSTMs address this problem through their unique architecture, allowing them to retain relevant information for extended periods.
- **Empirical Performance**: Numerous studies have shown that LSTMs outperform other models, such as ARIMA and traditional feedforward neural networks, in time series forecasting tasks, particularly in finance.

3.2 Model Architecture

The architecture of the LSTM model used in this project consists of the following layers:

- **Input Layer**: Accepts the input features for the model.
- **LSTM Layers**: One or more LSTM layers process the input data and learn the temporal patterns.
- **Dense Layer**: A fully connected layer that outputs the predicted stock price.
- **Output Layer**: Produces the final prediction.

The model was compiled using the Adam optimizer and the mean squared error (MSE) loss function. Bellow is model’s summary.

Layer (type)	Output Shape	Param #
lstm_4 (LSTM)	(None, 60, 100)	40,800
dropout_4 (Dropout)	(None, 60, 100)	0
lstm_5 (LSTM)	(None, 50)	30,200
dropout_5 (Dropout)	(None, 50)	0
dense_2 (Dense)	(None, 1)	51

Total params: 71,053 (277.55 KB)
Trainable params: 71,051 (277.54 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 2 (12.00 B)

4. Evaluation Metrics

To assess the performance of the LSTM model, we employed several evaluation metrics:

- **Mean Squared Error (MSE)**: MSE measures the average squared difference between predicted and actual values. In our model, the MSE was **48.65**, indicating how close the predictions were to the true values.
- **Root Mean Squared Error (RMSE)**: RMSE is the square root of MSE and provides a metric in the same units as the predicted values. Our model achieved an RMSE of **6.97**, which is useful for understanding the average prediction error.
- **Mean Absolute Error (MAE)**: MAE represents the average absolute difference between predicted and actual values, recorded at **5.50**. It offers a straightforward interpretation of prediction errors.
- **Mean Absolute Percentage Error (MAPE)**: MAPE indicates the average percentage error, providing insight into accuracy relative to the scale of the data. Our model recorded a MAPE of **2.91%**, suggesting a strong performance.
- **R-squared (R^2)**: R^2 represents the proportion of variance for the dependent variable explained by independent variables. Our model achieved an R^2 value of **0.92**, indicating a high level of explanatory power.

All these metrics are stored in a JSON file named `lstm_model_metrics.json`, providing a comprehensive overview of the model's performance.

5. API Implementation

To make the stock price prediction model accessible to users, we developed an API using the FastAPI framework. This API allows users to interact with the model seamlessly, providing various functionalities.

5.1 API Features

The FastAPI implementation includes the following endpoints:

- **Model Loading**: The API loads the pre-trained LSTM model and the corresponding Min-Max scaler for data preprocessing.
- **Evaluation Endpoint**: The `/evaluate` endpoint allows users to trigger model evaluation by sending a request. The API loads test data (in the form of NumPy arrays), computes evaluation metrics, and returns them in a JSON format.
- **Stock Data Downloading**: The `/download_stock_data` endpoint enables users to fetch historical stock data from Yahoo Finance for any specified ticker over the last 10 years. This feature helps users gather data for further analysis or model evaluation.
- **Prediction Endpoint**: The `/predict` endpoint accepts input in the form of a JSON object containing stock data (open, high, low, close, and volume) and returns the predicted stock price. This feature allows users to obtain predictions based on their input data.
- **Metrics Endpoint**: The `/metrics` endpoint provides users with the evaluation metrics saved in the JSON file. This feature enables users to review the model's performance without re-evaluating it.

5.2 API Specifications

Here are the specifications for the API:

Base URL

arduino

Copy code

```
http://localhost:8000
```

Endpoints

1. **GET /**
    - **Description**: Returns a message indicating that the API is running.

Response:

json

Copy code

```
{
  "message": "Stock prediction API is running"
}
```

2. **GET /evaluate**
    - **Description**: Evaluates the model using test data.

Response:

json

Copy code

```
{
  "Test Loss": <loss_value>,
  "Test MAE": <mae_value>,
  "Test MSE": <mse_value>
}
```

3. **GET /download_stock_data**

- **Description**: Downloads historical stock data for a given ticker.
- **Parameters**:
  - `ticker` (string): The stock ticker symbol.

Response:

json

Copy code

```
{
  "message": "Data for <ticker> downloaded successfully"
}
```

4. **POST /predict**
   - **Description**: Predicts the stock price based on input data.

Request Body:

json

Copy code

```
{
  "open": <float>,
  "high": <float>,
  "low": <float>,
  "close": <float>,
  "volume": <float>
}
```

Response:

json

Copy code

```
{
```

```
  "prediction": <predicted_value>
}
```

5. **GET /metrics**
    - **Description**: Returns the evaluation metrics of the model.

Response: (as loaded from `lstm_model_metrics.json`)

json

Copy code

```
{
  "Mean Squared Error": 48.65055376241213,
  "Root Mean Squared Error": 6.974994893361007,
  "Mean Absolute Error": 5.497448539307181,
  "Mean Absolute Percentage Error": 2.9074069092573644,
  "R-squared": 0.9153326545880341
}
```

5.3 Potential Use Cases

The API offers several potential applications:

Traders and Investors: The API allows traders and investors to predict future stock prices based on historical data, aiding their investment decisions and strategies. By providing predictions of stock movements, users can optimize their entry and exit points for trades, thereby enhancing profitability.

Data Analysts: Analysts can utilize the API to evaluate model performance and gain insights into stock price trends. By accessing historical data and using the model's predictions, analysts can conduct deeper analyses on market behavior, enabling them to identify patterns and make data-driven recommendations to stakeholders.

Financial Institutions: Banks and investment firms can integrate the API into their financial systems to automate trading strategies or inform clients about potential investment opportunities. By leveraging predictive analytics, financial institutions can enhance their service offerings and improve client engagement.

Developers and Researchers: Developers can build applications that integrate stock price predictions into trading platforms, financial dashboards, or market analysis tools. Researchers can use the API to test different machine learning models, experiment with new features, or study stock market behavior in relation to various economic indicators.

Educational Institutions: Students and educators can use the API in finance and data science courses to demonstrate the application of machine learning in real-world scenarios. The API can serve as a practical tool for teaching concepts of stock market analysis, predictive modeling, and financial technology.

Portfolio Managers: Portfolio managers can use the API to assess potential investments and manage risk effectively. By analyzing the predicted price movements, they can make more informed decisions about asset allocation, hedging strategies, and overall portfolio performance.

Hobbyist Traders: Individual traders interested in stock trading can leverage the API to improve their trading skills by making predictions and validating their strategies against model outputs. This can serve as a learning tool, helping them understand market dynamics and refine their trading tactics.

6. Conclusion

The stock price prediction API has broad applicability across various sectors in finance and technology. By offering predictive analytics based on historical stock data, it empowers users to make more informed decisions, optimize strategies, and deepen their understanding of market behavior.