Low Level Design Document

Stock Price Prediction System

Document Version: 1.0
Last Revised Date: 2024-06-07
Written By: Tejas Pandav

DOCUMENT CONTROL

Date Issued	Version	Description	Author
2024-05-22	1.0	Initial Version	Tejas Pandav

Contents

1. Abstract

This document provides the low-level design details for the Stock Price Prediction application, focusing on the implementation specifics for each module, class, and method. It includes detailed descriptions of the data processing, model training, prediction workflow, and the Streamlit application interface. The application uses machine learning techniques to predict future stock prices based on historical data, enhancing traditional investment models with automated predictions.

2. Introduction

The Stock Price Prediction application aims to forecast future stock prices using machine learning techniques. This document breaks down the high-level components into detailed implementations, including functions, methods, data flows, and user interactions. The application leverages historical stock data, applies feature engineering techniques, trains a machine learning model, and provides a user-friendly interface for predictions and data visualization.

3. Module Descriptions

Fetch Historical Stock Data

The `fetch_stock_data` module is responsible for fetching historical stock data from Yahoo Finance. It takes the stock symbol, start date, and end date as inputs and returns a DataFrame containing the historical stock data, including columns such as Date, Open, High, Low, Close, Adj Close, and Volume. This module ensures the availability of accurate historical data for further processing and analysis.

Feature Engineering

The `create_features` module enhances the raw stock data by adding technical indicators such as the Simple Moving Average (SMA) and the Relative Strength Index (RSI). These features are crucial for capturing market trends and price momentum, which are important inputs for the machine learning model. The module takes a DataFrame containing stock data and returns a DataFrame with additional columns for SMA and RSI.

Data Preparation

The `prepare_data` module prepares the stock data for model training. It selects the relevant features (SMA and RSI) and the target variable (closing prices) from the enhanced DataFrame. The module ensures that the data is clean and suitable for training by handling missing values and normalizing the features. The outputs of this module are the feature matrix (X) and the target vector (y), which are used for training the machine learning model.

Model Training

The `train_model` module trains an XGBoost regressor model using the prepared data. It takes the feature matrix (X) and the target vector (y) as inputs and returns a trained XGBoost model. The module includes hyperparameter tuning and cross-validation to ensure the model's accuracy and robustness. This trained model is then used for making predictions on future stock prices.

Prediction

The `predict_future_prices` module predicts future stock prices for the next 30 days using the trained XGBoost model. It generates future dates, prepares the data for these dates by applying the same feature engineering techniques, and makes predictions using the model. The module returns a list of future dates and the corresponding predicted prices. It also updates the DataFrame with predicted prices for iterative predictions.

Streamlit Application

The `main` module defines the structure of the Streamlit application. It includes pages for displaying actual stock values, making predictions, and providing detailed information about stock tickers. The module handles user inputs, fetches and processes data, trains the model, and displays the results in an interactive web application. The Streamlit interface ensures a seamless user experience with easy navigation and clear visualizations.

4. Class and Method Specifications

fetch_stock_data(symbol, start_date, end_date)

The `fetch_stock_data` function fetches historical stock data from Yahoo Finance. It requires the stock symbol (e.g., "AAPL"), start date, and end date as inputs. The function returns a DataFrame containing columns such as Date, Open, High, Low, Close, Adj Close, and Volume. This data is essential for subsequent analysis and model training.

create_features(data)

The `create_features` function adds technical indicators to the stock data. It takes a DataFrame containing historical stock data as input and returns a DataFrame with additional columns for SMA and RSI. The function applies a rolling window to calculate the SMA and a custom function to compute the RSI, capturing important market trends and price momentum.

calculate_RSI(close_prices, window=14)

The `calculate_RSI` function calculates the Relative Strength Index (RSI) for a series of closing prices. It takes the closing prices and an optional window size (default is 14) as inputs. The function returns a series with RSI values, which are used to gauge the speed and change of price movements. The RSI helps identify overbought or oversold conditions in the market.

prepare_data(data)

The `prepare_data` function prepares the stock data for model training. It takes a DataFrame containing stock data with features (SMA and RSI) and returns the feature matrix (X) and target vector (y). The function handles missing values by dropping rows with NaNs and normalizes the features to ensure consistent input for the machine learning model.

train_model(X_train, y_train)

The `train_model` function trains an XGBoost regressor model. It requires the training feature matrix (X_train) and the target vector (y_train) as inputs. The function performs hyperparameter tuning and cross-validation to optimize the model's performance. It returns the trained XGBoost model, ready for making predictions on future stock prices.

predict_future_prices(model, data)

The `predict_future_prices` function predicts future stock prices for the next 30 days using the trained XGBoost model. It takes the trained model and a DataFrame containing recent stock data as inputs. The function generates future dates, prepares the data for these dates by applying the same feature engineering techniques, and makes predictions. It returns a list of future dates and the corresponding predicted prices.

5. User Interface Design

Streamlit Application

The Streamlit application is divided into three main pages: Actual Value, Prediction, and Ticker Information.

- **Home Page**: The home page displays the title "Stock Price Prediction App" and provides navigation options.
- **Actual Value Page**: This page allows users to enter a stock symbol and select a time period for fetching historical data. It displays the fetched data in a table and plots the actual closing prices.

- **Prediction Page**: This page allows users to enter a stock symbol, fetch recent data, train the model, and display the predicted future prices. It shows the predicted prices in a table and plots the actual vs predicted prices.
- **Ticker Information Page**: This page provides detailed information about a stock ticker, including company details, financial ratios, and other relevant data.

6. Error Handling and Logging

Error Handling

The application includes error handling mechanisms to ensure robustness and user-friendly error messages.

- **Data Fetching Errors**: Handles network errors and invalid symbols by displaying appropriate messages to the user.
- **Model Training Errors**: Catches and logs errors that occur during model training, providing feedback to the user and logs for debugging.
- **Prediction Errors**: Ensures predictions handle edge cases, such as missing data or invalid inputs, and provides clear messages to the user.

Logging

Logging is configured to capture events, errors, and warnings. Log files are stored with timestamps to facilitate debugging and monitoring. The logs include details about data fetching, model training, prediction processes, and user interactions.

7. Testing Strategy

Unit Tests

Unit tests are designed to test each function independently with various inputs and edge cases. Mocking is used to simulate API responses from Yahoo Finance, ensuring the functions handle different scenarios correctly.

Integration Tests

Integration tests ensure the end-to-end data flow from fetching data to making predictions works seamlessly. These tests validate the interaction between different modules and the overall application workflow.

Performance Tests

Performance tests simulate multiple users accessing the application simultaneously to ensure it can handle high loads. Response time is measured for data fetching, model training, and predictions to ensure the application performs efficiently under different conditions.

8. References

- Yahoo Finance API: https://pypi.org/project/yfinance/
- Streamlit Documentation: https://docs.streamlit.io/
- XGBoost Documentation: https://xgboost.readthedocs.io/
- Pandas Documentation: https://pandas.pydata.org/
- Matplotlib Documentation: https://matplotlib.org/stable/users/index.html