Unified Mentor - Stock Market

-Adimalla Nithin Siddhartha

Introduction

This document provides a detailed account of the process undertaken to analyze historical stock price data and develop a predictive model. The primary objective is to forecast the adjusted close price of stocks using a combination of raw price data, trading volume, and technically derived features. This analysis utilizes an LSTM neural network, a type of recurrent neural network particularly adept at processing sequential data, making it suitable for time series forecasting. The entire process, from data loading and preprocessing through model training, evaluation, and hyperparameter tuning, is described in detail.

Data Collection

The initial step involved acquiring the stock price data. This was done by loading the `stocks.csv` file into a pandas DataFrame. The DataFrame provides a tabular structure to hold the time-series stock data, with columns representing different aspects of the stock's performance on specific dates. The raw data includes information such as the ticker symbol, date, opening price, highest price, lowest price, closing price, adjusted closing price, and trading volume.

Data Preprocessing

Before any analysis or modeling could take place, the raw data required several preprocessing steps to ensure its quality and prepare it for consumption by the model.

Handling Missing Values: A check for missing values was performed across all columns. In this specific dataset, no missing values were found. However, in a real-world scenario, strategies like imputation (filling missing values with estimates) or removal of rows/columns with excessive missing data would be necessary.

Data Type Conversion: The 'Date' column, initially loaded as a generic object type, was explicitly converted to datetime objects. This conversion is crucial for time series analysis as it allows for proper handling of temporal relationships and enables time-based indexing and operations.

Feature Selection: A subset of the columns from the original DataFrame was selected to form the basis of the features for the model. The selected features included 'Date', 'Open', 'High', 'Low', 'Close', 'Volume', and 'Adj Close'. This step focuses the analysis on the most relevant information for predicting the adjusted close price.

Exploratory Data Analysis (EDA)

Exploratory Data Analysis was conducted to gain a deeper understanding of the dataset's characteristics, identify patterns, and inform subsequent modeling decisions.

Descriptive Statistics: Standard descriptive statistics (count, mean, standard deviation, min, max, quartiles) were calculated for the numerical columns ('Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'). These statistics provide a summary of the central tendency, dispersion, and shape of the data's distribution. Additionally, the Interquartile Range (IQR) and outlier thresholds based on the IQR method were computed for reference.

Time Series Visualization: The core of the data is its time-series nature. To visualize this, line plots of the 'Close' price and 'Volume' over time were generated. These plots help in observing trends, seasonality (if any), and volatile periods in the stock's history. The 'Close' price plot reveals the general movement of the stock's value, while the 'Volume' plot indicates trading activity.

Correlation Analysis: A correlation matrix was computed for the numerical features. This matrix quantifies the linear relationship between pairs of variables. A heatmap visualization of the correlation matrix provided a clear and intuitive representation of these relationships. High correlations between 'Open', 'High', 'Low', 'Close', and 'Adj Close' are expected for stock price data, while the correlation with 'Volume' might reveal insights into how trading activity relates to price movements.

Feature Engineering

To potentially improve the predictive power of the model, additional features were engineered from the existing data. These features are often technical indicators used in financial analysis.

Simple Moving Averages (SMA): Calculated the 20-day and 50-day Simple Moving Averages of the 'Close' price. Moving averages smooth out price data to create a single flowing line, making it easier to spot trends. Short-term SMAs (like 20-day) are more sensitive to recent price changes than long-term SMAs (like 50-day). These were added as new columns ('SMA_20', 'SMA_50') to the `df_processed` DataFrame.

Price Change: Calculated the difference between the 'Close' price and the 'Open' price for each day. This feature, 'Price_Change', directly represents the daily net change in stock value, which can be a useful predictor.

Lagged Close Price: Created a lag feature by shifting the 'Close' price column by one day. 'Close_Lag1' represents the previous day's closing price, a fundamental feature in time series forecasting as past values are strong predictors of future values.

Handling NaNs after Feature Engineering: The calculation of moving averages and lagged features introduces `NaN` values at the beginning of the time series. These rows containing `NaN` values were removed using `dropna()` to ensure the model receives complete data.

Model Selection

Given the sequential and time-dependent nature of stock price data, a Recurrent Neural Network (RNN) was chosen as the modeling approach. Specifically, a Long Short-Term Memory (LSTM) network was selected. LSTMs are a type of RNN capable of learning long-term dependencies, making them particularly effective for time series forecasting where past price movements can influence future prices.

Model Training and Evaluation (Initial Model)

The prepared data was used to train and evaluate an initial version of the LSTM model.

Data Splitting: The `df_processed` DataFrame was split into training and testing sets based on a chronological split. An 80/20 split ratio was used, with the first 80% of the data used for training and the remaining 20% for testing. This chronological split is essential in time series forecasting to prevent data leakage from the future into the past.

Data Scaling: Both the features (X) and the target variable (y) were scaled using `MinMaxScaler`. This process transforms the data to a common scale (between 0 and 1), which helps to prevent features with larger values from dominating the learning process and improves the convergence of the neural network. The scaler was `fit_transform` on the training data and only `transform` on the testing data to avoid data leakage.

Data Reshaping: LSTM layers in Keras require input data in a 3D format: `[samples, time steps, features]`. Since we are predicting the next time step based on the current time step's features, the data was reshaped accordingly, with `time steps` set to 1.

Initial Model Architecture: A Sequential LSTM model was built with two LSTM layers and two Dropout layers to mitigate overfitting. The final layer was a Dense layer with one unit for the prediction. The model was compiled using the 'adam' optimizer and 'mean_squared_error' as the loss function.

Initial Model Training: The initial model was trained on the scaled and reshaped training data for a specified number of epochs and a batch size. A validation split of 10% was used from the training data to monitor the model's performance during training and detect potential overfitting.

Initial Model Evaluation: The trained model was evaluated on the scaled testing data. The predictions were then inverse-transformed back to the original price scale. Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) were calculated between the actual and predicted values on the test set. These metrics provide a measure of the average squared difference and the square root of the average squared difference between the actual and predicted values, respectively. A lower MSE and RMSE indicate better model performance. A plot of the actual vs. predicted prices for the initial model on the test set was generated to visualize the model's performance.

Model Tuning and Optimization

To further enhance the model's performance, hyperparameter tuning was performed using the Keras Tuner library.

Hyperparameter Search Space: A search space for hyperparameters was defined within the `build_model` function. This included the number of units in the LSTM layers (`lstm_units_1`, `lstm_units_2`), the dropout rates (`dropout_1`, `dropout_2`), and the optimizer ('adam' or 'rmsprop').

Keras Tuner Setup: The `Hyperband` tuner was chosen for efficient hyperparameter search. The objective was to minimize the validation loss (`val_loss`) during the tuning process. The tuner was configured with parameters like `max_epochs` and `factor`. A directory and project name were specified for storing the tuning results.

Tuning Execution: The `tuner.search()` method was called with the training data and a validation split. Keras Tuner iteratively builds and trains models with different hyperparameter combinations within the defined search space, using the validation loss to guide the search.

Retrieving Best Hyperparameters: After the search was complete, the `tuner.get_best_hyperparameters()` method was used to retrieve the hyperparameter combination that resulted in the best performance (lowest validation loss) during the tuning process. The best hyperparameters were printed for inspection.

Tuned Model Building and Training: A new model was built using the `tuner.hypermodel.build()` method with the best hyperparameters found. This tuned model was then trained on the entire training dataset (without a separate validation split during final training) for the number of epochs determined by the tuner.

Tuned Model Evaluation: The trained tuned model was used to make predictions on the scaled testing data. The predictions were inverse-transformed. MSE and RMSE were calculated for the tuned model's predictions against the actual test set values.

Performance Comparison: The MSE and RMSE values of the initial model and the tuned model were printed side-by-side to clearly show the impact of hyperparameter tuning on the model's performance. A plot of the actual vs. tuned predicted prices was generated to visually compare the tuned model's predictions against the actual prices. The comparison of evaluation metrics and the visual representation demonstrated that the tuned model achieved a lower error, indicating improved predictive accuracy.