PREDICTING STOCK PRICE USING DEEP LEARNING

BATCH MEMBER

621721243040 : PRAVENRAJ D V

Project Title: Stock Price Prediction

Phase 4: Development Part 2

Topic: Start building the stock price prediction model by

Feature engineering, Model training, Evaluation.



Phase 4 submission document

Stock Price Prediction

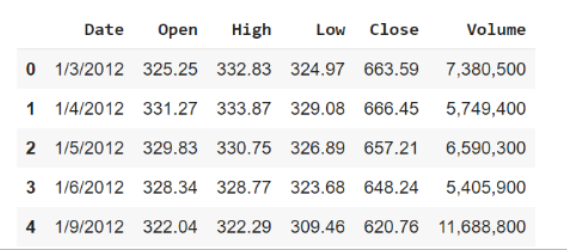
**Introduction:**

Stock Price Prediction using machine learning helps you discover the future value of company stock and other financial assets traded on an exchange. The entire idea of predicting stock prices is to gain significant profits.

**LSTMs** work in a three-step process:

* The first step in **LSTM** is to decide which information to be omitted from the cell in that particular time step. It is decided with the help of a sigmoid function. It looks at the previous state (ht-1) and the current input xt and computes the function.
* There are two functions in the second layer. The first is the sigmoid function, and the second is the tanh function. The sigmoid function decides which values to let through (0 or 1). The tanh function gives the weightage to the values passed, deciding their level of importance from -1 to 1.
* The third step is to decide what will be the final output. First, you need to run a sigmoid layer which determines what parts of the cell state make it to the output. Then, you must put the cell state through the tanh function to push the values between -1 and 1 and multiply it by the output of the sigmoid gate.

**Given data set:**



**Google Stock Price Prediction Using LSTM**

### **Import the Libraries.**

#import libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

### **Load the Training Dataset.**

There are five columns. The Open column tells the price at which a stock started trading when the market opened on a particular day. The Close column refers to the price of an individual stock when the stock exchange closed the market for the day. The High column depicts the highest price at which a stock traded during a period. The Low column tells the lowest price of the period. Volume is the total amount of trading activity during a period of time.

dataset\_train = pd.read\_csv(“Google\_Stock\_Price\_Train.csv”)

dataset\_train.head()

### **Use the Open Stock Price Column to Train Your Model.**

training\_set=dataset\_train.iloc[:,1:2].values

print(training\_set)

print(training\_set.shape)

[[325.25]

[331.27]

[329.83]

……

[793.7]

[783.33]

[782.75]]

(1258,1)

### **4. Normalizing the Dataset.**

From sklearn.preprocesssing import minmaxscalar

Scalar=minmaxscalar(feature\_range=(0,1))

Scaled\_training\_set

array([[0.0.8581368],

[0.09701243],

[0.09433366],

…..,

[0.95725128],

[0.93576041],

[0.93688146]])

### **5. Creating X\_train and y\_train Data Structures**

x\_train=[]

y\_train=[]

for i in range(60,1258):

x\_train.append(scaled\_trained\_set[i-60:i,0])

y\_train.append(scaled\_trained\_set[i,0])

x\_train=np.array(x\_train)

y\_train=np.array(y\_train)

print(x\_train.shape)

print(y\_train.shape)

(1198,60)  
 (1198,)

### **6. Reshape the Data**

x\_train=np.reshape(x\_train,(x\_train.shape[0],x\_train.shape[1],1))

x\_train.shape

(1198,60,1)

### **7. Building the Model by Importing the Crucial Libraries and Adding Different Layers to LSTM.**

from keras.models import sequential

from keras.layers import LSTM|

from keras.layers import dense

from keras.layers import dropout

regressor =Sequential( )

regressor.add(LSTM(units = 50, return\_sequences= True, input\_shape = (X\_traín.shape[1], 1)))

regressor. add(Dropout(0 . 2) )

regressor.add(LSTM(units = 50, return\_sequences=True)

regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return\_sequences= True) )

regressor. add(Dropout(0. 2) )

regressor.add(LSTM(units = 50))

regressor. add(Dropout(0 . 2) )

regressor.add(Dense(units=1))

### **8. Fitting the Model**

regressor.compile(optimizer=’adam’,loss=’mean\_squared\_error’)

regressor.fit(x\_train,y\_train,epochs=100,batch\_size=32)

epoch1/100

38/38 [====================] – 11s 114s/step – loss:0.1011

epoch2/100

38/38 [====================] – 4s 117s/step – loss:0.0061

epoch3/100

38/38 [====================] – 4s 118s/step – loss:0.0063

### **9. Extracting the Actual Stock Prices of Jan-2017**

dataset\_test=pd.read\_csv(“google\_stock\_price\_test.csv”)

actual\_stock price = dataset\_test.iloc[:,1:2].values

### **10. Preparing the Input for the Model**

dataset\_total=pd.concat((dataset\_train[‘open’],dataset\_test[‘open’]),axis=0)

inputs=dataset\_total[len(dataset\_total)-len(dataset\_test)-60:].values

inputs=inputs.reshape(-1,1)

inputs=scalar.transform(inputs)

x\_test=[]

for i in range(60,80):

x\_test.append(inputs[i-60:I,0])

x\_test=np.array(x\_test)

x\_test=np.reshape(x\_test,(x\_test.shape[0],x\_test.shape[1],1))

### **11. Predicting the Values for Jan 2017 Stock Prices.**

predicted\_stock\_price=regressor.predict(x\_test)

predicted\_stock\_price=scalar.inverse\_transform(ptredicted\_stock\_price)

### **12. Plotting the Actual and Predicted Prices for Google Stocks**

plt.plot(actual\_stock\_price,color=’red’,label=’actual google stock price’)

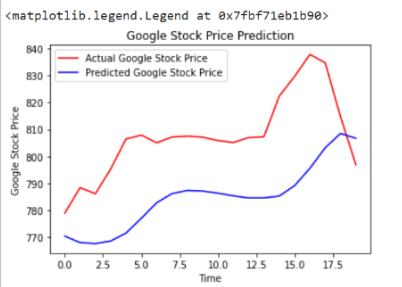
plt.plot(predicted\_stock\_price,color=’blue’,label=’predicted google stock price’)

plt.title(‘google stock price prediction’)

plt.xlabel(‘time’)

plt.ylabel(‘google stock price’)

plt.legend()



**Feature engineering:**

Feature engineering is a critical step in developing predictive models for stock price movements using deep learning techniques. Properly engineered features can help your model capture meaningful patterns and relationships in historical price data. Here are some feature engineering ideas for stock price prediction using deep learning:

**1.Price and Volume Data:**

* Historical prices: Include features like daily open, high, low, and close prices.
* Trading volume: Add daily trading volume as a feature.
* Price returns: Calculate daily returns, such as percentage change in closing prices.

**2.Moving Averages:**

* Simple moving averages (SMA): Calculate the SMA over various time periods (e.g., 10 days, 50 days, 200 days).
* Exponential moving averages (EMA): Calculate the EMA over different time frames.

**3.Volatility Measures:**

* Historical volatility: Compute the historical volatility using metrics like the standard deviation of returns over a specified window.
* Bollinger Bands: Create features based on Bollinger Bands to capture price volatility.

**4.Technical Indicators:**

* Relative Strength Index (RSI): A momentum oscillator that measures the speed and change of price movements.
* Moving Average Convergence Divergence (MACD): A trend-following momentum indicator.
* Stochastic Oscillator: A momentum indicator comparing a particular closing price to a range of its prices over a specific period.

**5.Lagged Features:**

* Include lagged versions of the target variable and other relevant features, which can help capture autocorrelation in the time series data.

**6.Calendar and Economic Events:**

* Include features related to important calendar events like holidays, earnings reports, and economic indicators like interest rates, GDP, etc.

**7.Market Sentiment:**

* Incorporate sentiment analysis of news articles, social media, or financial reports related to the stock or the market in general.

**8.Market Index Data:**

* Include features related to broader market indices (e.g., S&P 500, NASDAQ) to capture overall market trends and sentiment.

**9.Fundamental Data:**

* If available, include fundamental data like earnings per share (EPS), price-to-earnings (P/E) ratio, and other financial metrics.

**10.Seasonal Patterns:**

* Identify and include features that capture seasonal patterns if relevant to the stock or industry.

**11.Correlations and Cross-Correlations:**

* Calculate and include correlations between your target stock and related stocks, sectors, or market indices.

**12.Time of Day and Day of Week:**

* Incorporate features that capture intraday patterns and day-of-week effects on stock prices.

**13.Feature Scaling:**

* Normalize or standardize your features to ensure that deep learning models can work effectively.

**14.Feature Selection:**

* Use techniques like feature importance analysis or dimensionality reduction to select the most relevant features and eliminate noise.

**15.Custom Features:**

* Experiment with creating custom features specific to the industry or stock you are analyzing.

**16.Data Preprocessing:**

* Handle missing data and outliers appropriately, as these can significantly affect model performance.

**17.Sequential Data:**

* If using recurrent neural networks (RNNs) or Long Short-Term Memory (LSTM) models, structure your data into sequences of historical prices and indicators.

**Model Training:**

Training a deep learning model for stock price prediction is a complex task that involves several steps, from data preparation to model architecture selection, training, and evaluation. Here's a step-by-step guide on how to train a deep learning model for stock price prediction:

**1.Data Collection and Preprocessing:**

* Collect historical stock price data, including features you've engineered during the feature engineering stage.
* Split the data into training, validation, and test sets. A common split might be 70% for training, 15% for validation, and 15% for testing.
* Normalize or standardize the data to ensure that all features have the same scale. This is especially important for deep learning models.

**2.Sequence Generation (for Time Series Models):**

* If you're using RNNs, LSTMs, or CNNs for time series data, create sequences of historical data with corresponding target values. For example, if you're using daily data, a sequence might consist of the past N days' data, and the target would be the next day's price.

**3.Selecting the Model Architecture:**

* Choose a suitable deep learning architecture for your task. Common choices for stock price prediction include:
* Recurrent Neural Networks (RNNs) and Long Short-Term Memory networks (LSTMs) for sequential data.
* Convolutional Neural Networks (CNNs) for analyzing image-based financial charts.
* Feedforward Neural Networks (FNNs) for tabular data or simpler time series predictions.
* Experiment with different architectures to find the one that works best for your dataset.

**4.Loss Function and Metrics:**

* Select an appropriate loss function for regression tasks, such as Mean Squared Error (MSE) or Mean Absolute Error (MAE).
* Choose evaluation metrics, including Mean Absolute Error (MAE), Mean Squared Error (MSE), or others that are relevant to your specific problem.

**5.Model Training:**

* Train your deep learning model using the training dataset. During training, you'll aim to minimize the chosen loss function.
* Implement mini-batch training to improve convergence speed and prevent overfitting.
* Monitor training progress by tracking loss and metrics on the validation set.
* Implement techniques to prevent overfitting, such as dropout, early stopping, and regularization.

**6.Hyperparameter Tuning:**

* Experiment with different hyperparameters, including learning rate, batch size, the number of hidden layers, and the number of neurons in each layer, to optimize model performance.
* Use techniques like grid search or random search to find the best combination of hyperparameters.

**7.Regularization:**

* Implement regularization techniques such as L1 and L2 regularization to prevent overfitting.

**8.Feature Importance Analysis:**

* Analyze feature importance to understand which features are the most informative for the model.

**9.Model Evaluation:**

* Evaluate the model's performance on the test dataset using the chosen evaluation metrics. This step will give you an idea of how well your model generalizes to unseen data.

**10.Backtesting and Real-World Application:**

* If you plan to use your model for trading or investment, conduct backtesting to see how it would have performed in a real-world setting.
* Implement a strategy for trading based on model predictions and evaluate its performance.

**11.Monitoring and Model Maintenance:**

* Continuously monitor the model's performance and retrain it as necessary, as stock market dynamics can change over time.

**12.Deployment:**

* If your model performs well in a real-world setting, you can deploy it for live predictions.

**Evaluating:**

Evaluating the performance of a deep learning model for stock price prediction is crucial to determine how well the model is performing and whether it's providing meaningful insights for trading or investment decisions. Here are some common evaluation techniques and metrics for assessing the performance of your stock price prediction model:

**1.Mean Absolute Error (MAE):**

* MAE measures the average absolute difference between the predicted and actual stock prices. It provides a straightforward understanding of prediction accuracy. A lower MAE indicates better performance.

**2.Mean Squared Error (MSE):**

* MSE measures the average squared difference between the predicted and actual prices. It punishes larger errors more severely than MAE. Lower MSE values indicate better performance, but they might be less intuitive to interpret due to the squared nature of the metric.

**3.Root Mean Squared Error (RMSE):**

* RMSE is the square root of the MSE, which gives a measure of the average error in the same units as the target variable (stock prices). It is more interpretable than MSE.

**4.Mean Absolute Percentage Error (MAPE):**

* MAPE calculates the percentage difference between predicted and actual prices, which is particularly useful when you want to understand the relative error in terms of percentage. It's especially relevant for comparing predictions across different stocks with varying price levels.

**5.Directional Accuracy (Hit Ratio):**

* This metric evaluates whether the model correctly predicts the direction (up or down) of stock price movements. It can be a useful measure for trading strategies.

**6.Sharpe Ratio and Other Financial Metrics:**

* If you plan to use your model for trading, you can evaluate its performance using financial metrics like the Sharpe ratio, which considers risk-adjusted returns.

**7.Backtesting:**

* Backtesting involves simulating trading or investment decisions based on model predictions and assessing how well these decisions would have performed historically. Backtesting can provide insights into the model's practical utility.

**8.Cross-Validation:**

* Implement cross-validation techniques, such as k-fold cross-validation, to assess the model's stability and generalization performance.

**9.Out-of-Sample Testing:**

* It's important to evaluate your model's performance on a separate, unseen dataset (test set) to ensure it generalizes well to new data. This is especially important in time series forecasting.

**10.Benchmark Comparison:**

* Compare the model's performance to that of a simple benchmark, such as a random walk (predicting the next price as the current price) or a basic moving average strategy, to assess whether the model adds value beyond simple approaches.

**11.Confidence Intervals:**

* Calculate confidence intervals around your predictions to quantify the uncertainty of your model. This is particularly important when making financial decisions based on predictions.

**12.Visual Inspection:**

* Plot the actual and predicted stock prices over time to visually assess how well the model captures price movements and trends.

**13.Post-Analysis:**

* Perform post-analysis to understand the causes of errors or outliers in the model's predictions. This can help refine your model and your trading or investment strategies.

**Conclusion:**

The stock market plays a remarkable role in our daily lives. It is a significant factor in a country's GDP growth. In this tutorial, you learned the basics of the stock market and how to perform stock price prediction using machine learning.