Artificial Intelligence and Machine Learning

Project Report

Semester-IV (Batch-2022)

STOCK TREND PREDICTION SYSTEM

Supervised By:

Faculty Name : DR. KIRAN DEEP SINGH

Submitted By:

NAME:BHAVESH KHULLAR

ROLL NO.-2210990216

GROUP-G6

Department of Computer Science and Engineering
Chitkara University Institute of Engineering & Technology,
Chitkara University, Punjab

ABSTRACT:

Predicting Stock Trends Using LSTM Models in Python

In today's dynamic and unpredictable financial markets, accurate prediction of stock trends has become crucial for investors and traders to make informed decisions. This project delves into the development and deployment of a sophisticated Stock Trend Prediction Model using Long Short-Term Memory (LSTM) networks, implemented in Python. Traditional stock forecasting methods often struggle to capture the intricate patterns and temporal dependencies inherent in market data, limiting their predictive capabilities. However, LSTM models, a subset of recurrent neural networks (RNNs), excel at learning sequential patterns from time-series data, making them well-suited for stock market prediction tasks.

This project encompasses a comprehensive workflow, beginning with the collection and preprocessing of historical stock data and relevant technical indicators. Subsequently, extensive feature engineering is conducted to extract meaningful patterns and relationships from the data. The LSTM model is then meticulously designed and trained on the pre-processed dataset, leveraging its ability to capture long-term dependencies and nonlinear relationships in the input data.

To evaluate the efficacy of the developed model, rigorous testing and validation are performed using real-world stock market data.

Ultimately, this project aims to provide investors and financial analysts with a powerful tool for enhancing their decision-making processes in the stock market.

TABLE OF CONTENTS

INTRODUCTION:

In the fast-paced world of financial markets, the ability to predict stock trends accurately is a coveted skill sought after by investors and analysts alike. Traditionally, analysts have relied on statistical methods and fundamental analysis to forecast market movements. However, with the advent of deep learning techniques, particularly in the form of neural networks, a new era of predictive analytics has emerged.

Deep learning, a subset of machine learning, has shown remarkable promise in handling complex and unstructured data, making it well-suited for tasks such as stock market prediction. By leveraging neural networks with multiple layers, deep learning models can automatically learn intricate patterns and relationships in large datasets, potentially leading to more accurate predictions.

In this project, we delve into the realm of stock trend prediction using deep learning techniques implemented in Python. Our goal is to develop robust predictive models capable of analyzing historical stock data and forecasting future trends with a high degree of accuracy. By harnessing the power of deep learning algorithms, we aim to uncover hidden patterns and signals within the data that may elude traditional analysis methods.

INTRODUCTION:

BACKGROUND:

The realm of stock market prediction has long been characterized by its inherent volatility and complexity, presenting a formidable challenge for analysts and researchers alike. Traditional statistical approaches, while valuable, often struggle to capture the nuanced patterns and interdependencies present in financial data. However, with the emergence of machine learning methodologies, particularly within the domain of deep learning, there has been a notable shift in how stock trends are forecasted and analyzed.

These advanced techniques offer the potential to uncover hidden insights and patterns within vast datasets, enabling more accurate predictions and informed decision-making in the financial markets. Deep learning models, such as Long Short-Term Memory (LSTM) networks, have gained prominence for their ability to capture temporal dependencies and non-linear relationships in sequential data, making them particularly well-suited for time series forecasting tasks.

In recent years, LSTM models have demonstrated promising results in stock market prediction, outperforming traditional statistical methods in various scenarios. By leveraging the sequential nature of historical stock price data, LSTM networks can learn complex patterns and trends, allowing them to make more accurate predictions about future price movements.

INTRODUCTION:

OBJECTIVE:

The objective of a stock trend prediction system is to forecast the future movement of stock prices based on historical data, market indicators, and various analytical techniques.

- **Accurate Forecasting:** The primary objective is to accurately predict the direction and magnitude of future stock price movements. This helps investors make informed decisions about buying, selling, or holding stocks.

- **Risk Management:** By predicting stock trends, investors can better manage their investment risks. They can identify potential losses and take preventive measures to mitigate them.

- **Profit Maximization:** Investors aim to maximize their profits by buying stocks when they are expected to rise in value and selling them when they are expected to decline. A reliable prediction system assists in identifying profitable trading opportunities.

- **Decision Support:** The system should provide valuable insights and decision support tools to investors, enabling them to make well-informed investment decisions based on data-driven analysis rather than intuition or speculation.

- **Accessibility:** The system should be accessible to a wide range of users, from individual investors to institutional traders, and should be user-friendly with clear outputs and explanations.

PROBLEM DEFINITION AND REQUIREMENTS:

PROBLEM STATEMENT:

In today's dynamic and ever-changing financial markets, investors face the daunting challenge of predicting stock price movements accurately. The lack of a reliable method for forecasting stock trends often leads to missed investment opportunities and increased risk exposure.

The objective of this project is to develop a robust stock trend prediction system that leverages historical market data, advanced analytics techniques, and machine learning algorithms to forecast future stock price movements with enhanced accuracy.

- **Data Complexity:** Financial markets generate vast amounts of data, including historical stock prices, trading volumes, economic indicators, and company-specific metrics. Handling and analyzing this data efficiently to extract meaningful insights pose significant challenges.

- **Market Volatility:** Stock prices are influenced by a myriad of factors, including economic conditions, geopolitical events, and investor sentiment. Predicting stock trends in the face of market volatility requires models that can adapt to changing conditions and mitigate the impact of unpredictable events.

- **Model Accuracy:** Developing predictive models that can accurately forecast stock price movements is a complex task. Balancing model complexity with interpretability and generalization requires careful consideration to ensure reliable predictions.

PROBLEM DEFINITION AND REQUIREMENTS:

REQUIREMENTS:

Software Requirements:

- **Jupyter Notebook:** The project will be implemented using Jupyter Notebook for ease of code execution, visualization, and documentation.
- **Python Libraries:** The following Python libraries will be utilized for data manipulation, analysis, and machine learning model development:
- **Pandas:** For data manipulation and analysis.
- **NumPy:** For numerical computations.
- **Scikit-learn:** For implementing machine learning algorithms.
- **Matplotlib:** For data visualization.
- **Anaconda Distribution:** Anaconda will be used as the primary Python distribution to ensure seamless integration of required libraries and easy package management.

Hardware Requirements:

Since the dataset and model training do not demand extensive computational resources, the project can be executed on standard computing hardware. However, to ensure smooth performance, it is recommended to have:

- A computer with at least 4GB of RAM.
- Sufficient storage space for dataset storage and model files.

METHODOLOGY:

- **Data Collection:**

  Gather historical stock price data from reliable sources such as financial databases, APIs, or online repositories. The dataset should include relevant features such as opening price, closing price, highest price, lowest price, trading volume, and any other indicators that may influence stock prices.

- **Data Preprocessing:**

  Clean the data by handling missing values, outliers, and inconsistencies.

  Normalize the data to ensure all features are on a similar scale, typically between 0 and 1. This helps the model converge faster during training.

  Split the data into training, validation, and test sets. The training set is used to train the model, the validation set is used to tune hyperparameters and prevent overfitting, and the test set is used to evaluate the model's performance.

- **Model Architecture:**

  Design the LSTM architecture, specifying the number of LSTM layers, the number of units (neurons) in each layer, and any additional layers such as dropout or batch normalization.

  Configure the input shape based on the number of time steps (historical data points) and the number of features.

  Compile the model, specifying the loss function (e.g., mean squared error for regression tasks), optimizer (e.g., Adam), and evaluation metrics (e.g., mean absolute error, mean squared error).

- **Model Evaluation:**

  Evaluate the trained model using the test set to assess its performance on unseen data.

  Calculate evaluation metrics such as mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE) to quantify the model's accuracy.

  Visualize the model's predictions against the actual stock prices to gain insights into its performance and identify any areas for improvement.

- **Model Interpretation and Analysis:**

  Analyze the LSTM model's predictions and identify any patterns or anomalies in the forecasted stock prices.

  Interpret the importance of input features and assess their contribution to the model's predictions.

  Conduct sensitivity analysis to evaluate the robustness of the model to changes in input parameters and features.

- **Continuous Improvement and Monitoring:**

  Monitor the LSTM model's performance over time and retrain it periodically with updated data to ensure continued accuracy and relevance.

  Incorporate feedback from users and stakeholders to refine the model and address any limitations or shortcomings.

  Stay updated with advancements in deep learning techniques and research to explore new methodologies for improving stock price prediction accuracy**.**

RESULT:

The result of the stock trend prediction system would ideally be twofold:

- **Accurate Predictions:** The primary result is generating accurate predictions of future stock price movements. The system would provide forecasts indicating whether a stock's price is likely to increase, decrease, or remain stable over a given time horizon. These predictions would enable investors to make informed decisions about buying, selling, or holding stocks, thereby maximizing returns and minimizing risks.

- **Decision Support:** Beyond predictions, the system would offer decision support tools and insights to assist investors in interpreting the forecasts and making strategic investment decisions. This could include identifying key trends, risk assessment metrics, and recommended actions based on the predicted outcomes. Ultimately, the result would be empowering investors with the information they need to navigate the complexities of the financial markets with confidence and efficiency.

RESULT:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#import pandas_datareader as data
```

[3] ✓ 3.7s Python

```
start = '2010-01-04'
end = '2019-12-30'

df = pd.read_csv('C:\\Users\\CHAITANYA\\Downloads\\AAPL.csv')
df.head()
```

[4] ✓ 0.0s Python

	Date	Open	High	Low	Close	Adj Close	Volume
0	2010-01-04	7.622500	7.660714	7.585000	7.643214	6.461978	493729600
1	2010-01-05	7.664286	7.699643	7.616071	7.656429	6.473148	601904800
2	2010-01-06	7.656429	7.686786	7.526786	7.534643	6.370186	552160000
3	2010-01-07	7.562500	7.571429	7.466071	7.520714	6.358408	477131200
4	2010-01-08	7.510714	7.571429	7.466429	7.570714	6.400681	447610800

+ Code + Markdown

```
df = df.drop(['Date', 'Adj Close'], axis = 1)
df.head()
```

✓ 0.0s Python

	Open	High	Low	Close	Volume
0	7.622500	7.660714	7.585000	7.643214	493729600
1	7.664286	7.699643	7.616071	7.656429	601904800
2	7.656429	7.686786	7.526786	7.534643	552160000
3	7.562500	7.571429	7.466071	7.520714	477131200
4	7.510714	7.571429	7.466429	7.570714	447610800

```python
ma100 = df.Close.rolling(100).mean()
ma100
```

✓ 0.0s Python

```
0            NaN
1            NaN
2            NaN
3            NaN
4            NaN
          ...
2510    59.201075
2511    59.401700
2512    59.643125
2513    59.875125
2514    60.106325
Name: Close, Length: 2515, dtype: float64
```

```python
ma200 = df.Close.rolling(200).mean()
ma200
```

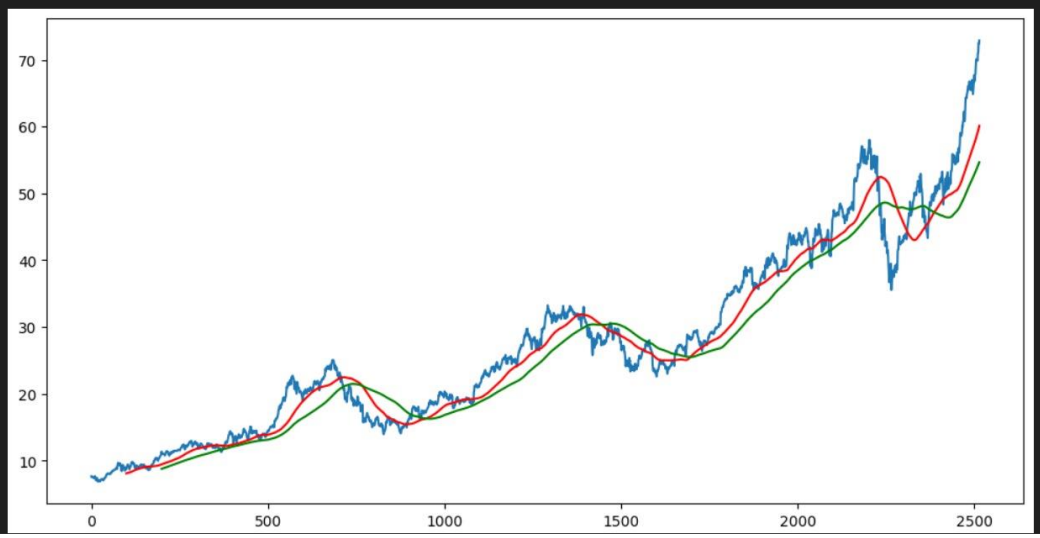
✓ 0.0s Python

```
0            NaN
1            NaN
2            NaN
3            NaN
4            NaN
          ...
2510    54.132313
2511    54.261513
2512    54.396763
2513    54.529350
```

```python
plt.figure(figsize = (12,6))
plt.plot(df.Close)
plt.plot(ma100, 'r')
plt.plot(ma200, 'g')
```

✓ 0.1s Python

```
[<matplotlib.lines.Line2D at 0x1365fb524d0>]
```



```python
df.shape
```

✓ 0.0s Python

```
(2515, 5)
```

```python
data_training = pd.DataFrame(df['Close'][0:int(len(df)*0.70)])
data_testing = pd.DataFrame(df['Close'][int(len(df)*0.70): int(len(df))])

print(data_training.shape)
print(data_testing.shape)
```

✓ 0.0s Python

```
(1760, 1)
(755, 1)
```

```python
data_training.head()
```

✓ 0.0s Python

	Close
0	7.643214
1	7.656429
2	7.534643
3	7.520714
4	7.570714

```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0,1))
```

✓ 4.4s Python

```python
data_training_array = scaler.fit_transform(data_training)
data_training_array
```

✓ 0.0s Python

```
array([[0.02971782],
       [0.03021855],
       [0.02560389],
       ...,
       [0.84388656],
       [0.85089658],
       [0.84616013]])
```

```python
x_train = []
y_train = []

for i in range(100, data_training_array.shape[0]):
    x_train.append(data_training_array[i-100: i])
    y_train.append(data_training_array[i, 0])

x_train , y_train = np.array(x_train), np.array(y_train)
```

✓ 0.0s Python

```python
x_train.shape
```

✓ 0.0s Python

```
(1660, 100, 1)
```

```python
from keras.layers import Dense, Dropout, LSTM
from keras.models import Sequential
```

✓ 10.8s Python

```python
model = Sequential()

model.add(LSTM(units = 50, activation = 'relu', return_sequences = True, input_shape = (x_train.shape[1], 1)))
model.add(Dropout(0.2))

model.add(LSTM(units = 60, activation = 'relu', return_sequences = True))
model.add(Dropout(0.3))

model.add(LSTM(units = 80, activation = 'relu', return_sequences = True))
model.add(Dropout(0.4))

model.add(LSTM(units = 120, activation = 'relu'))
model.add(Dropout(0.5))

model.add(Dense(units = 1))
```

✓ 0.2s Python

```python
model.summary()
```

✓ 0.0s Python

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 100, 50)	10,400
dropout (Dropout)	(None, 100, 50)	0
lstm_1 (LSTM)	(None, 100, 60)	26,640
dropout_1 (Dropout)	(None, 100, 60)	0
lstm_2 (LSTM)	(None, 100, 80)	45,120
dropout_2 (Dropout)	(None, 100, 80)	0
lstm_3 (LSTM)	(None, 120)	96,480
dropout_3 (Dropout)	(None, 120)	0
dense (Dense)	(None, 1)	121

Total params: 178,761 (698.29 KB)

Trainable params: 178,761 (698.29 KB)

Non-trainable params: 0 (0.00 B)

```
model.compile(optimizer = 'adam', loss = "mean_squared_error")
hist = model.fit(x_train, y_train, epochs = 50)
```

✓ 4m 19.3s Python

```
Epoch 1/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 12s 112ms/step - loss: 0.1310
Epoch 2/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 6s 106ms/step - loss: 0.0158
Epoch 3/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 6s 115ms/step - loss: 0.0122
Epoch 4/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 5s 99ms/step - loss: 0.0107
Epoch 5/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 5s 96ms/step - loss: 0.0102
Epoch 6/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 5s 94ms/step - loss: 0.0107
Epoch 7/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 5s 94ms/step - loss: 0.0074
Epoch 8/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 5s 92ms/step - loss: 0.0077
Epoch 9/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 5s 96ms/step - loss: 0.0081
Epoch 10/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 5s 92ms/step - loss: 0.0089
Epoch 11/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 5s 92ms/step - loss: 0.0078
Epoch 12/50
52/52 ━━━━━━━━━━━━━━━━━━━━ 5s 94ms/step - loss: 0.0067
Epoch 13/50
...
```

```
model.save('keras_model.h5')
```

✓ 0.0s Python

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We
```

```
data_testing.head()
```

✓ 0.0s Python

	Close
1760	29.182501
1761	28.955000
1762	29.037500
1763	29.004999
1764	29.152500

```
past_100_days = data_training.tail(100)
```

✓ 0.0s Python

```
final_df = pd.concat([past_100_days, data_testing], ignore_index=True)
```

✓ 0.0s Python

```
input_data = scaler.fit_transform(final_df)
```
✓ 0.0s Python

```
input_data
```
✓ 0.0s Python

Outputs are collapsed ···

```
input_data.shape
```
✓ 0.0s Python

```
(855, 1)
```

```
x_test = []
y_test = []

for i in range(100, input_data.shape[0]):
    x_test.append(input_data[i-100: i])
    y_test.append(input_data[i, 0])
```
✓ 0.0s Python

```
x_test, y_test = np.array(x_test), np.array(y_test)
print(x_test.shape)
print(y_test.shape)
```
✓ 0.0s Python

```
y_predicted = model.predict(x_test)
```
✓ 1.7s Python

```
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step
```

+ Code + Markdown

```
y_predicted.shape
```
✓ 0.0s Python

```
(755, 1)
```

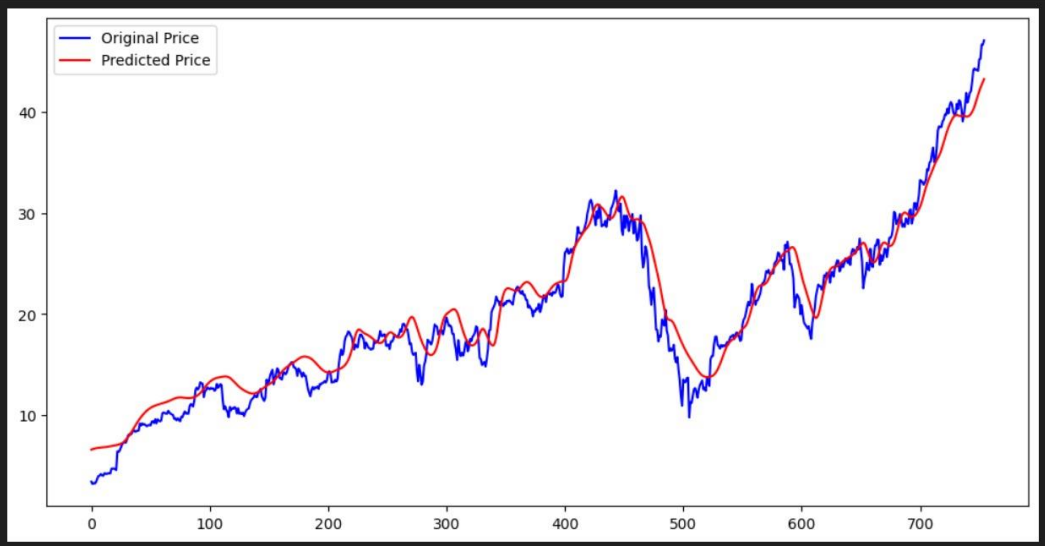
```
scaler.scale_
```
✓ 0.0s Python

```
array([0.02123255])
```

```
scale_factor = 1/0.02123255
y_predicted = y_predicted*scale_factor
y_test = y_test*scale_factor
```
✓ 0.0s Python

```
plt.figure(figsize=(12,6))
plt.plot(y_test, 'b', label = 'Original Price')
plt.plot(y_predicted, 'r', label = 'Predicted Price')
plt.legend()
plt.show()
```

✓ 0.1s Python



REFERENCES

- **https://www.geeksforgeeks.org/**
- **https://www.simplilearn.com/**