2022

FE 520 FINAL REPORT

DEVIKA

SUDARSHANA

TEJAS

TABLE OF CONTENTS

ABSTRACT

Predicting stock prices using machine learning techniques has gained significant attention in recent years due to the potential to accurately forecast market trends and make informed investment decisions. There are various approaches to using machine learning for stock price prediction, including using historical stock data and financial news articles as input features to train predictive models. Some popular techniques include decision tree algorithms, support vector machines, and deep learning models such as long short-term memory (LSTM) networks.

One key challenge in predicting stock prices is the inherent volatility and noise in financial markets. This can make it difficult to accurately forecast short-term price movements, and it is often more practical to focus on longer-term trends. Another challenge is the need to properly feature engineering, which involves selecting the most relevant and meaningful input data for the model. In addition, the use of machine learning for stock price prediction is often accompanied by ethical concerns, such as the potential for insider trading and the impact on market stability.

Overall, the use of machine learning for stock price prediction has the potential to improve investment decision-making and provide valuable insights into market trends. However, it is important to carefully consider the limitations and ethical implications of these approaches and to approach stock price prediction with caution.

METHODOLOGY

For our machine learning model, we used a dataset that contained information about the financial sector, including stock prices. We specifically chose a stock dataset for an Indian company called Reliance Industries. The dataset included multiple variables, such as date, open price, high price, low price, close price, and various statistics related to trading volume and turnover. Our target variable, or the variable we wanted to predict, was the 'close' price, which represents the final value of the stock at the end of the day. As the other variables were not known ahead of time, we only used the date as our input feature. We split the date into multiple features, including the year, month, week, day, and day of the week, to capture potentially relevant trends in the stock price. We included the day of the week as a feature because it can influence the value of the stock, with certain days (such as Mondays and Fridays) potentially having different trends compared to the rest of the week.

DATA

We used pandas to handle missing data, perform operations on columns and rows, and transform data.

NumPy is used for performing mathematical calculations, especially with large datasets.

```
import os
import pandas as pd
import numpy as np
```

After importing the dataset, we discarded the unimportant columns

```
df = mydata.copy()
df = df.iloc[:,[0,4]]
print(df.columns)
```

```
Index(['Date', 'Close'], dtype='object')
```

Train and Test datasets

In machine learning, it is common to split a dataset into a training set and a test set. The training set is used to build and train a model, while the test set is used to evaluate the performance of the model.

```
train = datafull[:3655]
test = datafull[3655:]
```

```
      Year  Month  Week  Day  Dayofweek
0     2002     12     1   31          1
1     2003      1     1    1          2
2     2003      1     1    2          3
3     2003      1     1    3          4
4     2003      1     2    6          0
...    ...    ...   ...  ...        ...
3650  2017      8    35   30          2
3651  2017      8    35   31          3
3652  2017      9    35    1          4
3653  2017      9    36    4          0
3654  2017      9    36    5          1

[3655 rows x 5 columns]
0        297.70
1        297.60
2        293.00
3        287.90
4        284.85
         ...
3650    1564.15
3651    1593.50
3652    1609.35
```

For our model, the dataset is split into two parts train and test. The train set is from 2003 to 2018 and the test set is from 2018 to 2020. The machine learning model is trained upon the train set and the performance of the model is tested on the test set.

LIBRARIES

```
import os
import pandas as pd
import numpy as np
import scipy.sparse
import matplotlib.pyplot as plt
from fastai.torch_basics import *
from fastai.data.all import *
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM
```

We imported the following libraries

panda and NumPy for data handling, matplotlib for data visualization, and fastai, sklearn, and Keras for machine learning.

MODELS

1. Linear Regression: We have implemented a linear regression model on the data. The linear regression model returns an equation that determines the relationship between the independent variables and the dependent variable. The equation for linear regression:

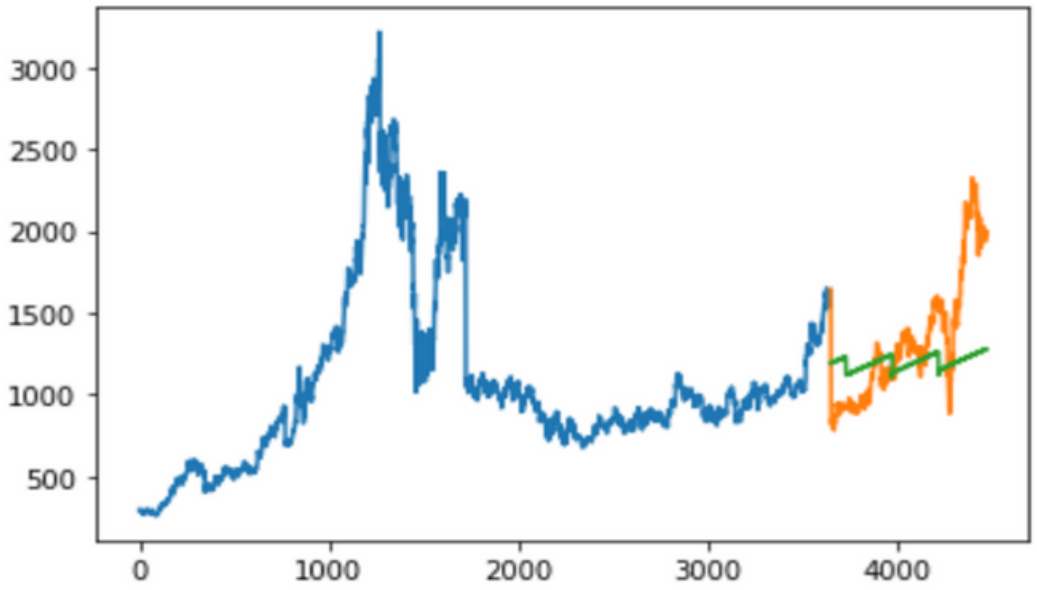
$$Y = \theta_1 X_1 + \theta_2 X_2 + \dots \theta_n X_n$$

```
#model1 linear
model = LinearRegression()
model.fit(x_train,y_train)
preds = model.predict(x_test)
rms=np.sqrt(np.mean(np.power((np.array(y_test)-np.array(preds)),2)))
print(rms)
```

```
383.14844736352217
```

The root mean square value(rmse) obtained from linear regression is: 383.148

The plot for linear regression model



2. K nearest neighbor

KNN does not build a model or make any assumptions about the underlying data distribution. Instead, it simply stores the training data and makes predictions by finding the nearest neighbors at the prediction time. This makes KNN relatively simple to implement and fast for small datasets, but it can be computationally expensive for larger datasets.

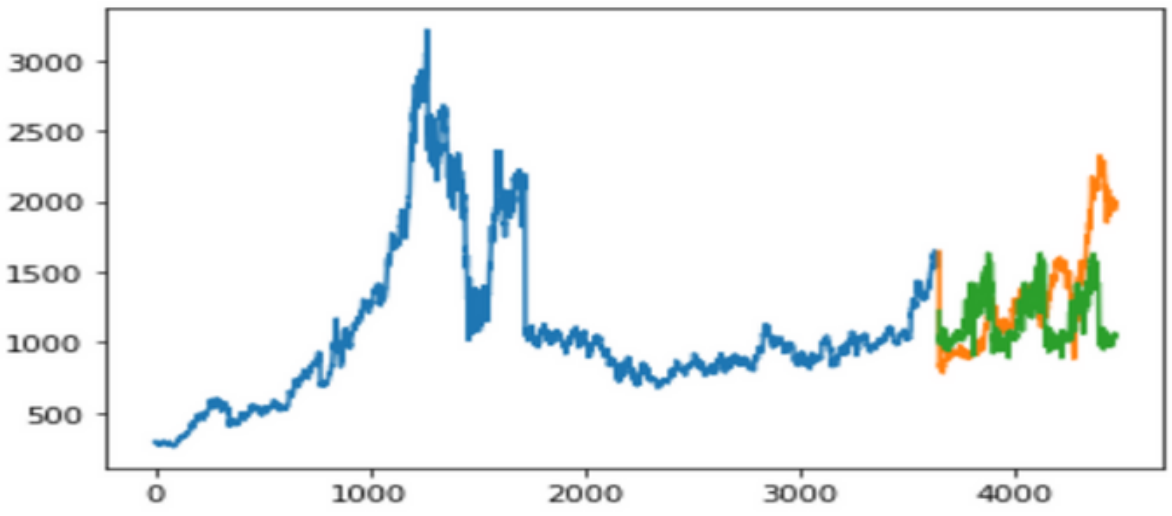
```
# model2 knn
model2 = KNeighborsRegressor(n_neighbors = 3)
model2.fit(x_train,y_train)
preds2 = model2.predict(x_test)
rms2=np.sqrt(np.mean(np.power((np.array(y_test)-np.array(preds2)),2)))
print(rms2)
```

```
434.9206381887921
```

The root mean square error(rmse) value obtained from KNN is 434.9206

The following is the plot for KNN model

```
test['Predictions'] = 0
test['Predictions'] = preds2

test.index = datafull[3655:].index
train.index = datafull[:3655].index

plt.plot(train['Close'])

plt.plot(test[['Close', 'Predictions']])
```



Support Vector Regression Model :

We tried using the Support Vector Regression Model but as the Support Vector Regressor is not good, hence we decided not to proceed with this model.

```
plt.plot(train['Close'])

plt.plot(test[['Close', 'Predictions']])

svr = SVR(kernel='rbf', C=1e3, gamma=0.1)
svr.fit(x_train, y_train)
svm_confidence = svr.score(x_test, y_test)
print('svm confidence: ', svm_confidence)
```

```
svm confidence:  -0.7860300473684487
```

3. Long Short Term Memory

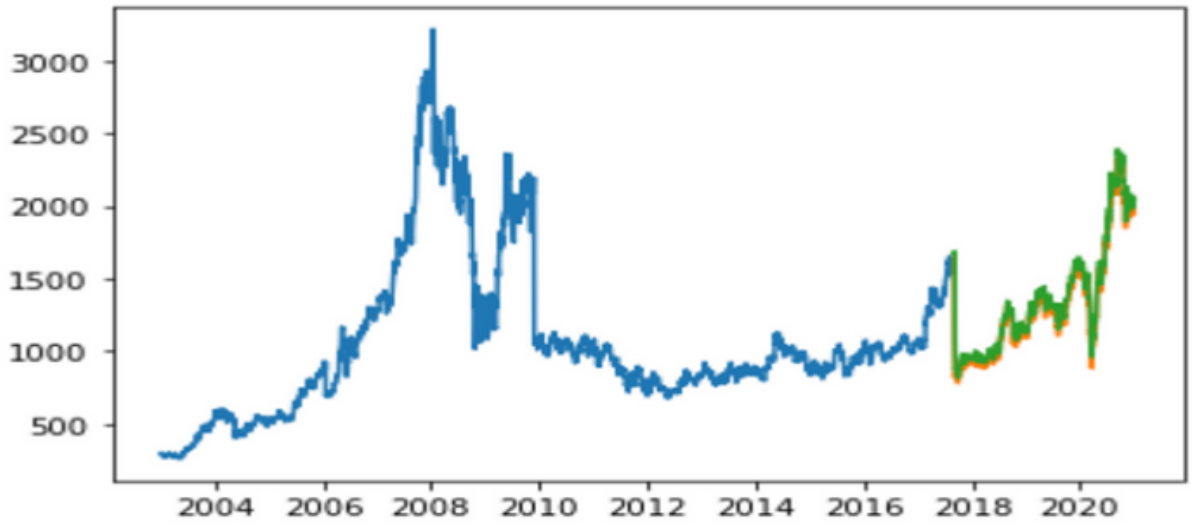
Long short-term memory (LSTM) is a type of recurrent neural network (RNN) that is particularly well-suited for modeling sequential data, such as time series or natural language. LSTMs are designed to remember and preserve long-term dependencies in data by using "memory cells" that can store information over long periods of time. They also have "gates" that can control the flow of information into and out of the memory cells, allowing them to selectively preserve or forget certain information.

The input gate: The input gate adds information to the cell state
The forget gate: It removes the information that is no longer required by the model
The output gate: Output Gate at LSTM selects the information to be shown as output

```
#predicting 246 values, using past 60 from the train data
inputs = df[len(df) - len(valid) - 60:].values
inputs = inputs.reshape(-1,1)
inputs  = scaler.transform(inputs)

X_test = []
for i in range(60,inputs.shape[0]):
    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))
closing_price = model3.predict(X_test)
closing_price = scaler.inverse_transform(closing_price)


rms3=np.sqrt(np.mean(np.power((valid-closing_price),2)))
```

The root mean square value obtained from LSTM is 61.267921

The plot obtained from LSTM model



RESULTS

•Linear regression model required no parameter tuning. The model is a very simple model and hence performed very poorly on the dataset. We got a loss of over 300 which is very poor for any model

•For the KNN model we performed parameter tuning, we found the best value for the number of neighbours is 6. This model also performed poorly and we got a loss of over 400. It was surprising to see this model perform worse than linear regression which is a much simpler model.

•We used LSTM as it is a popular model used for stock price predictions. We set the past to up to 60 training values for this model. To avoid over fitting we only used 2 layers each containing 50 units. After that we passed it into a dense layer to flatten it and used the adam optimizer. We ran it for 10 epochs and kept the batch size 1, thus using stochastic gradient descent. As expected, this model performed the best and gave us a loss less than 0.01