REPORT FOR STOCK SENTIMENT ANALYSIS USING MACHINE LEARNING TECHNIQUES

AKSHAT SHARMA

Enrollment No. : 22117014

ELECTRICAL ENGINEERING(3rd yr.)

Overview:

A BRIEF INTRODUCTION TO STOCK SENTIMENT ANALYSIS:

Sentiment analysis is the process of analyzing digital text to determine if the emotional tone of the message is positive, negative, or neutral.Stock sentiment analysis involves collecting news articles,tweets, etc related to specific stocks and then using machine learning models to predict the movement of markets with the goal of trying to develop a trading strategy to maximize our profits.Sentiment analysis also assesses how geopolitical developments, economic conditions, or the latest corporate affairs will affect market sentiment.It is an effective method for risk mitigation in response to major or even minor events in the market which otherwise might lead to drastic losses.Its a well known fact that sentiment of the traders can provide us valuable insights to make trading decisions.

APIs and Libraries used:

1. NYT(New York Times) Archives API
2. Pandas
3. Numpy
4. TextBlob
5. Requests
6. Sci-kit learn
7. Matplotlib
8. Yfinance API

Procedure:

1. COLLECTING THE NEWS HEADLINES:

   The news headlines for three stocks namely -:APPLE,MICROSOFT AND AMAZON have been extracted from the nyt archives using its api.

2. DATA PREPROCESSING:

   The news headlines extracted are then cleaned.The headlines are all converted into lowercase and all the non-alphanumeric characters are removed.

3. SENTIMENT SCORES:

The headlines are assigned polarity and subjectivity scores by taking help from the TextBlob library.

4. TRAINING THE MODEL:

   A machine learning model is trained on the acquired dataset.

5. EVALUATING THE MODEL:

   The model is evaluated based on its predictions and tested on the test data which was separated earlier.

6. TRADING STRATEGY:

   A trading strategy involving predicting the buy and sell points is introduced on the basis of the predictions made by the model.

7. EVALUATING THE STRATEGY:

   The strategy is evaluated using financial metrics such as sharpe ratio, maximum drawdowns, number of trades executed and win ratio.

8. VISUALIZATION:

   The final portfolio value is calculated and the graphs with buy and sell points are plotted .

Some prior information:

The initial investment is assumed to be $100000 and we have collected the data from the year 2017 to the present year 2024(month of May).

CODE:

1. IMPORTING REQUIRED LIBRARIES:

```
import requests
import pandas as pd
import yfinance as yf
import time
from urllib3.util.retry import Retry
from requests.adapters import HTTPAdapter
import pandas as pd
import re
import pandas as pd
from sklearn.ensemble import VotingClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve
import matplotlib.pyplot as plt

from sklearn.metrics import roc_curve, auc
import numpy as np
```

- This section of the python code just involves importing all the libraries which have been used in the program.

2. COLLECTING NEWS ARTICLES:

```
api_key = "Tar1vB6RJGhcZflwTq2RS4hxeBZIcLFC"
url = 'https://api.nytimes.com/svc/archive/v1/{}/{}.json?api-key={}'
stocks = ["apple", "microsoft", "amazon"]
data = []
date = []
stock_list = []

# Create a session with retry strategy
session = requests.Session()
retry_strategy = Retry(
    total=5,
    backoff_factor=10,
    status_forcelist=[429, 500, 502, 503, 504],
    allowed_methods=["GET"]
)
adapter = HTTPAdapter(max_retries=retry_strategy)
session.mount("https://", adapter)
session.mount("http://", adapter)

a=0
```

```
for stock in stocks:
    for year in range(2017, 2025):
        for month in range(1, 13):
            print(a)
            a=a+1
            if(month==6 and year==2024):
                break
            try:
                response = session.get(url.format(year, month, api_key))
                response.raise_for_status()
                data_json = response.json()
                articles = data_json['response']['docs']
                sto_article = []
                for article in articles:
                    headline = article.get('headline', {}).get('main', '').lower()
                    snippet = article.get('snippet', '').lower()
                    if stock in headline or stock in snippet:
                        sto_article.append(article)

                for article in sto_article:
                    stock_list.append(stock.upper())
                    headline = article['headline']['main']
                    date_time = article['pub_date']
                    date_only = pd.to_datetime(date_time).date()
                    data.append(headline)
                    date.append(date_only)


                time.sleep(12)

            except:
                print("ERROR")


df = pd.DataFrame({
    'Date': date,
    'Stock': stock_list,
    'Headline': data
})

print("end")
```

- In this section of the code, a call is made to the NYT Archives API, an official API provided by the New York Times. This API allows retrieval of all articles published in a specific year and month.
- A retry strategy is then implemented. This is crucial because the server may occasionally fail to process requests due to various reasons.
- The delay between retries increases with each new retry.
- Subsequently, iteration is performed through all stocks and the entire time period to gather all news headlines during that period. Initially, the entire data is stored in the form of lists.
- A sleep time of 12 seconds is specified after each call, as the NYT API only allows 5 requests per minute.
- Finally, a DataFrame is created using the data.
- Note that the value of ‘a’ is printed and incremented to monitor the program's progress and ensure its proper execution.

3. DATA PREPROCESSING:

```
def clean_text(text):
    # Remove all non-alphanumeric characters
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    return text.lower()

final_df=df.groupby(["Date","Stock"])["Headline"].apply(lambda x: " ".join(x.astype(str))).reset_index()
final_df["Headline"]=final_df["Headline"].apply(clean_text)

print(final_df.head(10))
```

- In this section, a function is created which removes any non-alphanumeric character from the news headlines.The text is converted to lowercase.This is done to maintain uniformity in the data.
- The dataframe is then grouped by the 'Date' and the 'Stock' columns.The lambda function joins all the headlines in each group into a single string, separated by spaces.
- Finally the DataFrame is stored in a new dataframe called "final_df" and the first 10 rows are printed to make sure that we have got our desired results.

4. ASSIGNING SENTIMENT SCORES TO TEXT:

```
from textblob import TextBlob as tb

final_df['Polarity_score']=float(0)
final_df['Subjectivity_score']=float(0)

for i in range(0,len(final_df['Headline'])):
    score=tb(final_df['Headline'][i])
    final_df.at[i,'Polarity_score']=float(score.sentiment[0])
    final_df.at[i,'Subjectivity_score']=float(score.sentiment[1])



print(final_df.head())
```

- The textblob library is imported and it is used to give polarity and subjectivity scores to the news headlines.

5. LABELING STOCK PRICE MOVEMENTS:

```
final_df["Price Movement"]=-1
final_df['Open']=-1
final_df['Close']=-1


for i in range(0,len(final_df['Date'])):
    ticker=''
    if(final_df['Stock'][i]=='APPLE'):
        ticker='AAPL'
    elif(final_df['Stock'][i]=='MICROSOFT'):
        ticker='MSFT'
    else:
        ticker="AMZN"

    date=final_df['Date'][i]
    try:
        data = yf.download(ticker, start=date, end=pd.to_datetime(date) + pd.DateOffset(1))
    except:
       continue


    if not data.empty:
        if(data.iloc[0]['Open']<data.iloc[0]['Close']):
            final_df.at[i,'Price Movement']=1

        else:
            final_df.at[i,'Price Movement']=0
        final_df.at[i,'Open']=data.iloc[0]['Open']
        final_df.at[i,'Close']=data.iloc[0]['Close']

final_df=final_df[final_df['Price Movement']!=-1]


print(final_df.head())
```

- The Yfinance api is used to get the opening and closing prices for each stock on a particular date.
- Some of the news headlines might have been pertaining to the days on which the stock market was closed.

- So, initially the new columns have been designated the value (-1) . This has been done so that the days on which the stock market was closed can be removed from the dataset in the end as shown in the code as well.

6. TRAINING THE MACHINE LEARNING MODEL:

```
final_df['Date'] = pd.to_datetime(final_df['Date'])

# Ensure data is sorted by date
final_df = final_df.sort_values('Date')

# Split data for each stock
stocks = ["APPLE","AMAZON","MICROSOFT"]
stock_data = {}

# Loop through each stock symbol in the 'stocks' lis
for stock in stocks:

    req_df = final_df[final_df['Stock'] == stock]


    stock_data[stock] = req_df.copy()

# Define features and target
features = ['Subjectivity_score', 'Polarity_score']
target = 'Price Movement'
```

```
def train_and_predict(stock_df):
    split = int(len(stock_df) * 0.8)  # 80 percent data is for training
    train_df = stock_df[0:split]
    test_df = stock_df[split:len(stock_df)]

    x_train = train_df[features]
    y_train = train_df[target]
    x_test = test_df[features]
    y_test = test_df[target]

    # Initialize the models
    log_reg = LogisticRegression()
    svm = SVC(probability=True)
    rf = RandomForestClassifier()

    # Create the ensemble model using VotingClassifier
    ensemble_model = VotingClassifier(estimators=[
        ('lr', log_reg),
        ('svm', svm),
        ('rf', rf)],
        voting='soft'
    )

    # Train the ensemble model
    ensemble_model.fit(x_train, y_train)

    # Make predictions
    y_predicted = ensemble_model.predict(x_test)
    y_pred_prob = ensemble_model.predict_proba(x_test)[:, 1]

    # Store predictions in the dataframe
    stock_df.loc[test_df.index, 'Predicted Prob'] = y_pred_prob
    stock_df.loc[test_df.index, 'Predicted Movement'] = y_predicted

    return stock_df, test_df.index

# Train and predict for each stock
test_indices = {}
for stock in stocks:
    stock_data[stock], test_indices[stock] = train_and_predict(stock_data[stock])
```

- The data is first sorted by date.The features and the target for the ML model are defined.

- 80 percent data is used for training the model and the rest 20 percent is used for testing the model and acquiring the various metrics associated with it.
- Ensemble method comprising of Logistic Regression,SVC and Random Forest is used for better and more accurate predictions.

7. BUILDING TRADING STRATEGY:

```
def simulate_trading(stock_data, test_indices, initial_capital):
    capital = initial_capital
    positions = {}
    for stock in stocks:
       positions[stock] = 0


    buy_prices = {}
    for stock in stocks:
        buy_prices[stock] = 0
    num_trades = 0
    num_wins = 0
    returns = []
    capital_curve = [capital]

    # To create a combined dataframe for all stocks
    dfs=[]
    for stock in stocks:
        dfs.append(stock_data[stock])
    combined_df=pd.concat(dfs)
    combined_df=combined_df.sort_values('Date')

    combined_df = combined_df[combined_df.index.isin(np.concatenate(list(test_indices.values())))]

    for index, row in combined_df.iterrows():
        stock = row['Stock']
        if row['Predicted Movement'] == 1:  # buy signal
            if positions[stock] == 0 and capital > 0 and row['Close']!=0:
                positions[stock] = capital / row['Close']
                buy_prices[stock] = row['Close']
                capital = 0
                num_trades += 1
                stock_data[stock].at[index, 'Action'] = 'Buy'
```

```
        elif row['Predicted Movement'] == 0:  # Sell signal
            if positions[stock] > 0:
                capital = positions[stock] * row['Close']  # Sell all shares
                profit = capital - (positions[stock] * buy_prices[stock])
                returns.append(profit / (positions[stock] * buy_prices[stock]))
                capital_curve.append(capital)
                if row['Close'] > buy_prices[stock]:
                    num_wins += 1
                positions[stock] = 0
                stock_data[stock].at[index, 'Action'] = 'Sell'

    #To calculate the final capital
    for stock in stocks:
        if positions[stock] > 0:
            capital =capital+ positions[stock] * stock_data[stock].iloc[-1]['Close']
            positions[stock] = 0

    capital_curve.append(capital)

    return stock_data, capital_curve, returns

# Simulate trading for the combined portfolio
initial_capital = 100000
total_capital = 100000

all_true_labels = []
all_pred_probs = []
all_returns = []
all_capital_curves = []
all_predictions = []

stock_data, capital_curve, returns = simulate_trading(stock_data, test_indices, initial_capital)
total_capital =total_capital+ capital_curve[-1] - initial_capital
all_returns.extend(returns)
all_capital_curves.extend(capital_curve)

for stock in stocks:
    all_predictions.extend(stock_data[stock].loc[test_indices[stock], 'Predicted Movement'])
    all_true_labels.extend(stock_data[stock].loc[test_indices[stock], target])
    all_pred_probs.extend(stock_data[stock].loc[test_indices[stock], 'Predicted Prob'])
```

- A trading strategy involving predicting buy and sell points is made on the basis of the sentiment analysis done earlier.
- As said earlier the initial capital is taken to be $100000 and the final portfolio value is calculated subsequently.
- Also, note that the strategy has been tested on the test dataset which was separated earlier.The strategy is tested on the time period between April 2022 to May 2024 as depicted in the stock price charts plotted in the end.

8. PERFORMANCE METRICS:

- The final performance metrics are calculated and displayed.These are used to determine the accuracy,precision,recall etc. for the entire portfolio.The success of

the trading strategy deployed can be measured with these metrics.

- This can easily be achieved with the code given below.

```
sharpe_ratio = np.mean(all_returns) / np.std(all_returns) * np.sqrt(252)
drawdowns = np.maximum.accumulate(all_capital_curves) - all_capital_curves
max_drawdown = np.max(drawdowns)
num_trades = len(all_returns)
num_wins = sum(np.array(all_returns) > 0)
win_ratio = num_wins / num_trades if num_trades > 0 else 0

# Calculate accuracy, precision, recall, F1-score for the entire portfolio
accuracy = accuracy_score(all_true_labels, all_predictions)
precision = precision_score(all_true_labels, all_predictions)
recall = recall_score(all_true_labels, all_predictions)
f1 = f1_score(all_true_labels, all_predictions)

# Calculate ROC Curve and AUC for the combined predictions
fpr, tpr, _ = roc_curve(all_true_labels, all_pred_probs)
roc_auc = roc_auc_score(all_true_labels, all_pred_probs)

print('Final Capital:', {total_capital})
print('Profit:', total_capital - initial_capital)
print("Profit Percentage:", (total_capital - initial_capital) / initial_capital * 100)
print('Sharpe Ratio:', sharpe_ratio)
print('Maximum Drawdown:',max_drawdown)
print('Number of Trades Executed:',num_trades)
print('Win Ratio:', win_ratio)
print("AUC:", roc_auc)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

9. PLOTTING THE CHARTS:

```
plt.figure(figsize=(12, 6))
for stock in stocks:
    test_df = stock_data[stock].loc[test_indices[stock]]
    plt.plot(test_df['Date'], test_df['Close'], label=f'{stock} Close Price', alpha=0.5)
    buys = test_df[test_df['Action'] == 'Buy']
    sells = test_df[test_df['Action'] == 'Sell']
    plt.scatter(buys['Date'], buys['Close'], label=f'{stock} Buy Signal', marker='^', color='dark green', alpha=1)#buy marker
    plt.scatter(sells['Date'], sells['Close'], label=f'{stock} Sell Signal', marker='v', color='red', alpha=1)#sell marker
plt.title('Stock Prices with Buy/Sell Signals (Testing Period)')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.show()

# Plot Cumulative Capital Curve
plt.figure(figsize=(12, 6))
plt.plot(all_capital_curves, label='Capital Curve',color='orange')
plt.title('Cumulative Capital Curve')
plt.xlabel('Number of Trades executed')
plt.ylabel('Capital')
plt.legend()
plt.show()

# Plot ROC Curve
plt.figure(figsize=(12, 6))
plt.plot(fpr, tpr, color='red', lw=1, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='blue', lw=1, linestyle='-.')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

- This is the last section of the program.All the essential charts involving buy and

sell signals are plotted with the help of this code.

RESULTS

1. PERFORMANCE METRICS:

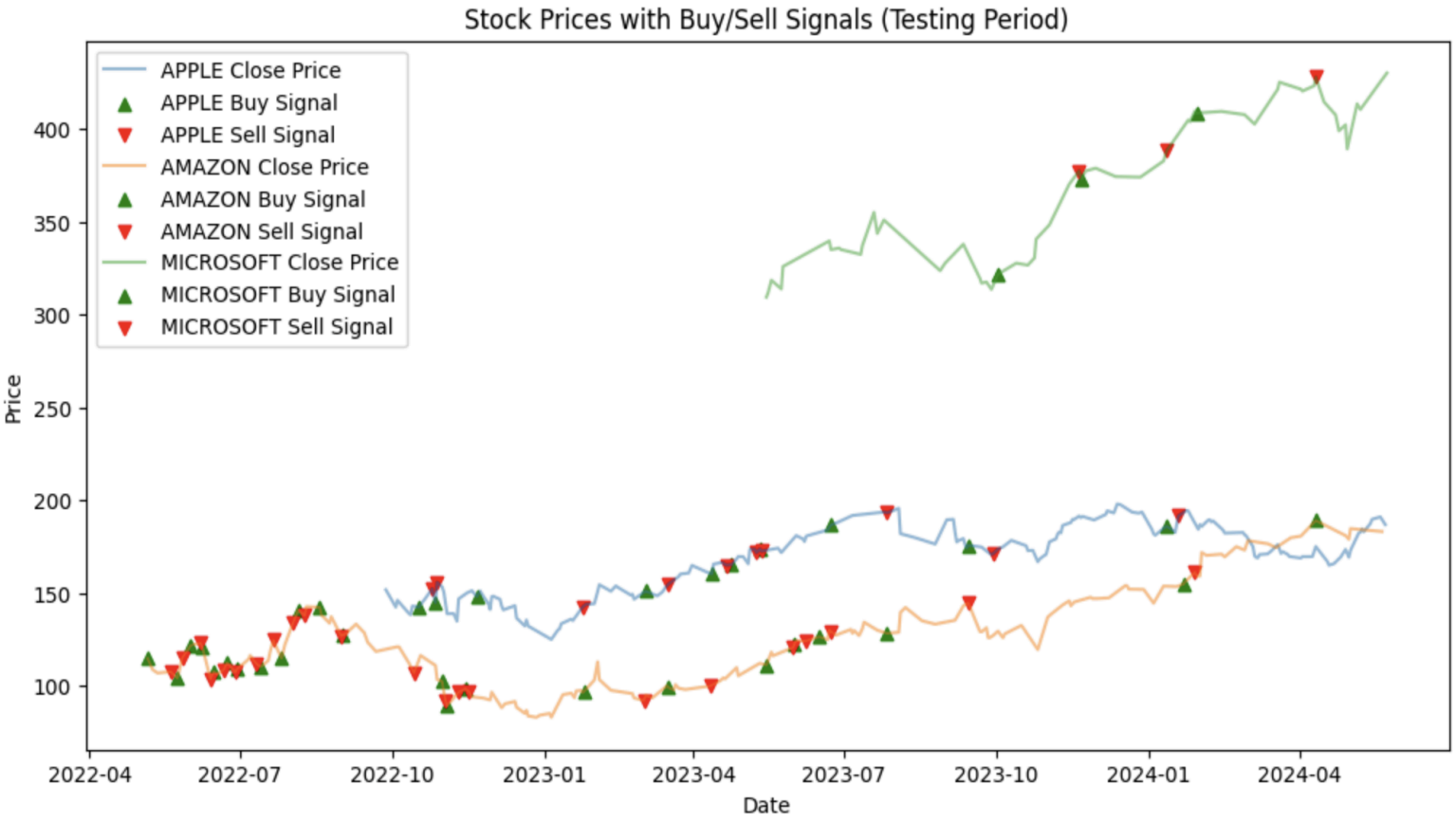
```
Final Capital: 161104.60824365623
Profit: 61104.60824365623
Profit Percentage: 61.10460824365623
Sharpe Ratio: 3.616205822062811 5
Maximum Drawdown: 31668.46379179432
Number of Trades Executed: 35
Win Ratio: 0.6571428571428571
AUC: 0.5443439581752255
Accuracy: 0.5583524027459954
Precision: 0.5619596541786743
Recall: 0.826271186440678
F1-score: 0.6689536878216124
```

- The portfolio's final capital is $161,104.61, starting from an initial capital of $100,000. This indicates a positive outcome from the trading strategy.
- The profit percentage of 61.104 percentage signifies a substantial return over the testing period.
- A Sharpe ratio of 3.62 is considered very good as it is even better than the sharpe ratio of S&P 500 which is 2.14 , indicating that the portfolio's returns are high relative to the risk taken.
- The maximum drawdown is $ 31668 which is a significant number but the final final of the portfolio indicates that the portfolio recovered well from this significant drop.
- The win ratio indicates that almost two-thirds of our executed trades were profitable and this is a good performance indicator.
- The AUC(Area under the curve) is equivalent to 0.54 which is slightly better than random guessing.This suggests that the model has a limited capability to distinguish between positive and negative movements.
- An accuracy of 55.83 percent means that the model could predict the movements of stock price correctly almost 56 percent of the times.Precision of 56 percent

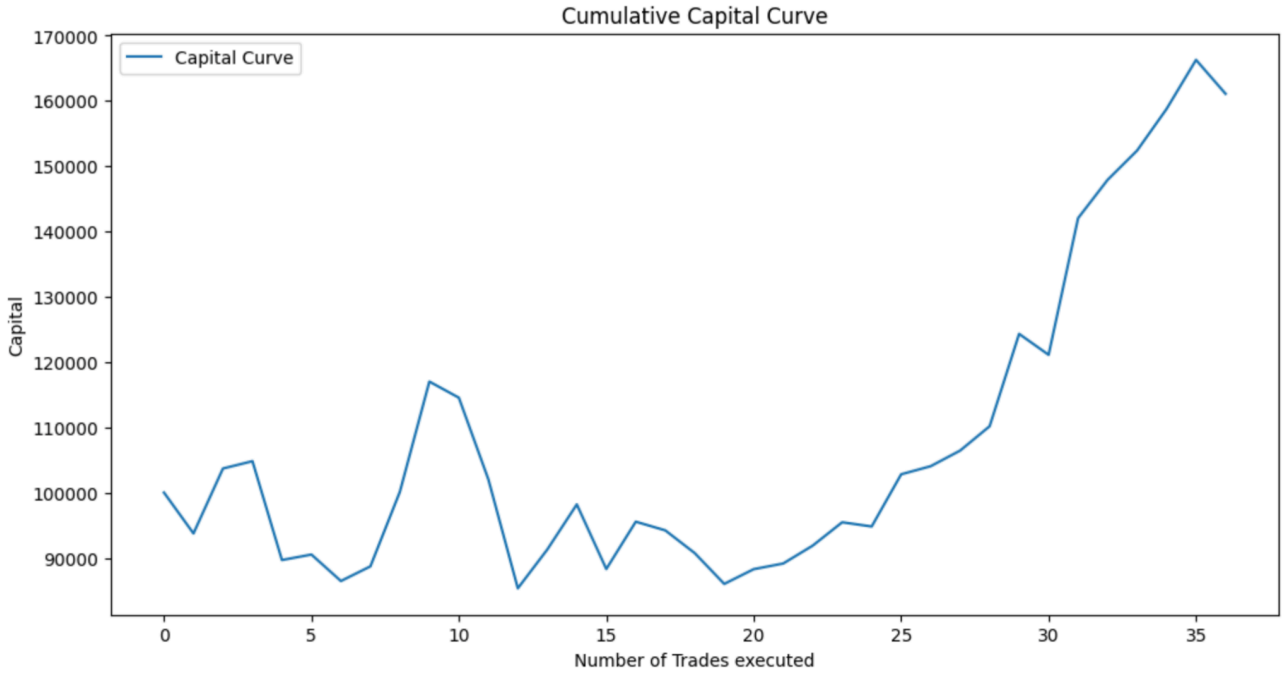
signifies that when the model predicted a positive movement, it was correct 56.20% of the times.

- Recall of 82.63% indicates that the model was able to identify 82.63% of the actual positive movements.
- The F1-score of 66.90% is a balance between precision and recall, showing that the model has a moderate overall classification performance.

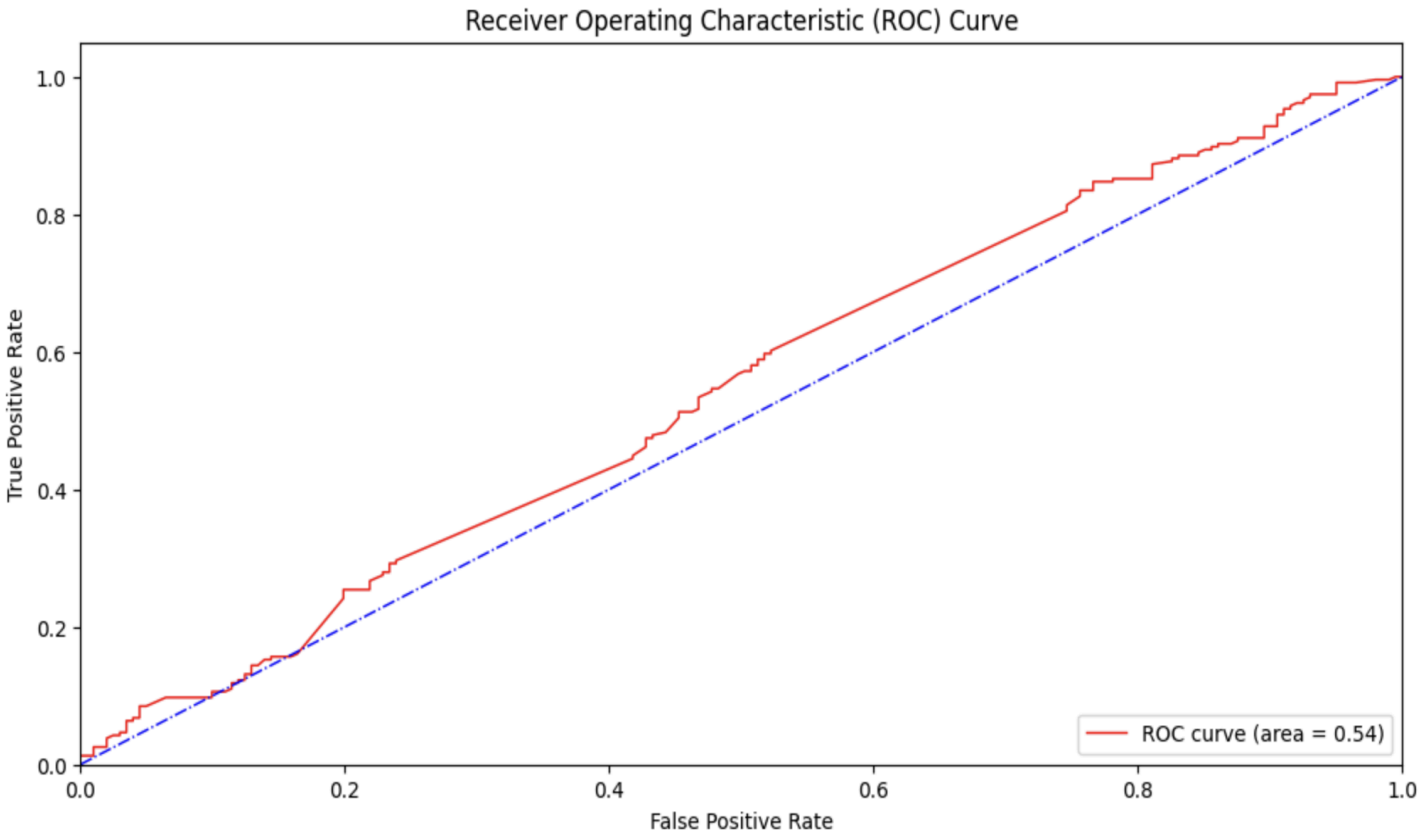
2. BUY AND SELL POINTS:



3. CUMULATIVE CAPITAL CURVE:


Cumulative Capital Curve
Capital Curve
Capital
Number of Trades executed
170000
160000
150000
140000
130000
120000
110000
100000
90000
0
5
10
15
20
25
30
35

4. ROC CURVE:


Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
ROC curve (area = 0.54)
0.0
0.2
0.4
0.6
0.8
1.0

CONCLUSION:

The trading strategy implemented in this project has shown promising results with a significant amount of profit over the testing period.The high sharpe ratio indicates that the strategy was able to deliver good return while managing and mitigating the risk effectively.

However, the high maximum drawdown might be concerning for many investors.The recall of the model is quite high which is a sign of good performance but the accuracy and precision metrics indicate that there is some scope of improvement in the strategy.The AUC is also slightly better than random guessing.

Overall , the strategy proved to be effective but it can be made better with further refinements to handle the drawdowns and the accuracy effectively to get satisfactory results.