Title:AIDI 1002 Final Project (Enhancing Stock Market Predictive Models Using Machine Learning Techniques)

Group Member Names : Raju Thapa, Sudarshan Bhandari

INTRODUCTION:

The purpose of this project is to explore and enhance the predictive power of financial models using machine learning techniques. The primary focus is on the S&P 500 stock data, employing Random Forest and Multi-Layer Perceptron (MLP) models to predict stock movements and optimize trading strategies.

AIM: To approach the single feature setting discussed in Ghosh, Pushpendu & Neufeld, Ariel & Sahoo, Jajati. (2020) and upgrade it to a new single-feature trading strategy by employing different models to make it more comparable to the multi-feature setting.

Github Repo: https://github.com/Raju979/FinalProjectMLP

Data: https://github.com/pushpendughosh/Stock-market-forecasting/blob/master/data/Open-1990.csv

---

DESCRIPTION OF PAPER:

Ghosh, Pushpendu & Neufeld, Ariel & Sahoo, Jajati. (2020) introduced multi-feature setting and compared it with single feature setting of Fischer & Krauss(2018). The multi-feature setting outperformed the single feature setting. This term project we tried to implement different model in given single feature setting to make comparison with aim of developing new single feature setting that outperforms the given original single feature setting and hence making it more compareable to multi-feature setting discussed by Ghosh, Pushpendu & Neufeld, Ariel & Sahoo, Jajati. (2020).

PROBLEM STATEMENT :

Update currently available original single feature setting machine learning model to develop a new and better single feature setting machine learning model.

CONTEXT OF THE PROBLEM:

- Stock market is volatile in nature and with development of Machine Learning many development have been made to understand stock market trends and predict prices.

- Use of various training methodology and settings have been used to train and analyze effectveness of prediction with available data.

SOLUTION:

- To upgrade existing model and to itroduce Multi-Layer Perceptron (MLP) algorithm to improve outcome. Compaire new models with original model to recognize a most promising approcah to compete with multi-feature setting.

Background

Our model utilizes historical stock data from the S&P 500 index, focusing on daily close prices and other financial indicators.

Reference Explanation:

The paper details a standard approach using Random Forest, which we have adapted and extended by adjusting parameters and incorporating an MLP for comparative analysis.

Dataset/Input:

The dataset comprises historical daily prices and financial indicators of the S&P 500 index stocks spanning several years.

Weakness:

The initial model does not account for non-linear interactions between features as effectively as potential neural network approaches.

Implement paper code :

```
# This is formatted as code

import pandas as pd
import numpy as np
import random
import time
import pickle
from sklearn.preprocessing import OneHotEncoder
from Statistics import Statistics
from sklearn.ensemble import RandomForestClassifier

import os
SEED = 9
os.environ['PYTHONHASHSEED']=str(SEED)
random.seed(SEED)
np.random.seed(SEED)

SP500_df = pd.read_csv('data/SPXconst.csv')
all_companies = list(set(SP500_df.values.flatten()))
all_companies.remove(np.nan)

constituents = {'-'.join(col.split('/')
[::-1]):set(SP500_df[col].dropna())
                for col in SP500_df.columns}

constituents_train = {}
for test_year in range(1993,2016):
    months = [str(t)+'-0'+str(m) if m<10 else str(t)+'-
'+str(m)
              for t in range(test_year-3,test_year) for m
in range(1,13)]
    constituents_train[test_year] = [list(constituents[m])
for m in months]
    constituents_train[test_year] = set([i for sublist in
constituents_train[test_year] for i in sublist])


def trainer(train_data,test_data):
    random.seed(SEED)
    np.random.seed(SEED)

    train_x,train_y = train_data[:,2:-2],train_data[:,-1]
    train_y = train_y.astype('int')

    print('Started training')
    clf = RandomForestClassifier(n_estimators=1000,
max_depth=20, random_state = SEED, n_jobs=-1)
    clf.fit(train_x,train_y)
    print('Completed ',clf.score(train_x,train_y))

    dates = list(set(test_data[:,0]))
    predictions = {}
    for day in dates:
        test_d = test_data[test_data[:,0]==day]
        test_d = test_d[:,2:-2]
        predictions[day] = clf.predict_proba(test_d)[:,1]
    return predictions
```

```
def simulate(test_data,predictions):
    rets = pd.DataFrame([],columns=['Long','Short'])
    k = 10
    for day in sorted(predictions.keys()):
        preds = predictions[day]
        test_returns = test_data[test_data[:,0]==day]
[:,-2]
        top_preds = predictions[day].argsort()[-k:][::-1]
        trans_long = test_returns[top_preds]
        worst_preds = predictions[day].argsort()[:k][::-1]
        trans_short = -test_returns[worst_preds]
        rets.loc[day] =
[np.mean(trans_long),np.mean(trans_short)]
    return rets

def create_label(df,perc=[0.5,0.5]):
    perc = [0.]+list(np.cumsum(perc))
    label = df.iloc[:,1:].pct_change(fill_method=None)
[1:].apply(
        lambda x:
pd.qcut(x.rank(method='first'),perc,labels=False), axis=1)
    return label

def create_stock_data(df,st):
    st_data = pd.DataFrame([])
    st_data['Date'] = list(df['Date'])
    st_data['Name'] = [st]*len(st_data)
    for k in list(range(1,21))+list(range(40,241,20)):
        st_data['R'+str(k)] = df[st].pct_change(k)
    st_data['R-future'] = df[st].pct_change().shift(-1)
    st_data['label'] = list(label[st])+[np.nan]
    st_data['Month'] = list(df['Date'].str[:-3])
    st_data = st_data.dropna()

    trade_year = st_data['Month'].str[:4]
    st_data = st_data.drop(columns=['Month'])
    st_train_data = st_data[trade_year<str(test_year)]
    st_test_data = st_data[trade_year==str(test_year)]
    return np.array(st_train_data),np.array(st_test_data)

result_folder = 'results-NextDay-240-1-RF'
for directory in [result_folder]:
    if not os.path.exists(directory):
        os.makedirs(directory)

for test_year in range(1993,1994):

    print('-'*40)
    print(test_year)
    print('-'*40)

    filename = 'data/Close-'+str(test_year-3)+'.csv'
    df = pd.read_csv(filename)
    label = create_label(df)
    stock_names = sorted(list(constituents[str(test_year-
1)+'-12']))
    train_data,test_data = [],[]

    start = time.time()
    for st in stock_names:
        st_train_data,st_test_data =
create_stock_data(df,st)
        train_data.append(st_train_data)
        test_data.append(st_test_data)

    train_data = np.concatenate([x for x in train_data])
    test_data = np.concatenate([x for x in test_data])

    print('Created
:',train_data.shape,test_data.shape,time.time()-start)


    predictions = trainer(train_data,test_data)
    returns = simulate(test_data,predictions)
    result = Statistics(returns.sum(axis=1))
    print('\nAverage returns prior to transaction
charges')
    result.shortreport()

    with open(result_folder+'/predictions-
'+str(test_year)+'.pickle', 'wb') as handle:
        pickle.dump(predictions, handle,
protocol=pickle.HIGHEST_PROTOCOL)

    returns.to_csv(result_folder+'/avg_daily_rets-
'+str(test_year)+'.csv')
    with open(result_folder+"/avg_returns.txt", "a") as
myfile:
        res = '-'*30 + '\n'
        res += str(test_year) + '\n'
        res += 'Mean = ' + str(result.mean()) + '\n'
        res += 'Sharpe = '+str(result.sharpe()) + '\n'
        res += '-'*30 + '\n'
        myfile.write(res)
```

Contribution Code :

```
# This is formatted as code

  #Upgraded to existing random forest model

  def upgraded_trainer1(train_data, test_data):
    # Adjusting parameters different than previous
    clf = RandomForestClassifier(
        n_estimators=500,  # Reduced number for quicker
execution; increase for potentially better performance.
        max_depth=25,      # Increasing depth allows the
model to learn more complex patterns.
        min_samples_split=10,  # Increasing to avoid
overfitting on the training data.
        max_features='sqrt',   # Using the square root of
features, typical for classification.
        random_state=SEED,
        n_jobs=-1
    )
    train_x, train_y = train_data[:, 2:-2], train_data[:,
-1].astype('int')
    clf.fit(train_x, train_y)
    print('Training complete with score:',
clf.score(train_x, train_y))

    # Prediction process remains unchanged
    dates = list(set(test_data[:, 0]))
    predictions = {}
    for day in dates:
        test_d = test_data[test_data[:, 0] == day]
        test_d = test_d[:, 2:-2]
        predictions[day] = clf.predict_proba(test_d)[:, 1]
    return predictions


    def trainer2(model, test_data):
    random.seed(SEED)
    np.random.seed(SEED)

    train_x,train_y = train_data[:,2:-2],train_data[:,-1]
    train_y = train_y.astype('int')

    print('Started training')
    clf_mlp = MLPClassifier(hidden_layer_sizes=(100,),
activation='relu', solver='adam', max_iter=300,
random_state=SEED)
    clf_mlp.fit(train_x,train_y)
    print('Completed ',clf_mlp.score(train_x,train_y))
    dates = list(set(test_data[:,0]))
    predictions = {}
    for day in dates:
        test_d = test_data[test_data[:,0]==day]
        test_d = test_d[:,2:-2]
        predictions[day] = clf_mlp.predict_proba(test_d)
[:,1]
    return predictions

  #added to MLP model

    def trainer(model, test_data):
    random.seed(SEED)
```

```
        np.random.seed(SEED)

        train_x,train_y = train_data[:,2:-2],train_data[:,-1]
        train_y = train_y.astype('int')

        print('Started training')
        clf_mlp = MLPClassifier(hidden_layer_sizes=(100,),
    activation='relu', solver='adam', max_iter=300,
    random_state=SEED)
        clf_mlp.fit(train_x,train_y)
        print('Completed ',clf_mlp.score(train_x,train_y))
        dates = list(set(test_data[:,0]))
        predictions = {}
        for day in dates:
            test_d = test_data[test_data[:,0]==day]
            test_d = test_d[:,2:-2]
            predictions[day] = clf_mlp.predict_proba(test_d)
    [:,1]
        return predictions
```

Results :

---

avg_returns from Random Forest:

Mean = 0.0062882669280149095

Sharpe = 6.687684244449364

---

avg_returns from Upgraded Random Forest:

Mean = 0.005862748362348335

Sharpe = 6.704790091448804

---

avg_returns from Multiple Linear Perceptron:

Mean = 0.003157689272982879

Sharpe = 3.4967283062888215

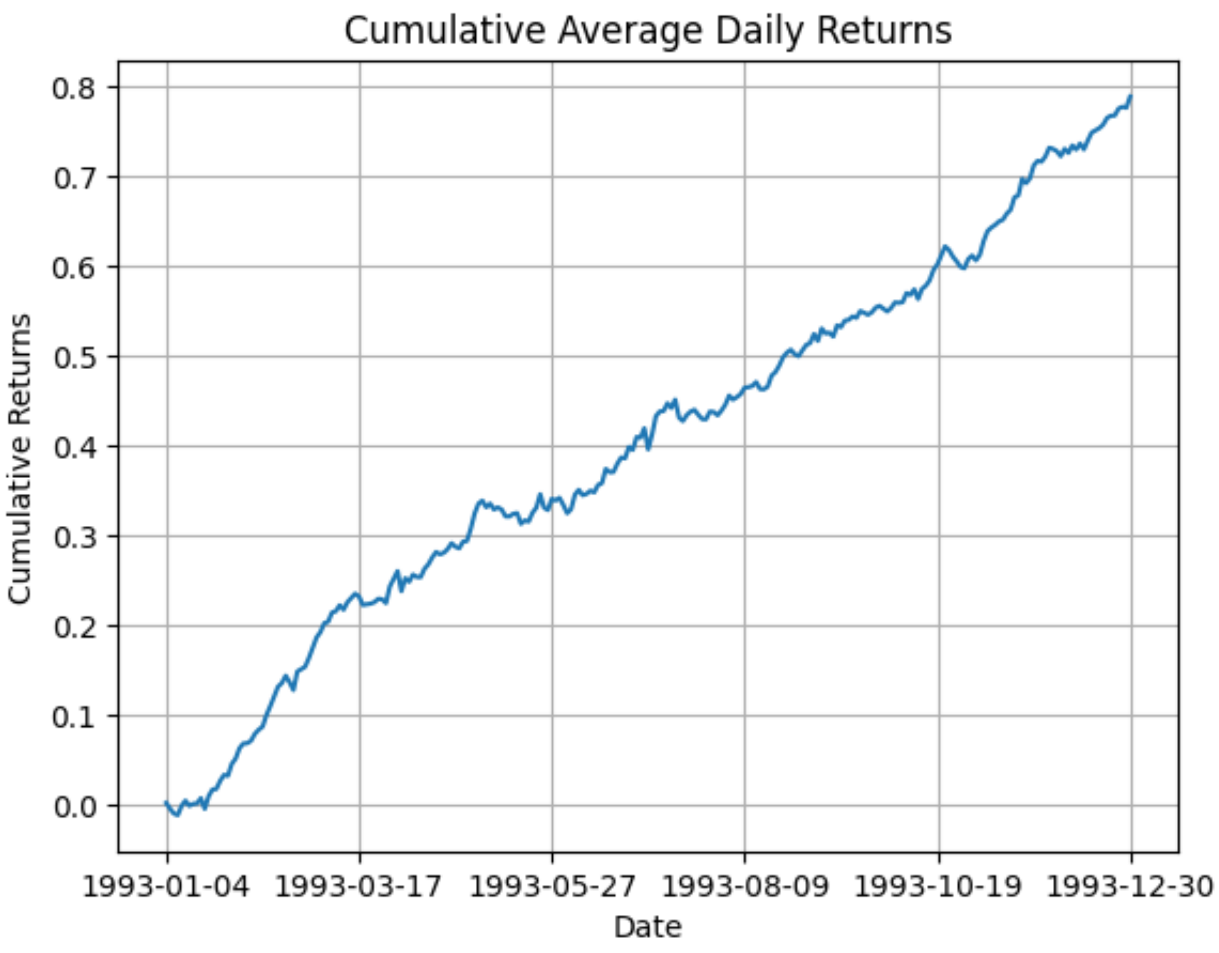
Observations :

The comparative analysis of the average returns and Sharpe ratios from the Random Forest, Upgraded Random Forest, and Multiple Linear Perceptron (MLP) models reveals distinct performance characteristics and suitability for stock market prediction. The original Random Forest model displayed the highest mean return at approximately 0.006288, coupled with an impressive Sharpe ratio of 6.687, indicating strong performance with substantial returns adjusted for volatility. This highlights the model's capability in managing risk while generating favorable returns.

The Upgraded Random Forest, despite recording a slight decrease in mean returns to 0.005863, demonstrated a slightly improved Sharpe ratio of 6.705. This marginal enhancement in the Sharpe ratio suggests that the adjustments made to the model's parameters effectively reduced volatility, thereby offering a more consistent risk-adjusted performance, albeit at a slight cost to raw returns.

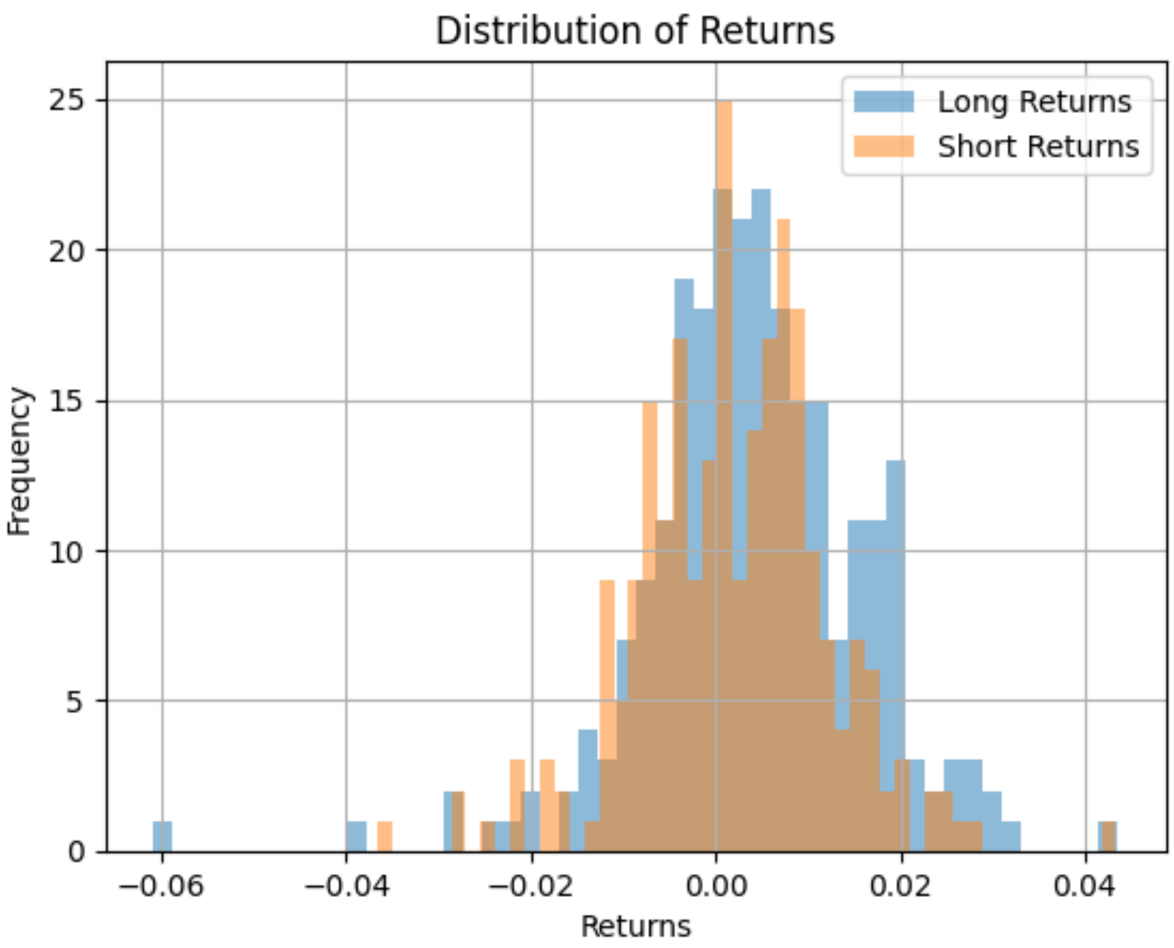
In contrast, the MLP model, while leveraging its strength in capturing complex non-linear relationships, yielded a lower mean return of 0.003158 and a significantly reduced Sharpe ratio of 3.497. This performance indicates a suboptimal balance between return and risk, suggesting that the MLP might be less efficient in this particular application or requires further optimization to enhance its effectiveness in financial forecasting.

In [ ]:
```
#Cumulative Average Daily Returns- Random Forest
returns.mean(axis=1).cumsum().plot(title='Cumulative Average Daily Return
plt.xlabel('Date')
plt.ylabel('Cumulative Returns')
plt.grid(True)
plt.show()
```

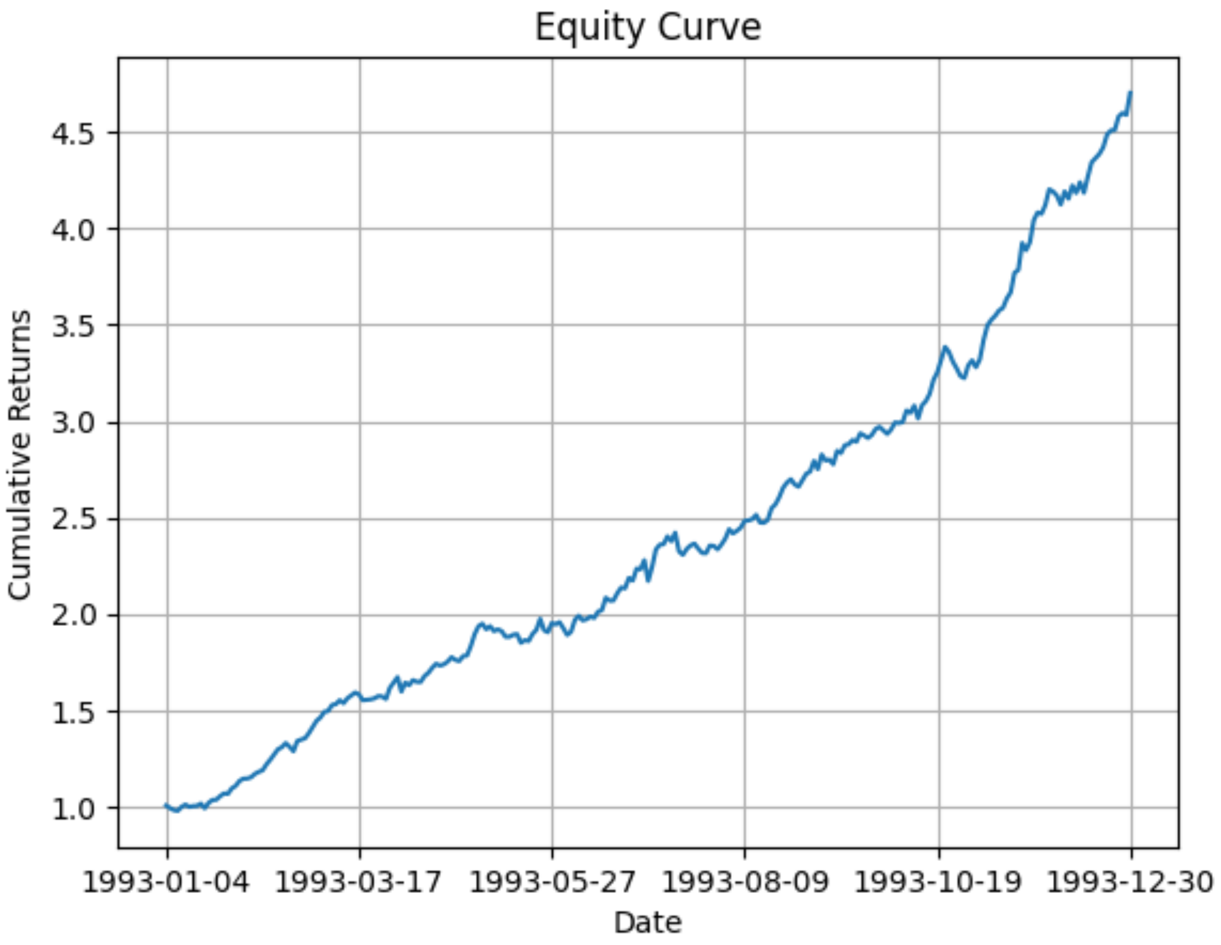


In [ ]:
```
# Histogram of long returns for Random Forest
```

```
returns['Long'].hist(bins=50, alpha=0.5, label='Long Returns')
# Histogram of short returns
returns['Short'].hist(bins=50, alpha=0.5, label='Short Returns')
plt.title('Distribution of Returns')
plt.xlabel('Returns')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

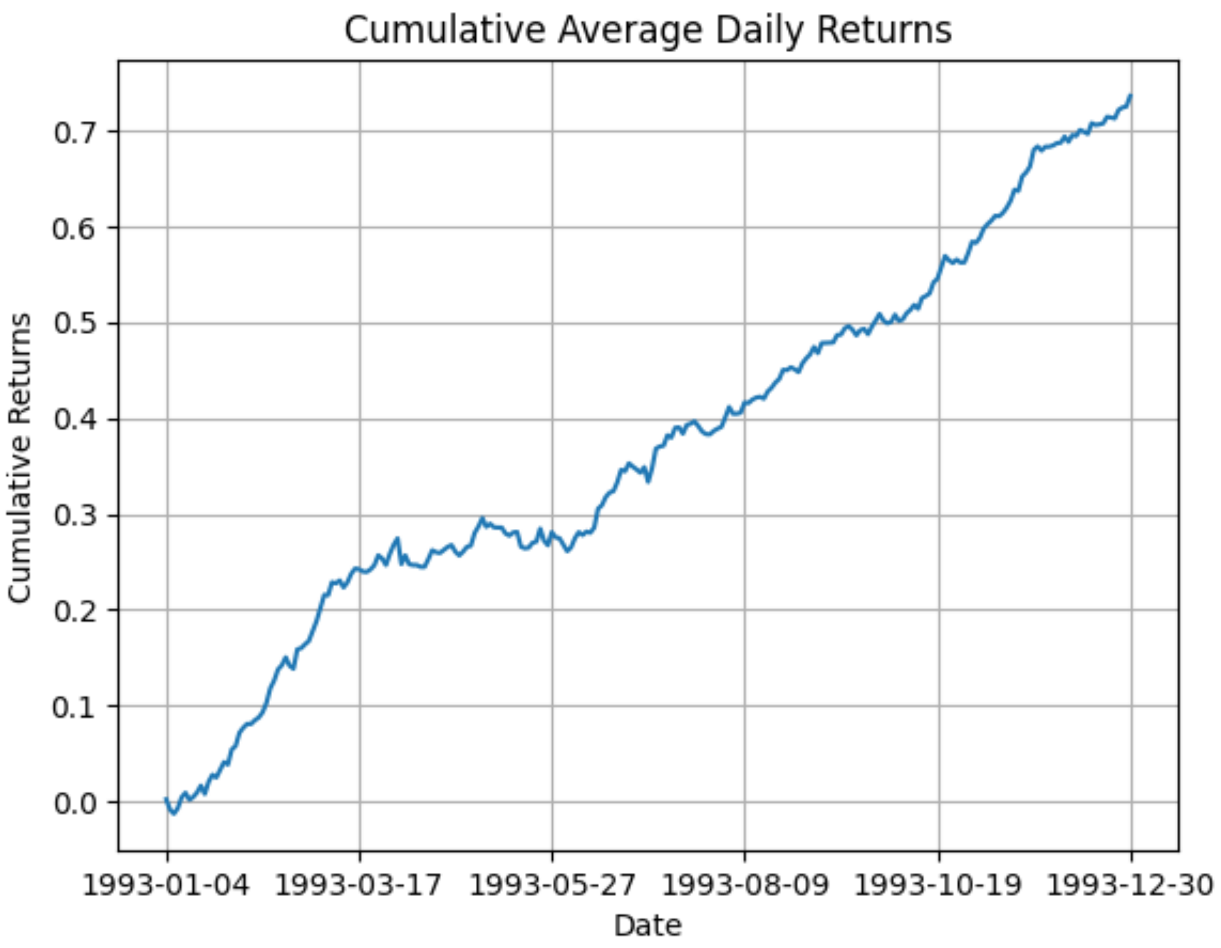


```
In [ ]: # Cumulative returns as an equity curve- Random Forest
(1 + returns.sum(axis=1)).cumprod().plot(title='Equity Curve')
plt.xlabel('Date')
plt.ylabel('Cumulative Returns')
plt.grid(True)
plt.show()
```

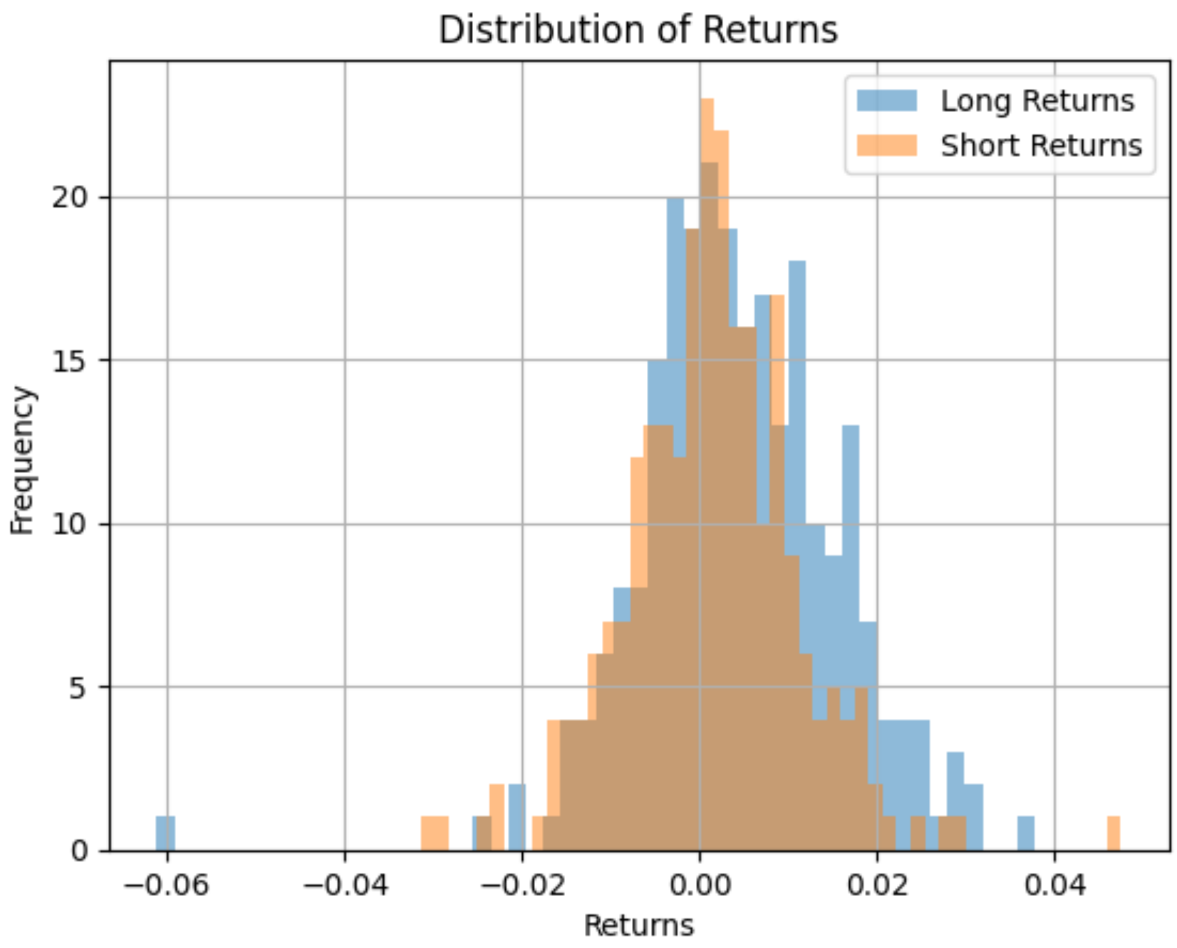


```
In [ ]:
```

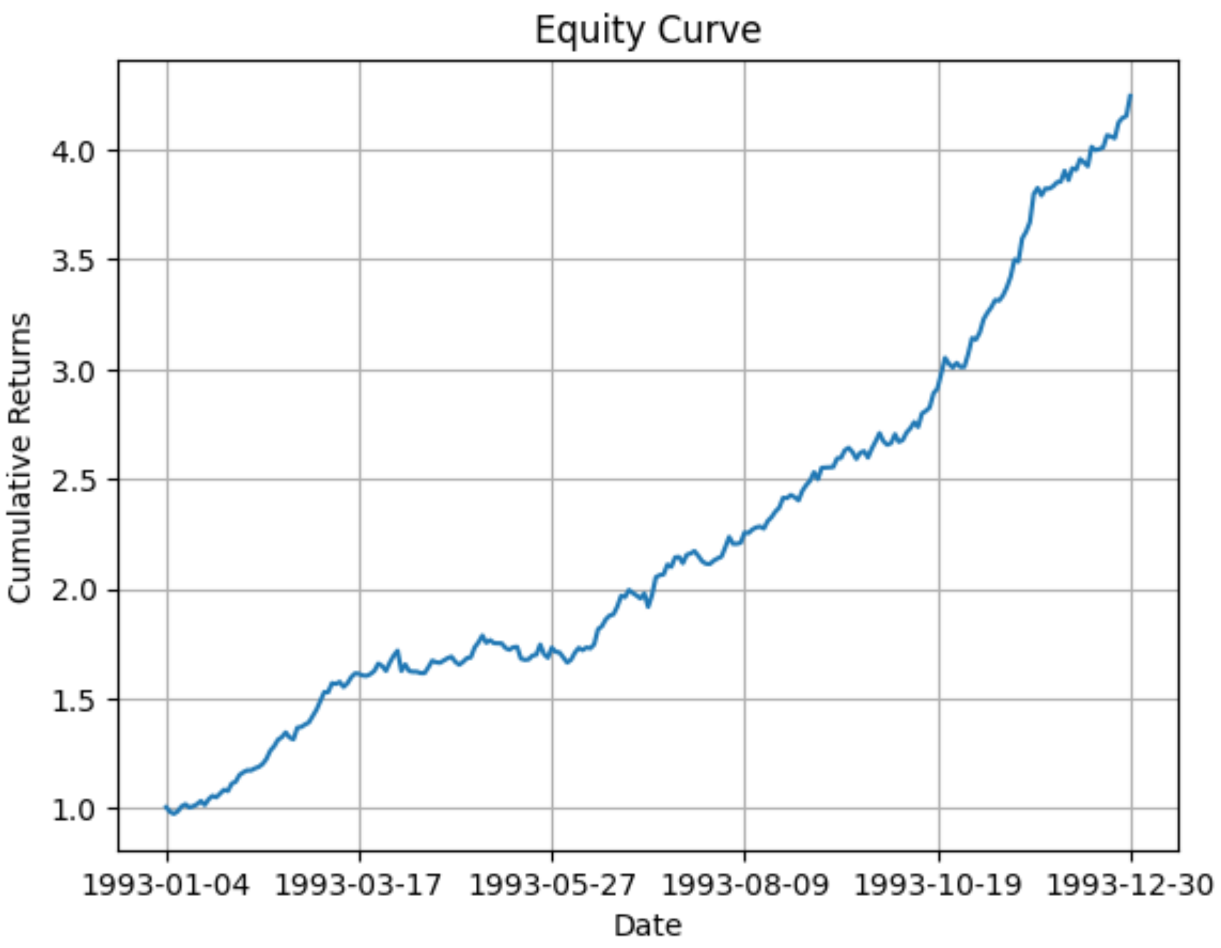
```
In [ ]: #Cumulative Average Daily Returns- Upgraded Random Forest
returns.mean(axis=1).cumsum().plot(title='Cumulative Average Daily Return
plt.xlabel('Date')
plt.ylabel('Cumulative Returns')
plt.grid(True)
plt.show()
```



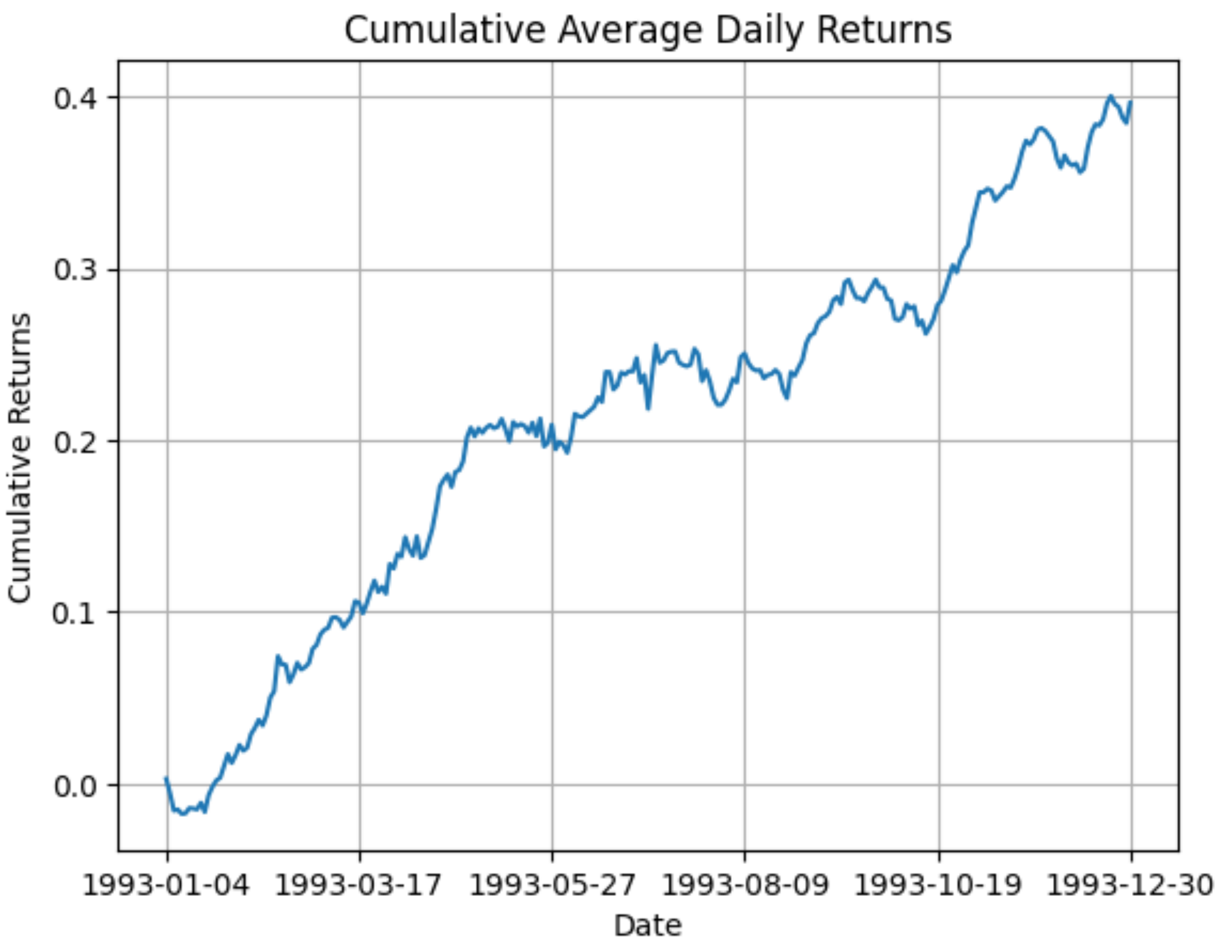
```
In [ ]: # Histogram of long returns– Upgraded Random Forest
        returns['Long'].hist(bins=50, alpha=0.5, label='Long Returns')
        # Histogram of short returns
        returns['Short'].hist(bins=50, alpha=0.5, label='Short Returns')
        plt.title('Distribution of Returns')
        plt.xlabel('Returns')
        plt.ylabel('Frequency')
        plt.legend()
        plt.show()
```



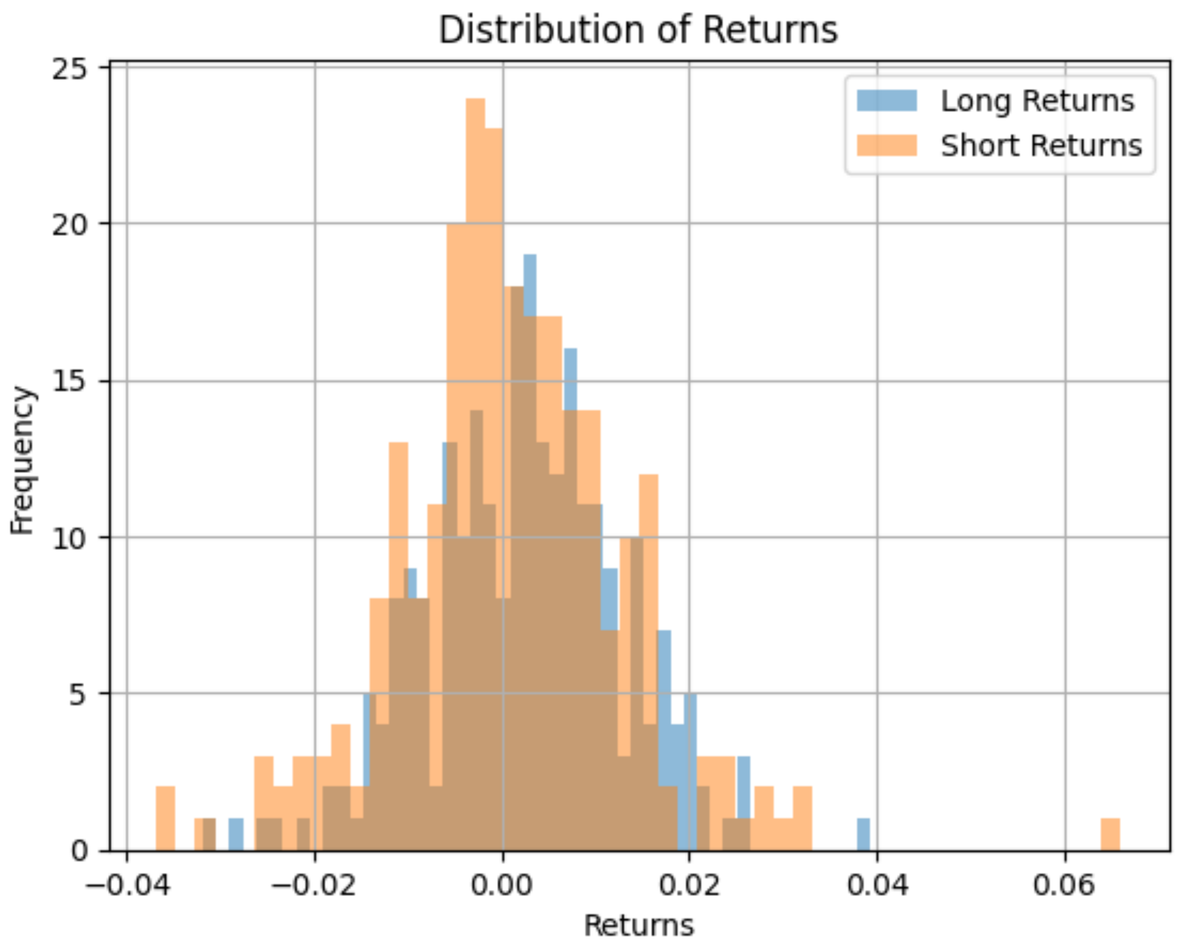
```
In [ ]: # Cumulative returns as an equity curve- Upgraded Random Forest
        (1 + returns.sum(axis=1)).cumprod().plot(title='Equity Curve')
        plt.xlabel('Date')
        plt.ylabel('Cumulative Returns')
        plt.grid(True)
        plt.show()
```



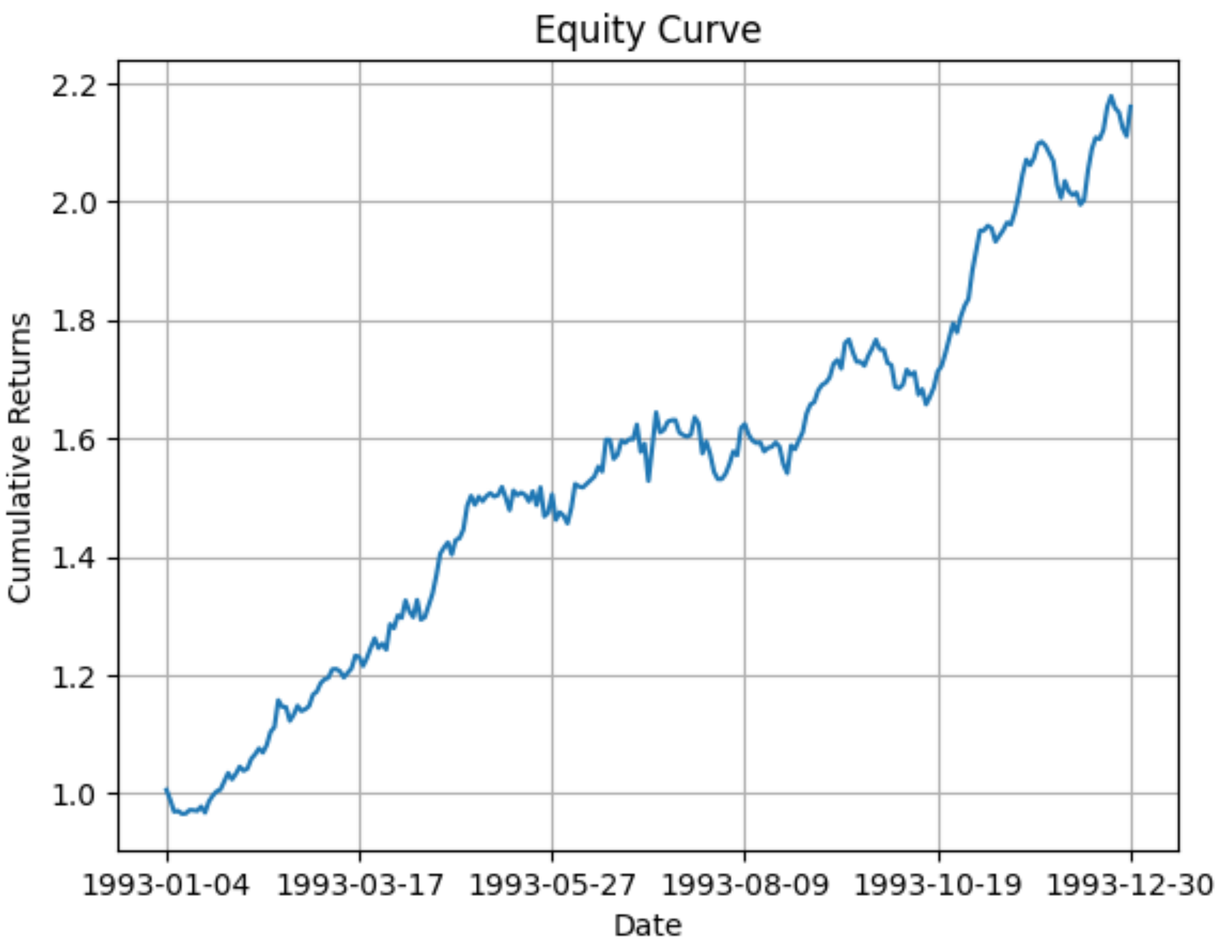
In [ ]:
```
#Cumulative Average Daily Returns- MLP
returns.mean(axis=1).cumsum().plot(title='Cumulative Average Daily Return
plt.xlabel('Date')
plt.ylabel('Cumulative Returns')
plt.grid(True)
plt.show()
```



```
In [ ]: # Histogram of long returns- MLP
        returns['Long'].hist(bins=50, alpha=0.5, label='Long Returns')
        # Histogram of short returns
        returns['Short'].hist(bins=50, alpha=0.5, label='Short Returns')
        plt.title('Distribution of Returns')
        plt.xlabel('Returns')
        plt.ylabel('Frequency')
        plt.legend()
        plt.show()
```



In [ ]:
```
# Cumulative returns as an equity curve- MLP
(1 + returns.sum(axis=1)).cumprod().plot(title='Equity Curve')
plt.xlabel('Date')
plt.ylabel('Cumulative Returns')
plt.grid(True)
plt.show()
```



Conclusion and Future Direction :

Learnings : The upgraded random forest model achieved a higher average return and sharp ratio compaired to MLP model and original Random forest model.MLP algorithm did not perform as expected by writters.

Results Discussion : The Random Forest model continues to be promissing approach for future prediction of price movements. The Random Forest model continued to demonstrate robustness ans stability in its performance. The MLP model while having potential showed more sensitivity to fluctuatng market.

Limitations : Limited data was available for training and testing, only single year data was publicly avaialble to use. The analysis was also limited to few performance and stastistical measures. Exploration of other machine learning model was limited by timeframe and scope of this project.

Future Extension : Further tuning hyperparameter of Random Forest will potentially improve performance and make it more compaireable to multi-feature setting.Also,conducting more comprehensive analysis of results is recommended. Exploration of other machine learning model to see if they can achieve better result than current model is highly recommended.

References:

[1]: Ghosh, Pushpendu & Neufeld, Ariel & Sahoo, Jajati. (2020). Forecasting directional movements of stock prices for intraday trading using LSTM and random forests.
https://www.researchgate.net/publication/340826434_Forecasting_directional_moveme

[2]: Fischer, Thomas & Krauss, Christopher. (2017). Deep learning with long short-term memory networks for financial market predictions. European Journal of Operational Research. 270. 10.1016/j.ejor.2017.11.054.