Stock Market Prediction using Machine Learning

Name: Bangaru Mohnish

Reg.No: RA1711003011397

School: SRM IST

Abstract

If a Human Investor Can be Successful Why can't a machine!

Predicting how the stock market will perform is one of the most difficult things to do. There are so many factors involved in the prediction – physical factors vs. physiological, rational and irrational behavior, etc. All these aspects combine to make share prices volatile and very difficult to predict with a high degree of accuracy.

Stock market analysis is divided into two parts – Fundamental Analysis and Technical Analysis.

- Fundamental Analysis involves analyzing the company's future profitability on the basis of its current business environment and financial performance.
- Technical Analysis, on the other hand, includes reading the charts and using statistical figures to identify the trends in the stock market.

Libraries Used:

1.Tensorflow

2.Numpy

3.Pandas

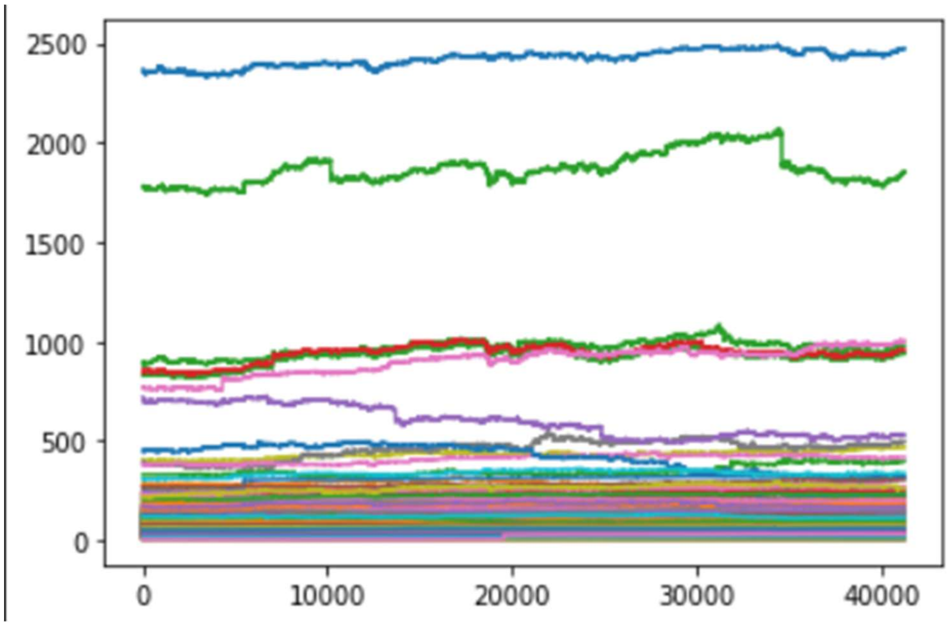
4.MinMaxScaler form sklearn

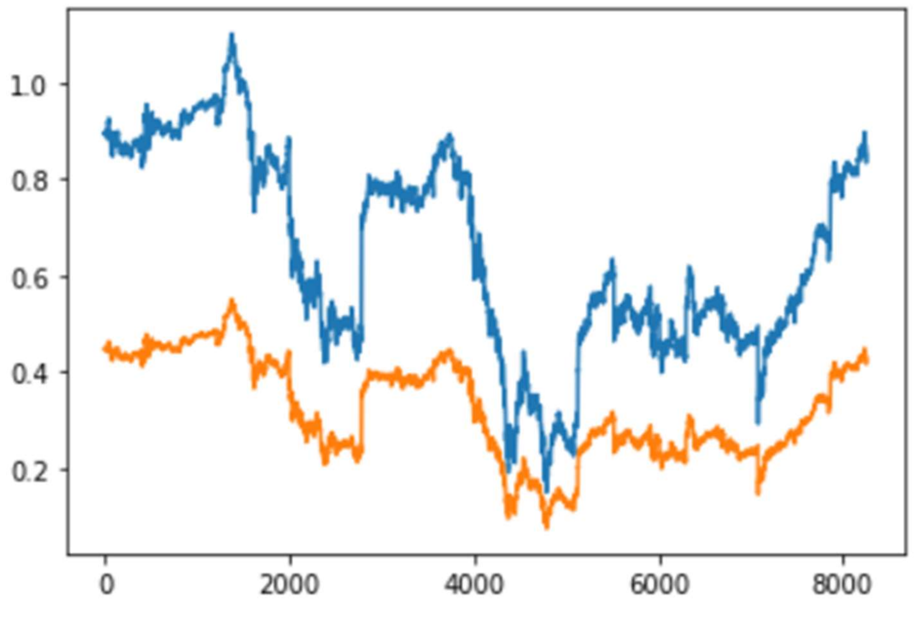
5.Pyplot form Matplotlib

Data Used:

1.S&P 500 time series data

Output Graphs:





Conclusion:

This Project has helped me gain a better insight of the stock market functioning and an approach to solve the problems faced by investors using my technical skills.

Name: Bangaru Mohnish

Reg.No: RA1711003011397

School: SRM Institute of Science and Technology

Appendix:

```python
import tensorflow as tf
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt

data = pd.read_csv('D:\Downloads\sp500\data_stocks.csv')
data = data.drop(['DATE'], 1)

n = data.shape[0]
p = data.shape[1]

data = data.values

plt.plot(data)

train_start = 0
train_end = int(np.floor(0.8*n))
test_start = train_end + 1
test_end = n
data_train = data[np.arange(train_start, train_end), :]
data_test = data[np.arange(test_start, test_end), :]

scaler = MinMaxScaler(feature_range=(-1, 1))
scaler.fit(data_train)
data_train = scaler.transform(data_train)
data_test = scaler.transform(data_test)

X_train = data_train[:, 1:]
y_train = data_train[:, 0]
X_test = data_test[:, 1:]
y_test = data_test[:, 0]

import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

```
a = tf.placeholder(dtype = tf.int8)
b = tf.placeholder(dtype = tf.int8)

c = tf.add(a, b)

graph = tf.Session()

graph.run(c, feed_dict = {a: 5, b:4})

n_stocks = X_train.shape[1]

n_neurons_1 = 1024
n_neurons_2 = 512
n_neurons_3 = 256
n_neurons_4 = 128

X = tf.placeholder(dtype=tf.float32, shape=[None, n_stocks])
Y = tf.placeholder(dtype=tf.float32, shape=[None])

weight_initializer = tf.variance_scaling_initializer(mode="fan_avg", distribution
="uniform", scale=sigma)
bias_initializer = tf.zeros_initializer()

W_hidden_1 = tf.Variable(weight_initializer([n_stocks, n_neurons_1]))
bias_hidden_1 = tf.Variable(bias_initializer([n_neurons_1]))
W_hidden_2 = tf.Variable(weight_initializer([n_neurons_1, n_neurons_2]))
bias_hidden_2 = tf.Variable(bias_initializer([n_neurons_2]))
W_hidden_3 = tf.Variable(weight_initializer([n_neurons_2, n_neurons_3]))
bias_hidden_3 = tf.Variable(bias_initializer([n_neurons_3]))
W_hidden_4 = tf.Variable(weight_initializer([n_neurons_3, n_neurons_4]))
bias_hidden_4 = tf.Variable(bias_initializer([n_neurons_4]))

W_out = tf.Variable(weight_initializer([n_neurons_4, 1]))
bias_out = tf.Variable(bias_initializer([1]))

hidden_1 = tf.nn.relu(tf.add(tf.matmul(X, W_hidden_1), bias_hidden_1))
hidden_2 = tf.nn.relu(tf.add(tf.matmul(hidden_1, W_hidden_2), bias_hidden_2))
hidden_3 = tf.nn.relu(tf.add(tf.matmul(hidden_2, W_hidden_3), bias_hidden_3))
hidden_4 = tf.nn.relu(tf.add(tf.matmul(hidden_3, W_hidden_4), bias_hidden_4))

out = tf.transpose(tf.add(tf.matmul(hidden_4, W_out), bias_out))

mse = tf.reduce_mean(tf.squared_difference(out, Y))

opt = tf.train.AdamOptimizer().minimize(mse)
```

```python
session = tf.InteractiveSession()
session.run(tf.global_variables_initializer())

plt.ion()
fig = plt.figure()
ax1 = fig.add_subplot(111)
line1, = ax1.plot(y_test)
line2, = ax1.plot(y_test * 0.5)
plt.show()

batch_size = 256
mse_train = []
mse_test = []

epochs = 10
for e in range(epochs):

    # Shuffle training data
    shuffle_indices = np.random.permutation(np.arange(len(y_train)))
    X_train = X_train[shuffle_indices]
    y_train = y_train[shuffle_indices]

for i in range(0, len(y_train) // batch_size):
    start = i * batch_size
    batch_x = X_train[start:start + batch_size]
    batch_y = y_train[start:start + batch_size]
    # Run optimizer with batch
    session.run(opt, feed_dict={X: batch_x, Y: batch_y})

if np.mod(i, 50) == 0:
    # MSE train and test
    mse_train.append(net.run(mse, feed_dict={X: X_train, Y: y_train}))
    mse_test.append(net.run(mse, feed_dict={X: X_test, Y: y_test}))
    print('MSE Train: ', mse_train[-1])
    print('MSE Test: ', mse_test[-1])
    # Prediction
    pred = net.run(out, feed_dict={X: X_test})
    line2.set_ydata(pred)
    plt.title('Epoch ' + str(e) + ', Batch ' + str(i))
    plt.pause(0.01)

mse_final = session.run(mse, feed_dict={X: X_test, Y: y_test})
print(mse_final)
```