IE4012
Machine Learning for Cyber Security
4rd Year, 1st Semester

Assignment

Stock Market Prediction

Submitted to

Sri Lanka Institute of Information Technology

In partial fulfillment of the requirements for the

Bachelor of Science Special Honors Degree in Information Technology

08/05/2020

Declaration

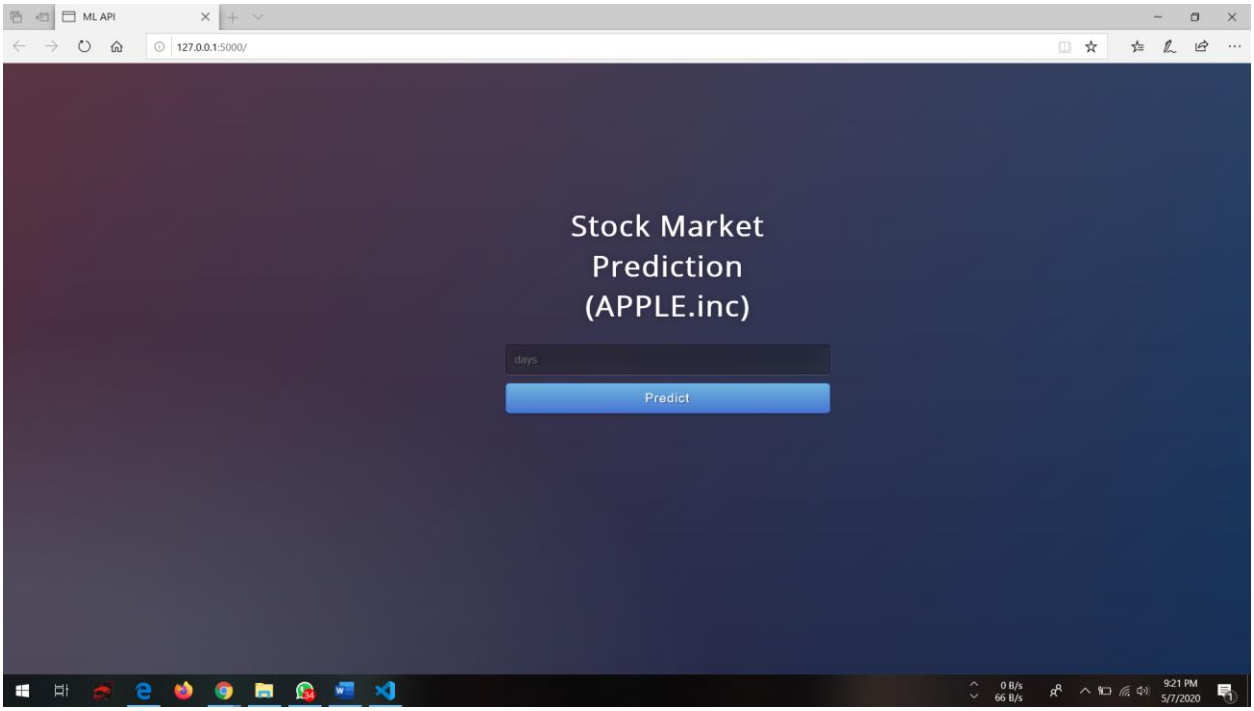
I certify that this report does not incorporate without acknowledgement, any material previously submitted for a degree or diploma in any university, and to the best of my knowledge and belief it does not contain any material previously published or written by another person, except where due reference is made in text.

Registration Number : IT17157438

Name : M Musharif

Apple Stock Market Prediction

The challenge is to predict apple stock market closing value so that user can a clear idea of market trends and invest money. In this model it will forecast closing values for given number of days. One of the toughest things to do is forecasting how the stock market will do. The prediction includes so many variables-physical factors vs. behavioral, moral, and irrational behavior, etc. All these things combine to make share prices unpredictable, and with a high degree of precision very difficult to forecast.



Reference - https://www.youtube.com/watch?v=EYnC4ACIt2g

https://medium.com/@randerson112358/predict-stock-prices-using-python-machine-learning-53aa024da20a

Machine learning algorithm

1. Linear Regression

Linear regression is a common Statistical Data Analysis technique. It is used to determine the extent to which there is a linear relationship between a dependent variable and one or more independent variables. Linear regression is suitable for predicting output that is continuous value, such as predicting the price of a property. The regression line is a straight line. Whereas logistic regression is for classification problems, which predicts a probability range between 0 to 1.

Linear Regression Pros:

- Simple to implement.
- Used to predict numeric values.

Linear Regression Cons:

- Prone to overfitting.
- Cannot be used when the relation between independent and dependent variable are nonlinear.

2. Support Vector Machine (SVM)

Support-vector machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems.

Support Vector Machine Pros:

- It is effective in high dimensional spaces.
- It works well with clear margin of separation.
- It is effective in cases where number of dimensions is greater than the number of samples.

Support Vector Machine Cons:

- It does not perform well, when we have large data set.
- Low performance if the data set is noisy (a large amount of additional meaningless information).

In this model I have used both SVM and Linear Regression and compare the results with each other.

Explanation of Code

In this model it will forecast closing values for given number of days. First, I will import the dependencies, that will make this program a little easier to write. I'm importing the machine learning following libraries.

```
1   import math
2   import pandas_datareader as web
3   import numpy as np
4   import matplotlib.pyplot as plt
5   from sklearn.linear_model import LinearRegression
6   from sklearn.model_selection import train_test_split
7   from sklearn.externals import joblib
8   from datetime import datetime
9   from flask import Flask, request, jsonify, render_template
10  plt.style.use('fivethirtyeight')
11
```

Next I get the stock data from pandas_datareader, and look at the collection of data. Here I get Apple stock data aka APPL, store it in a variable called 'df' short for data frame, and print the first five rows of data.

```
df = web.DataReader('AAPL', data_source='yahoo',start='1980-12-12', end=datetime.today().strftime('%Y-%m-%d'))
#get the close price
df = df[['Close']]
#print(df)
```

```
                 High         Low  ...       Volume   Adj Close
Date                               ...
1980-12-12   0.515625    0.513393  ...  117258400.0    0.406782
1980-12-15   0.488839    0.486607  ...   43971200.0    0.385558
1980-12-16   0.453125    0.450893  ...   26432000.0    0.357260
1980-12-17   0.464286    0.462054  ...   21610400.0    0.366103
1980-12-18   0.477679    0.475446  ...   18362400.0    0.376715
...               ...         ...  ...          ...         ...
2020-05-01  299.000000  285.850006  ...   60154200.0  289.070007
2020-05-04  293.690002  286.320007  ...   33392000.0  293.160004
2020-05-05  301.000000  294.459991  ...   36937800.0  297.559998
2020-05-06  303.239990  298.869995  ...   35512400.0  300.630005
2020-05-07  305.170013  301.970001  ...   18360486.0  303.809998

[9934 rows x 6 columns]
```

Using matplotlib we can visualize the closing values of apple stock market since 1980

```
#visualize the closing price
plt.figure(figsize=(12,4))
plt.title('Close Price History')
plt.plot(df['Close'])
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD $', fontsize=18)
plt.show()
```

I only need the Close (Close) price, so I am getting data only from the column 'Close' and storing this back into the variable 'df'. Then printing the first 5 rows of the new data set

```
#get the close price
df = df[['Close']]
print(df.head())
```

```
            Close
Date
1980-12-12  0.513393
1980-12-15  0.486607
1980-12-16  0.450893
1980-12-17  0.462054
1980-12-18  0.475446
```

Now, I'm creating a variable called forecast_out, to store the number of days into the future that I want to predict. This variable can change by the user. So, Default value of forecast_out is 30.

I also need a column (the target or dependent variable) that will hold the predicted price values 30 days into the future. The future price that I want that's 30 days into the future is just 30 rows down from the current Close price. So, I will create a new column called 'Prediction' and populate it with data from the Adj. Close column but shifted 30 rows up to get the price of the next 30 days, and then print the last 5 rows of the new data set.

```
# A variable for predicting 'n' days out into the future
forecast_out = 1
#create another column (the target dependent varible) shifted 'n' units up
df['Prediction'] = df[['Close']].shift(-forecast_out)
print(df)
```

```
                Close  Prediction
Date
1980-12-12   0.513393    0.486607
1980-12-15   0.486607    0.450893
1980-12-16   0.450893    0.462054
1980-12-17   0.462054    0.475446
1980-12-18   0.475446    0.504464
...               ...         ...
2020-05-01  289.070007  293.160004
2020-05-04  293.160004  297.559998
2020-05-05  297.559998  300.630005
2020-05-06  300.630005  304.000000
2020-05-07  304.000000         NaN

[9934 rows x 2 columns]
```

Next, I want to create the independent data set (X). This is the data set that I will use to train the machine learning model(s). To do this I will create a variable called ‘X’ , and convert the data into a NumPy (np) array after dropping the ‘Prediction’ column, then store this new data into ‘X’.

```
### Create the independant dataset (x)
# convert the datafreame to a numpy array
X = np.array(df.drop(['Prediction'],1))
#remove the last 'n' rows
X = X[:-forecast_out]
print(X)
```

```
[[  0.51339287]
 [  0.48660713]
 [  0.45089287]
 ...
 [293.16000366]
 [297.55999756]
 [300.63000488]]
```

I created the independent data set in the previous step, now I will create the dependent data set called ‘y’. This is the target data, the one that holds the future price predictions.

To create this new data set ‘y’, I will convert the data frame into a NumPy array and from the ‘Prediction’ column, store it into a new variable called ‘y’ and then remove the last 30 rows of data from ‘y’. Then I will print ‘y’ to make sure there are no NaN’s.

```
### Create the dependent dataset (y)
# convert the dataframe to a numpy array(All the values including NaN's)
Y = np.array(df['Prediction'])
# get all of the y values except the last 'n' rows
Y = Y[:-forecast_out]
print(Y)
```

```
[  0.48660713   0.45089287   0.46205357 ... 297.55999756 300.63000488
 303.80010986]
```

Now that I have my new cleaned and processed data sets ‘X’ & ‘y’. I can split them up into 80% training and 20 % testing data for the model.

```
# split the data
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
```

I can start creating and training the models. First, I will create and train the Support Vector Machine.

```
# create and train the svm
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)
svr_rbf.fit(x_train, y_train)
```

Let’s test the model by getting the score also known as the coefficient of determination R^2 of the prediction. The best possible score is 1.0, and the model returns a score of 0.9994120054063238.

```
## testing model: Score returns the coefficient of determination R^2 of the prediction
## best score is 1.0
svm_confidence = svr_rbf.score(x_test, y_test)
print("svm confidence: ", svm_confidence)
```

```
svm confidence:  0.9994120054063238
```

Next, I will create & train the Linear Regression model

```
# Create and train Linear regression model
lr = LinearRegression()
# train the model
lr.fit(x_train, y_train)
```

Let’s test the model by getting the score also known as the coefficient of determination R^2 of the prediction. The best possible score is 1.0, and the model returns a score of 0.9994618614480701.

```
## testing model: Score returns the coefficient of determination R^2 of the prediction
## best score is 1.0
lr_confidence = lr.score(x_test, y_test)
print("lr confidence: ", lr_confidence)
```

```
lr confidence:  0.9994618614480701
```

Looks like in this case the Linear Regression model will be better to use to predict the future price of Apple stock, because it's score is closer to 1.0.

Now I am ready to do some forecasting. I will take the last 30 rows of data from the data frame of the Close price, and store it into a variable called x_forecast after transforming it into a numpy array and dropping the 'Prediction' column of course. Then I will print the data to make sure the 30 rows are all there.

```
#forcase data
x_forecast = np.array(df.drop(['Prediction'],1))[-forecast_out:]
#print(x_forecast)
```

```
[[258.44000244]
 [247.74000549]
 [254.80999756]
 [254.28999329]
 [240.91000366]
 [244.92999268]
 [241.41000366]
 [262.47000122]
 [259.42999268]
 [266.07000732]
 [267.98999023]
 [273.25      ]
 [287.04998779]
 [284.42999268]
 [286.69000244]
 [282.79998779]
 [276.92999268]
 [268.36999512]
 [276.1000061 ]
 [275.02999878]
 [282.97000122]
 [283.17001343]
 [278.57998657]
 [287.73001099]
 [293.79998779]
 [289.07000732]
 [293.16000366]
 [297.55999756]
 [300.63000488]
 [303.77999878]]
```

I will print out the future price (next 10 days) predictions of Apple stock using the linear regression model using the x_forecast data !