EXPERIMENT-6

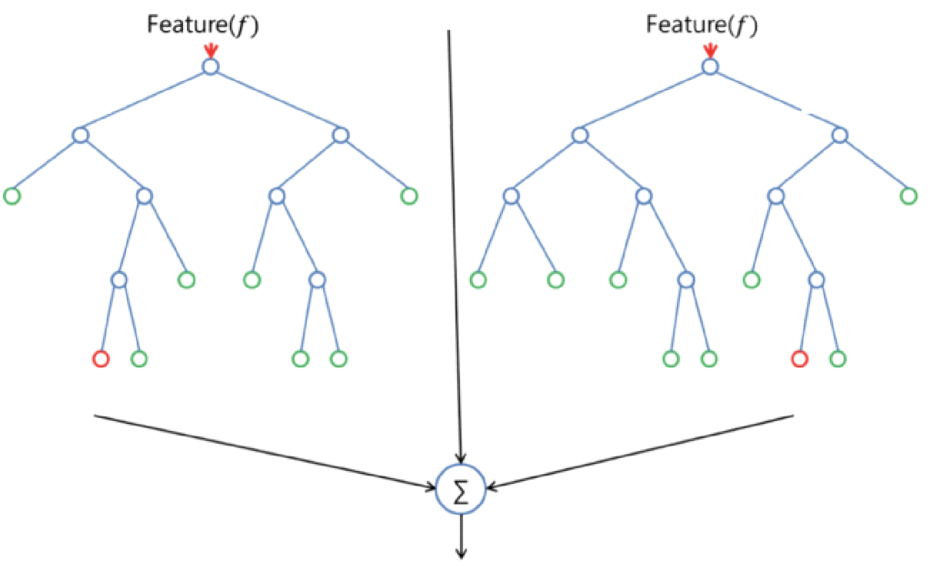
Problem Statement

Develop a machine learning method to predict stock price based on past price variation.

Algorithm

Random forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because of its simplicity and diversity (it can be used for both classification and regression tasks). In this post we'll learn how the random forest algorithm works, how it differs from other algorithms and how to use it.

One big advantage of random forest is that it can be used for both classification and regression problems, which form the majority of current machine learning systems. Let's look at random forest in classification, since classification is sometimes considered the building block of machine learning. Below you can see how a random forest would look like with two trees:



Random forest has nearly the same hyperparameters as a decision tree or a bagging classifier. Fortunately, there's no need to combine a decision tree with a bagging classifier because you can easily use the classifier-class of random forest. With random forest, you can also deal with regression tasks by using the algorithm's regressor.

Program Snippet

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
df= pd.read_csv('stock.csv')
df
```

Out[2]:

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
0	2018-10-08	208.00	222.25	206.85	216.00	215.15	4642146	10062.83
1	2018-10-05	217.00	218.60	205.90	210.25	209.20	3519515	7407.06
2	2018-10-04	223.50	227.80	216.15	217.25	218.20	1728786	3815.79
3	2018-10-03	230.00	237.50	225.75	226.45	227.60	1708590	3960.27
4	2018-10-01	234.55	234.60	221.05	230.30	230.90	1534749	3486.05
...	...	...	...	...	...	...	...	...
1230	2013-10-14	160.85	161.45	157.70	159.30	159.45	1281419	2039.09
1231	2013-10-11	161.15	163.45	159.00	159.80	160.05	1880046	3030.76
1232	2013-10-10	156.00	160.80	155.85	160.30	160.15	3124853	4978.80
1233	2013-10-09	155.70	158.20	154.15	155.30	155.55	2049580	3204.49
1234	2013-10-08	157.00	157.80	155.20	155.80	155.80	1720413	2688.94

1235 rows × 8 columns

In [3]:
```
df.head()
```

Out[3]:

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
0	2018-10-08	208.00	222.25	206.85	216.00	215.15	4642146	10062.83
1	2018-10-05	217.00	218.60	205.90	210.25	209.20	3519515	7407.06
2	2018-10-04	223.50	227.80	216.15	217.25	218.20	1728786	3815.79
3	2018-10-03	230.00	237.50	225.75	226.45	227.60	1708590	3960.27
4	2018-10-01	234.55	234.60	221.05	230.30	230.90	1534749	3486.05

In [4]:
```
df.tail()
```

Out[4]:

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
1230	2013-10-14	160.85	161.45	157.70	159.3	159.45	1281419	2039.09
1231	2013-10-11	161.15	163.45	159.00	159.8	160.05	1880046	3030.76
1232	2013-10-10	156.00	160.80	155.85	160.3	160.15	3124853	4978.80
1233	2013-10-09	155.70	158.20	154.15	155.3	155.55	2049580	3204.49
1234	2013-10-08	157.00	157.80	155.20	155.8	155.80	1720413	2688.94

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1235 entries, 0 to 1234
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Date                  1235 non-null   object
 1   Open                  1235 non-null   float64
 2   High                  1235 non-null   float64
 3   Low                   1235 non-null   float64
 4   Last                  1235 non-null   float64
 5   Close                 1235 non-null   float64
 6   Total Trade Quantity  1235 non-null   int64
 7   Turnover (Lacs)       1235 non-null   float64
dtypes: float64(6), int64(1), object(1)
memory usage: 77.3+ KB
```

In [6]:
```
df.shape
```

Out[6]:
```
(1235, 8)
```

In [7]:
```
df.columns.values
```

Out[7]:
```
array(['Date', 'Open', 'High', 'Low', 'Last', 'Close',
       'Total Trade Quantity', 'Turnover (Lacs)'], dtype=object)
```

In [8]:
```
df.corr()
```

Out[8]:

	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
Open	1.000000	0.998956	0.998776	0.997662	0.997704	0.367503	0.587026
High	0.998956	1.000000	0.998728	0.999130	0.999159	0.388798	0.605907
Low	0.998776	0.998728	1.000000	0.999008	0.999065	0.361695	0.582446
Last	0.997662	0.999130	0.999008	1.000000	0.999963	0.381269	0.599575
Close	0.997704	0.999159	0.999065	0.999963	1.000000	0.380801	0.599155
Total Trade Quantity	0.367503	0.388798	0.361695	0.381269	0.380801	1.000000	0.941976
Turnover (Lacs)	0.587026	0.605907	0.582446	0.599575	0.599155	0.941976	1.000000

In [9]:
```
df['Date']= pd.to_datetime(df.Date, format= '%Y-%m-%d')
df.index = df['Date']
df
```

Out[9]:

Date	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
2018-10-08	2018-10-08	208.00	222.25	206.85	216.00	215.15	4642146	10062.83
2018-10-05	2018-10-05	217.00	218.60	205.90	210.25	209.20	3519515	7407.06
2018-10-04	2018-10-04	223.50	227.80	216.15	217.25	218.20	1728786	3815.79
2018-10-03	2018-10-03	230.00	237.50	225.75	226.45	227.60	1708590	3960.27
2018-10-01	2018-10-01	234.55	234.60	221.05	230.30	230.90	1534749	3486.05
...	...	...	...	...	...	...	...	...
2013-10-14	2013-10-14	160.85	161.45	157.70	159.30	159.45	1281419	2039.09
2013-10-11	2013-10-11	161.15	163.45	159.00	159.80	160.05	1880046	3030.76
2013-10-10	2013-10-10	156.00	160.80	155.85	160.30	160.15	3124853	4978.80
2013-10-09	2013-10-09	155.70	158.20	154.15	155.30	155.55	2049580	3204.49
2013-10-08	2013-10-08	157.00	157.80	155.20	155.80	155.80	1720413	2688.94

1235 rows × 8 columns

In [10]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(16,8))
plt.plot(df['Close'],label='CloseHistory')
```

Out[10]: [<matplotlib.lines.Line2D at 0x1507bbbd0a0>]

Long Short Term Memory (LSTM)

LSTMs are widely used for sequence prediction problems and have proven to be extremely effective. The reason they work so well is because LSTM is able to store past information that is important, and forget the information that is not. LSTM has three gates:

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

In [11]:
```
data= df.sort_index(ascending=True, axis=0)
new_data = pd.DataFrame(index=range(0,len(df)), columns=['Date','Close'])
new_data
```

Out[11]:

	Date	Close
0	NaN	NaN
1	NaN	NaN
2	NaN	NaN
3	NaN	NaN
4	NaN	NaN
...	...	...
1230	NaN	NaN
1231	NaN	NaN
1232	NaN	NaN
1233	NaN	NaN
1234	NaN	NaN

1235 rows × 2 columns

In [12]:
```
for i in range(0,len(data)):
  new_data['Date'][i]= data['Date'][i]
  new_data['Close'][i]= data['Close'] [i]
new_data
```

Out[12]:

	Date	Close
0	2013-10-08 00:00:00	155.8
1	2013-10-09 00:00:00	155.55
2	2013-10-10 00:00:00	160.15
3	2013-10-11 00:00:00	160.05
4	2013-10-14 00:00:00	159.45
...	...	...
1230	2018-10-01 00:00:00	230.9
1231	2018-10-03 00:00:00	227.6
1232	2018-10-04 00:00:00	218.2
1233	2018-10-05 00:00:00	209.2
1234	2018-10-08 00:00:00	215.15

1235 rows × 2 columns

In [13]:
```
new_data.index = new_data.Date
new_data.drop('Date', axis=1, inplace=True)
```

In [14]:
```
dataset = new_data.values
train = dataset[0:987,:]
valid = dataset[987:,:]
```

In [15]:
```
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM
```

In [16]:
```
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(dataset)
```

In [17]:
```
x_train, y_train = [], []
for i in range(60,len(train)):
    x_train.append(scaled_data[i-60:i,0])
    y_train.append(scaled_data[i,0])
x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0],x_train.shape[1],1))
```

In [18]:
```
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1],1)))
model.add(LSTM(units=50))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(x_train, y_train, epochs=1, batch_size=1, verbose=2)
```

```
927/927 - 50s - loss: 0.0012
```

Out[18]:
```
<keras.callbacks.History at 0x1504549d1f0>
```

In [19]:
```
inputs = new_data[len(new_data) - len(valid) - 60:].values
inputs = inputs.reshape(-1,1)
inputs  = scaler.transform(inputs)
```

In [20]:
```
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 = model.predict(X_test)
closing_price = scaler.inverse_transform(closing_price)
```

In [21]:
```
train = new_data[:987]
valid = new_data[987:]
valid['Predictions'] = closing_price
plt.plot(train['Close'])
plt.plot(valid[['Close','Predictions']])
```

```
<ipython-input-21-f5cac910112e>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  valid['Predictions'] = closing_price
```

Out[21]:
```
[<matplotlib.lines.Line2D at 0x1504c112c40>,
 <matplotlib.lines.Line2D at 0x1504c112c70>]
```

GitHub link-

https://github.com/pranshuag9/machine-learning-lab/blob/main/lab7/Machine%20Learning%20Experiment%206.ipynb