Share_market

Ramanathan N

In [1]:

```
import numpy as np
import pandas as pd
import datetime
import quandl

%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
plt.style.use('seaborn-darkgrid')
plt.rc('figure', figsize=(16,10))
plt.rc('lines', markersize=4)
```

In [2]:

```
%pylab inline
start_date = datetime.date(2017,1,1)
end_date = datetime.date(2018,12,28)
data = pd.read_csv("D:\google chrome\BAJAJ_AUTO.csv", low_memory = False, skiprows =
#data = quandl.get("BSE/BOM532814", authtoken="43vNrp7GWQtzPqyguXPB")
print("The GTD dataset has {} samples with {} features.".format(*data.shape))
```

```
Populating the interactive namespace from numpy and matplotlib
The GTD dataset has 2347 samples with 9 features.
```

In [3]:

```
data=data.iloc[0:2350]
data=data.iloc[::-1]
data.head()
```

Out[3]:

	Date	Open	High	Low	Close	Volume	Adjustment Factor	Adjustment Type	Unn
2346	01-01-2010	669.844609	669.844609	669.844609	669.844609	0	NaN	NaN	
2345	04-01-2010	673.738166	676.772851	658.488399	661.160448	381510	NaN	NaN	
2344	05-01-2010	667.554280	670.741654	652.743492	664.214219	463938	NaN	NaN	
2343	06-01-2010	668.012346	668.012346	650.892144	653.945915	465832	NaN	NaN	
2342	07-01-2010	657.705870	659.996198	639.421418	642.475188	329288	NaN	NaN	

In [4]:

```
data=data.drop(['Adjustment Factor','Volume'], axis=1)
data = pd.DataFrame(data,columns=['Close','High','Low','Open'])
```

In [5]:

```
# calculate momentum for each day
# 5-day momentum

def momentum(df):
```

```
    n = len(df)
    arr = []
    for i in range(0,5):
        arr.append('N')
    for j in range(5,n):
        momentum = df.Close[j] - df.Close[j-5] #Equation for momentum
        arr.append(momentum)
    return arr

momentum = momentum(data)

# add momentum to data
data['Momentum'] = momentum
```

In [6]:
```
#Use pct_change() function to add the one day returns to the dataframe

data_pctchange=data.Close.pct_change()
data['Return'] = data_pctchange
```

In [7]:
```
#ROI function

def ROI(df,n):
    m = len(df)
    arr = []
    for i in range(0,n):
        arr.append('N')
    for j in range(n,m):
        roi= (df.Close[j] - df.Close[j-n])/df.Close[j-n] #Equation for ROI
        arr.append(roi)
    return arr

#Run the ROI function for 10, 20, and 30 day periods

ROI10=ROI(data,10)
ROI20=ROI(data,20)
ROI30=ROI(data,30)


#Add all 3 ROI results to dataframe

data['10 Day ROI']=ROI10
data['20 Day ROI']=ROI20
data['30 Day ROI']=ROI30
```

In [8]:
```
# calculate RSI for each day


def RSI(df,period):
    # get average of upwards of last 14 days: Ct - Ct-1
    # get average of downwards of last 14 days: Ct-1 - Ct
    n = len(df)
    arr = []
    for i in range(0,period):
        arr.append('N')
    for j in range(period,n):
        total_upwards = 0
        total_downwards = 0
        # this will find average of upwards
        for k in range(j,j-period,-1):
            if(df.Close[k-1] > df.Close[k]):
                total_downwards = total_downwards + (df.Close[k-1] - df.Close[k])
        avg_down = total_downwards / period
        for l in range(j,j-period,-1):
            if(df.Close[l] > df.Close[l-1]):
```

```
                total_upwards = total_upwards + (df.Close[l] - df.Close[l-1])
            avg_up = total_upwards / period
            RS = avg_up / avg_down
            RSI  = 100 - (100/(1+RS))
            arr.append(RSI)
    return arr


#Run RSI for 10, 14, and 30 day periods

RSI_14 = RSI(data,14)
RSI_10 = RSI(data,10)
RSI_30 = RSI(data,30)

# add RSI to data

data['10_day_RSI'] = RSI_10
data['14_day_RSI'] = RSI_14
data['30_day_RSI'] = RSI_30
```

In [9]:

```
# calculate EMA for each day
# formula: EMA = (2/(n+1))*ClosePrice + (1-(2/(n+1)))*previousEMA

def EMA(df, n):
    m = len(df)
    arr = []
    arr.append('N')
    prevEMA = df.Close[0]
    for i in range(1,m):
        close = df.Close[i]
        EMA = ((2/(n+1))*close) + ((1-(2/(n+1)))*prevEMA)
        arr.append(EMA)
        prevEMA = EMA
    return arr

#Calculate EMA with n=12 and n=26

EMA_12 = EMA(data, 12)
EMA_26 = EMA(data, 26)

#add EMA to dataframe

data['EMA_12'] = EMA_12
data['EMA_26'] = EMA_26
```

In [10]:

```
#Function to Classify each day as a 1 or a 0

def clas(df):
    n = len(df)
    arr = []
    for i in range(0,len(df)-1):
        if (100*((df.Close[i+1]-df.Open[i+1])/df.Open[i+1]))>=.3:
            arr.append(1)
        else:
            arr.append(0)
    arr.append('N')
    return arr

clas=clas(data)

#Add Class to our dataframe
data['Class'] = clas
```

In [11]:

```
#SRSI: Stochastic RSI
```

```python
#SRSI = (RSI_today - min(RSI_past_n)) / (max(RSI_past_n) - min(RSI_past_n))
def SRSI(df,n):
    m = len(df)
    arr = []
    list_RSI = RSI(df,n)
    for i in range(0,n):
        arr.append('N')
    for j in range(n,n+n):
        last_n = list_RSI[n:j]
        if(not(last_n == []) and not(max(last_n) == min(last_n))):
            SRSI = (list_RSI[j] - min(last_n)) / (max(last_n)- min(last_n))
            if SRSI > 1:
                arr.append(1)
            else:
                arr.append(SRSI)
        else:
            arr.append(0)
    for j in range(n+n,m):
        last_n = list_RSI[2*n:j]
        if(not(last_n == []) and not(max(last_n) == min(last_n))):
            SRSI = (list_RSI[j] - min(last_n)) / (max(last_n)- min(last_n))
            if SRSI > 1:
                arr.append(1)
            else:
                arr.append(SRSI)
        else:
            arr.append(0)
    return arr

#Run SRSI for 10, 14, and 30 day periods
SRSI_10 = SRSI(data,10)
SRSI_14 = SRSI(data,14)
SRSI_30 = SRSI(data,30)

#Add SRSI to our dataframe
data['SRSI_10'] = SRSI_10
data['SRSI_14'] = SRSI_14
data['SRSI_30'] = SRSI_30
```

In [12]:

```python
# calculate Williams %R oscillator for each day

def Williams(df,n):
    m = len(df)
    arr = []
    for i in range(0,n-1):
        arr.append('N')
    for j in range(n-1,m):
        maximum = max(data.High[(j-n+1):j+1])
        minimum = min(data.Low[(j-n+1):j+1])
        val = (-100)*(maximum-df.Close[j])/(maximum-minimum)
        arr.append(val)
    return arr


williams = Williams(data,14)

#Add Williams%R to our dataframe
data['Williams'] = williams
```

In [13]:

```python
# True Range
# TR = MAX(high[today] - close[yesterday]) - MIN(low[today] - close[yesterday])
def TR(df,n):
    high = df.High[n]
    low = df.Low[n]
```

```
    close = df.Close[n-1]
    l_max = list()
    l_max.append(high)
    l_max.append(close)
    l_min = list()
    l_min.append(low)
    l_min.append(close)
    return (max(l_max) - min(l_min))

# Average True Range
# Same as EMA except use TR in lieu of close (prevEMA = TR(dataframe,14days))
def ATR(df,n):
    m = len(df)
    arr = []
    prevEMA = TR(df,n+1)
    for i in range(0,n):
        arr.append('N')
    for j in range(n,m):
        TR_ = TR(df,j)
        EMA = ((2/(n+1))*TR_) + ((1-(2/(n+1)))*prevEMA)
        arr.append(EMA)
        prevEMA = EMA
    return arr

ATR = ATR(data,14)

#Add ATR to our dataframe
data['ATR_14'] = ATR
```

In [14]:
```
#double check that the dataframe has all 21 features
data.shape
```

Out[14]: (2347, 20)

In [15]:
```
#def normalization function to clean data
def normalize(df):
    for column in df:
        df[column]=((df[column]-df[column].mean())/df[column].std())
```

In [16]:
```
#def positive values for running Multinomial Naive Bayes
def positivevalues(df):
    for column in df:
        if (df[column].min())<0:
            df[column]=(df[column]-df[column].min())
```

In [17]:
```
#Remove the first 30 index which could have a value 'N'
newdata=data.drop(data.index[0:30])

#Remove the last row of data because class has value 'N'
newdata=newdata.drop(newdata.index[-1])

#Remove 'High' and 'Low' columns to improve the algorithm
newdata=newdata.drop(['High','Low'], axis=1)

#Remove our 'Class' column because it acts as y in our algorithms
newdata=newdata.drop(['Class'], axis=1)

#check the features that remain in our algorithm
newdata.head()
```

Out[17]:

	Close	Open	Momentum	Return	10 Day ROI	20 Day ROI	30 Day ROI	10_day_R

	Close	Open	Momentum	Return	10 Day ROI	20 Day ROI	30 Day ROI	10_day_R
2316	682.804048	682.804048	28.2225	0.000000	-0.0205958	-0.0567768	-0.0148889	42.042
2315	684.292761	683.281200	98.2873	0.002180	-0.0183378	-0.0452686	-0.00815087	43.007
2314	686.144110	690.915627	73.4815	0.002705	-0.020246	-0.0589718	-0.0248995	42.442
2313	702.042804	687.098413	67.2566	0.023171	0.0278813	-0.0603497	-0.0212463	64.381
2312	691.869930	704.352218	-0.93607	-0.014490	0.0253303	-0.0270477	-0.0217092	62.70

In [18]:
```
#Normalize the data that we have filtered
normalize(newdata)
```

In [19]:
```
#Put the dataframe with our relevant features into X and our class into our y
X=newdata
y=clas[30:-1]
```

In [20]:
```
#Split up our test and train by splitting 70%/30%

X_train=X.drop(X.index[1211:])
X_test=X.drop(X.index[0:1211])
y_train=y[0:1211]
y_test=y[1211:]
```

In [21]:
```
#Import and run Logistic Regression and run a fit to train the model
from sklearn.linear_model import LogisticRegression

LR=LogisticRegression()
LR.fit(X_train,y_train)
```

Out[21]:
```
LogisticRegression()
```

In [22]:
```
#Predict the y test
y_pred_LR=LR.predict(X_test)
```

In [23]:
```
#Print the accuracy score of our predicted y using metrics from sklearn
from sklearn import metrics
print (metrics.accuracy_score(y_test, y_pred_LR))
```

```
0.6235294117647059
```

In [24]:
```
#Import and run Gaussian Naive Bayes and run a fit to train the model
from sklearn.naive_bayes import GaussianNB

GNB = GaussianNB()
GNB.fit(X_train,y_train)
```

Out[24]:
```
GaussianNB()
```

In [25]:
```
#Predict the y test
y_pred=GNB.predict(X_test)
```

In [26]:
```
#Print the accuracy score of our predicted y using metrics from sklearn
from sklearn import metrics
print (metrics.accuracy_score(y_test, y_pred))
```

```
0.6208144796380091
```