In [1]:

```python
import numpy as np
import pandas as pd
import math
from matplotlib import pyplot
import csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import MinMaxScaler

import time
```

In [2]:

```python
#Function to get the dot product of weight matrix and input matrix
def zipper(x,w):
    perceptronrule_Mul=np.dot(w,x)
    return perceptronrule_Mul
```

In [15]:

```python
#This function is basically the perceptron rule and is used to train the model and provide a final weig
ht matrix that can be
#used for caluclating testing accuracy during testing phase
def LMS(feature,lr,target,epochs):
    start_time = time.time()


    bias=0
    numb_featureval=len(feature.values[0])
    weight_value=[]
    rmse_error=[]


    accuracy={}
    acc=[]
    max_acc=0

    w=[0 for i in range(numb_featureval)] # Initializing the weight matrix with values as 1 with the nu
mber of values being the number of inputs

    #Iterating the epochs for training
    for itera in range(epochs):
        predicted_label=[]
        error_value=[]
        for x,y in zip(feature.values,target.values): # iterating the input values and the target value
s simultaneously
                                                        #so that the values can be compared for the same
 index.

            product=zipper(x,w) #zipper function called which returns the dot product of weight and inp
ut which is then compared
                                #with the bias to finally get the predicted label

            predicted_class= 1 if product>0 else -1

            #storing the predicted values in a list corresponding to their row index so that it can be
 compared to its actual target values
            predicted_label.append(predicted_class)

            error=0

            #Weight and bias updation if the prdicted value doesnt match with the target labels

            error=y-product  #calculating error which is (desired output- input*weight)
            w=w+(lr*x*error) #weight updation as per the error

            error_value.append(error) #appending the errors for a particular epoch into a list, which i
s further used for rmse calculation
        square=0
        mean=0
        root=0

        #Squaring and adding the values of errors, followed by taking mean and root of the computed ans
wer
        for i in error_value:
            square += (i**2)
        #Calculate Mean
        mean = (square / (float)(len(error_value)))
        #Calculate Root
        root = math.sqrt(mean)
        rmse_error.append(root)


        #Calling the accuracy_score function of sklearn to calculate the accuracy for each epoch by pas
sing the predicted labels list
        #the target labels as arguements

        acc.append(accuracy_score(predicted_label,target.values))

        weight_value.append(w)

        #print(root)
        #checking and storing the maximum accuracy and getting the epoch with the maximum accuracy to e
xtract the weights at that epoch
        if(acc[itera-1]>max_acc):

            max_acc=acc[itera-1]
            epoch_max_acc=itera


    elapsed_time = time.time() - start_time

    print("********************************************* Training Starts here *************************
*********************")
    print("")
    print("RMSE values as per Epochs: ")
    print("")
    for i in rmse_error:
        print(i)
    print("")
    print("Accuracy: ",max_acc,"Error rate: ",(1-max_acc))
    print("Training Time: ",round(elapsed_time,3))
    return weight_value[epoch_max_acc] #Returning the weight that led to maximum accuracy
```

In [16]:

```python
#This function uses parameters like radius, ditance between two moons, number of samples in total and w
idth as per which
#half moon will be generated and the data points will be stored along with their labels.

features=3 #number of attributes +label column
instances=3000
r=10
d=0
w=4



#First we need to check whether the number of samples are even or not
if (instances%2!=0) :
    print("*****Error****** Number of samples are not valid; They should be even ")
    instances=instances+1

#Matrix initialization of samples with 0 as initial value for x,y and label values.
valuesofSamples=np.zeros((features,instances),dtype=int)
#print(valuesofSamples)

# Boundary condition checking
if (r<w/2):
    print("*****Error******  Radius is not enough")

#Creating random float values of x and y coordinates of half the instances
randomval=np.random.random((2,int(instances/2)))
#print (randomval)

radii=(r-w/2)+w*randomval[0][:]

#Calculating outer radius for one half moon
theta=np.pi*randomval[1][:]

#Creation of datasets for both the half moons
x_class1=np.multiply(radii,np.cos(theta)) #X coordinate for 1st half data points
y_class1=np.multiply(radii,np.sin(theta)) #y coordinate for 1st half data points
label_class1=np.ones((1,len(x_class1)),dtype=int) #providing label as 1 to the entire 1st half moon dat
a
label_class1=np.hstack(label_class1)

x_class2=np.multiply(radii,np.cos(-theta))+r #X coordinate for 2nd half data points
y_class2=np.multiply(radii,np.sin(-theta))-d #y coordinate for 2nd half data points
label_class2=-1*np.ones((1,len(x_class2)),dtype=int) #providing label as -1 to the 2nd half moon data
label_class2=np.hstack(label_class2)



#Now we will create a single matrix with all the x,y coordinates of all the points belonging to both th
e halves
#using a nested list functionality,with their corresponding labels
valuesofSamples[0,:]=np.concatenate([x_class1,x_class2])
valuesofSamples[1,:]=np.concatenate([y_class1,y_class2])

valuesofSamples[2,:]=np.concatenate(([label_class1,label_class2]))

#converting to dataframe and Transposing it to get columns on the top
df=(pd.DataFrame(valuesofSamples)).T

DF=df.rename(columns={0:'x',1:'y',2:'labels'}) #Renaming the column name as per index

scalerObjct=MaxAbsScaler()  #Normalizing thee valus of the datasetusing MaxAbsScaler
Normalizeddf=scalerObjct.fit_transform(DF)
df_scaled = pd.DataFrame(Normalizeddf, columns=DF.columns)

df_scaled = df_scaled.astype({"labels": int}) #Converting label column values to int
df_scaled.sample(frac=1) # Randomizing the whole dataset
DF_train,DF_test=train_test_split(df_scaled, test_size=0.6665) #splitting the dataset with 1000 trainin
g and 2000 testing data points
#Separating the feature data and label data for the training set
feature_valTrain=DF_train[['x','y']]
Label_train=DF_train['labels']


final_weight=LMS(feature_valTrain,0.001,Label_train,50) #Storing the weight matrix that led to maximum
 accuracy
                                                                    #during training phase when Perceptr
onRule is called with
                                                                    #feature, learning rate as 0.01, lab
el and epoch as 600 as arguements.

#Separating the feature data and label data for the testing set
feature_valTest=DF_test[['x','y']]
Label_test=DF_test['labels']

list_test=[] #list to store predicted labels during testing 2000 data points with their original labels

#Predicting values with test data and caluclating accuracy similar to training phase but with weights f
ixed as final_weights
#received from training phase after calculating accuracy

calculated_out=[] #to store i*w while testing and then using it for rmse

test_time=time.time() #Starting the timer to calculate the testing time

for x,l in zip(feature_valTest.values, Label_test.values):
    result=zipper(x,final_weight)
    answ=1 if result>0 else -1
    list_test.append(answ)
    calculated_out.append(result)

test_accuracy=accuracy_score(list_test,Label_test.values)
mse = mean_squared_error(calculated_out, Label_test.values) #Getting mse calculated
rmse=np.sqrt(mse)
endtime_test=time.time()-test_time #calculating testing time

print("")
print("")
print("********************************************* Testing starts here ***************************
*********************")
print("")
print("RMSE: ", rmse)
print("Accuracy: ",test_accuracy,"Error rate : ",(1-test_accuracy))
print("Testing Time: ",round(endtime_test,3))
```

```
******************************************* Training Starts here **********************************
************

RMSE values as per Epochs:

0.8630311787990561
0.6662219971683386
0.5617888084663495
0.5102875886784338
0.486010951371721
0.4746683563776775
0.4692404360202154
0.4665077306612291
0.4650324712155386
0.4641719625433881
0.4636325432878378
0.4632743231808747
0.4630264900381704
0.4628504046258794
0.4627232486627239
0.4626305543915394
0.46256262585431734
0.46251270938812716
0.46247598293666625
0.46244895162466737
0.4624290603124906
0.4624144317447294
0.46240368306042173
0.4623957942371512
0.46239001246555717
0.4623857820868395
0.46238269303051655
0.46238044277553786
0.46237880825292305
0.46237762507964075
0.4623767722077471
0.4623761605768431
0.4623757247270894
0.46237541660183407
0.4623752009695732
0.4623750520433139
0.4623749509851752
0.4623748840652772
0.4623748413040972
0.46237481547194603
0.46237480135215864
0.4623747951989298
0.46237479433876005
0.46237479687780675
0.46237480148727617
0.4623748072462923
0.46237481352706405
0.46237481991114215
0.4623748261285152
0.4623748320134509

Accuracy:  0.971 Error rate:  0.029000000000000026
Training Time:  0.839


******************************************* Testing starts here **********************************
*****************

RMSE:  0.4520868067533158
Accuracy:  0.9705 Error rate :  0.029499999999999997
Testing Time:  0.008
```

In [ ]:

In [ ]:

In [ ]: