In [54]:
```
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
import sys
```

In [55]:
```
# Real life data is generally non-linealrly separable but using internet I g
ot sonar dataset which is
#linearly separable
# I have used sonar dataset which is linearly separable and normalized
# It can be downloaded from
# https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+%28Sonar%2C+Mi
nes+vs.+Rocks%29
# https://archive.ics.uci.edu/ml/machine-learning-databases/undocumented/con
nectionist-bench/sonar/

data = pd.read_csv("/home/ankit/Desktop/sonar.csv")
data.columns=['x'+str(x) for x in range(60)]+['Material']
#data
data.shape
```

Out[55]: (207, 61)

In [56]:
```
data1= data.to_numpy() #converting dataframe "data" to numpy array "data1"
dataset=[]
bias=1
for i in range(len(data1)):
    temp=[]
    temp.append(bias)
    for j in range(61):
        temp.append(data1[i][j])
    dataset.append(temp) # appending 1st column as bias=1 and all other colu
mns of matrix data1
    # and created a new matrix of size 207*62 for our purpose
#print(dataset)
```

In [57]:
```
# the original dataset has class-labels as "M" and "R", so we have to conver
t it into 1 and 0
for i in range(len(dataset)):
    if dataset[i][-1]=='M':
        dataset[i][-1]=1
    else:
        dataset[i][-1]=0
```

In [58]:
```
# here dataset has 62 dimensions which is much harder to visualize.
# if it is of 2 dimensions then I can use plt.scatter() to visualize it easi
ly
# for data visualization purpose, we have to reduce the dimensions using PCA
or t-SNE like techniques
# and then visualize it
```

In [59]:
```
# Implementing perceptron model

weights=[] # initializing weights
for i in range(62):
    weights.append(np.random.randn())
```

In [60]:
```
 # I have commented the print statements because datset is of size 207*62 an
d
 # for each iteration printing weight values, accuracy etc will take much ce
ll space and
#   browser's tab will be crashed.
```

In [73]:
```
def predict(inputs,weights): # This function takes values of features (x0,x
1,x2,...x61)

    #one by one from dataset and weight vector as input and return the value
according
    #to the definition of standard sigmoid function

    threshold = 0.0 # I have set the threshold as 0
    v = 0.0
    for input,weight in zip(inputs,weights):
        v += input*weight
    return 1 if v >= threshold else 0.0

def accuracy(matrix,weights): # This function gives the accuracy on the scal
e of 0 to 1
    # in terms of how many points are correctly out of total number of data
points

    num_correct=0.0 #initialized total correct points as 0
    preds=[]
    for i in range(len(matrix)):
        pred=predict(matrix[i][:-1],weights)
        preds.append(pred)
        if pred==matrix[i][-1]: num_correct += 1.0 #if data point is correct
ly classified then number of
            #correctly classified points are added by 1

    #print("Predictions:",preds)
    return num_correct/float(len(matrix))

def train_weights(matrix,weights,iterations=10000,l_rate=2.0): # This functi
on is to train weights
    #According to perceptron convergence theorem, after number of iteration
s, maximum accuracy will be reached
    # and after that weight vector will not be changed. Since, generated dat
a is of random numbers.
    #So, each time when we run the cell, dataset will be different and so, n
umber of iterations
    #to get the estimated weight vector with accuracy 1.0 will be diferent a
nd so accordingly you can change
    # the value of number of iterations.
    for epoch in range(iterations):
        cur_acc=accuracy(matrix,weights)
        #print("\nIteration  %d \nWeights:  "%epoch,weights)
        #print("Accuracy: ", cur_acc )
        if cur_acc==1.0 and True : break

        for i in range(len(matrix)):
            prediction = predict(matrix[i][:-1],weights)
            error = matrix[i][-1] - prediction #to get the difference betwee
n predicted and actual class-label

            #if True:
                #print("Training on data at index %d..."%i)
            for j in range(len(weights)):
                #if True:
                    #sys.stdout.write("\t Weight[%d]: %0.5f ---> "%(j,weight
s[j]))
                weights[j]= weights[j] + (l_rate*error*matrix[i][j]) # weigh
t is modified
                #according to the definition
                #if True: sys.stdout.write("%0.5f\n"%weights[j])
    print("\n Final estimated weight vector with accuracy %0.5f is:"%cur_ac
c)
    return weights
```

In [74]:
```
train_weights(dataset, weights=weights,iterations=10000,l_rate=2.0)
```

```
Final estimated weight vector with accuracy 0.76812 is:
```

Out[74]:
```
[-203.17110395184255,
 818.7882306233622,
 69.60006214532568,
 -914.3413837569893,
 423.7397133399329,
 63.57050550888441,
 3.4016365867032436,
 -301.1331489812688,
 -62.76645755855038,
 77.25285500541368,
 -112.39771248969848,
 149.92561204842283,
 238.43852940057386,
 -47.97981890622493,
 -131.7664667678387,
 290.1308160550853,
 -293.96850558758916,
 -126.69448440074946,
 258.99623614762976,
 -87.34567419086481,
 263.8929677822673,
 -302.45759333133907,
 364.69990211088157,
 -277.42034703862413,
 226.31865937949067,
 65.8573420008936,
 -218.66026513585024,
 223.62338409452164,
 -114.56406398119087,
 -95.90795696615383,
 409.46303496452174,
 -435.68070100246143,
 64.93810886537776,
 283.0438971028584,
 -285.9652428058532,
 212.96430768963387,
 -121.49174291845327,
 -99.98076412648497,
 -26.09526699775043,
 172.47015492688797,
 -208.47688873041471,
 123.6674724683513,
 -63.65433106709056,
 68.24175262405726,
 109.65606297772256,
 -195.51060532126593,
 496.6918478010961,
 -403.18530102973506,
 535.0791266496789,
 1061.5258858452055,
 -2103.7004603361393,
 443.39808052848355,
 1011.4019796309334,
 899.4584747837401,
 246.52898639842738,
 513.1434143644434,
 -46.04851682022659,
 -817.90591011236,
 1144.8225459179941,
 874.214068349856,
 576.5851885334675,
 721054.536820923]
```

In [ ]:

```
```