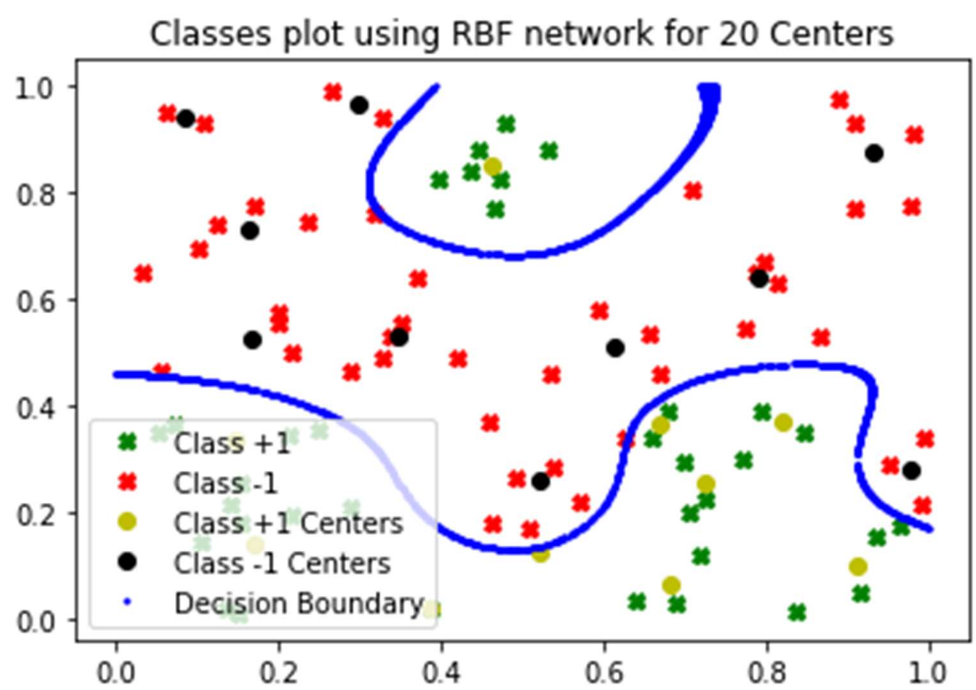
PURVANG LAPSIWALA

UIN 662689378

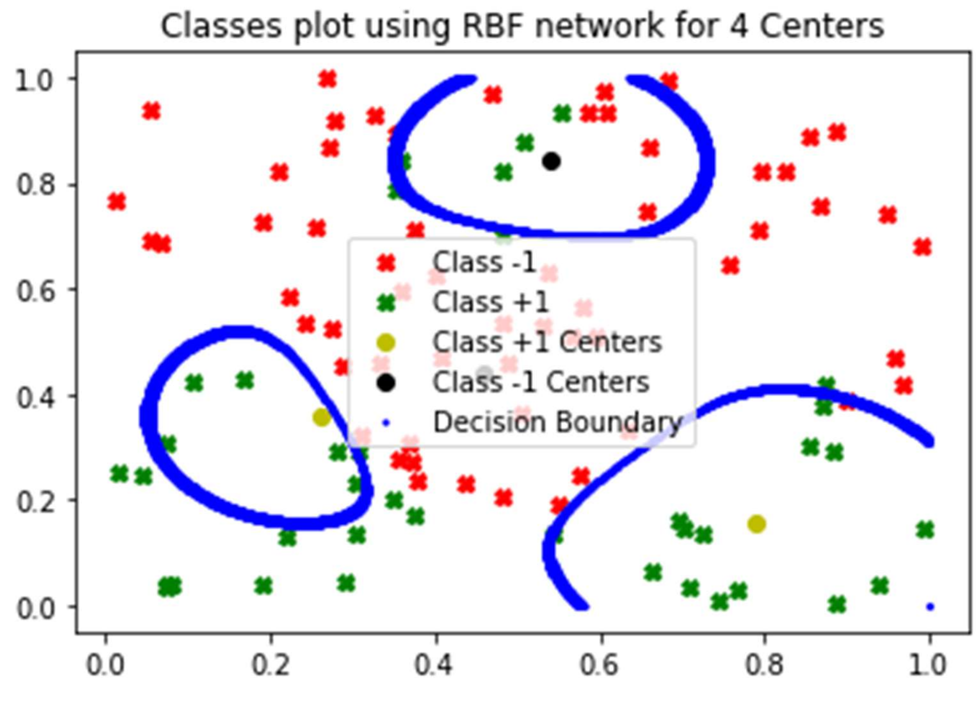
By classifying given input point in 2 classes, each class having 10 clusters and assign given input to most nearby and appropriate cluster by using k means algorithm, we can classify data points using radial basis network which is by research most appropriate and suitable for classification as well as interpolation problems in neural network. In radial basis network, type of kernel used is gaussian kernel with mean and standard deviation parameters. In gaussian radial basis function, Each RBF neuron stores a "prototype" vector which is just one of the vectors from the training set. Each RBF neuron compares the input vector to its prototype, and outputs a value between 0 and 1 which is a measure of similarity. The prototype vector is also often called the neuron's "center", since it's the value at the center of the bell curve. To find the centers, K means algorithm is used and each training input is assigned to individual center based on distance. And then with those updated centers, gaussian kernel is made and each output of gaussian kernel is treated as input to PTA and weights are adjusted such that error is minimum for classification.

For the 10 centers per each class with total 20 centers, it takes 16 epochs to correctly classify all input and to make error zero.

While doing this project, it is observed that there must be minimum number of centers available to correctly classify the patterns. As per the code and parameters used in the above example, there must be 6 centers per class is required to correctly classify with zero error.



But, with the same parameters, when given inputs are assigned to just two centers and trying to classify them, it is observed that network never converge and there are always some misclassifications and output saturates with maximum accuracy of around 80%. So, with just 2 centers per class, it is not possible to correctly classify with zero error.



Python Code:

```
import numpy.random as random

import numpy as np

import matplotlib.pyplot as pylab

import math


n=100

eta=1

resolution=600


x= np.zeros((n,2))
```

```
for j in range(n):
   temp = (1-0)*random.sample(2) + 0
   x[j] = temp
x=np.matrix(np.array(x))

theta = random.uniform(-1,1)



def weights(m):
   w=[]
   for i in range(2*m):
      w.append(random.uniform(-1,1))
   return w

def signum(g):
   if(g<0):
      act=-1
   else:
      act=1
   return act

def update_center(index_c,c_final):
   f=0
   for value in index_c:
      x[value]=c_final[f]
      f+=1

def kmeans(c,x):
   count=0
```

```
flag=1
while(flag):
   A={}
   for i in range(m):
      A.setdefault(i,[])


   for i in range(len(x)):
      dist=[]
      for j in range(len(c)):
         dist.append(np.linalg.norm(x[i]-c[j]))
      A[np.argmin(dist)].append(x[i])

   out=0.0
   for i in range(m):
      sum=0
      if(len(A[i])!=0):
         for value in A[i]:
            sum=sum+value
         avg=sum/(len(A[i]))
         diff=np.linalg.norm(avg-c[i])
         c[i]=avg
         out=diff+out
   if out==0.0:
      flag=0


   count+=1
   print("Count",count)
```

```
    return c


def sun_mountain(m):
    p=0
    q=0
    pylab.title("Classes plot using RBF network for 20 Centers")
    s=0
    t=0
    for j in range(n):
        if (x[j,1]<((math.sin(10*x[j,0]))/5)+0.3) or ((math.pow((x[j,1]-0.8),2)+math.pow((x[j,0]-
0.5),2))<math.pow(0.15,2)):

            des.append(1)
            if p<m:
                c1[p]=x[j]
                index_c1.append(j)
                p+=1
            else:
                x1.append(x[j])
                pylab.plot(x[j,0],x[j,1],'gX',label='Class +1'if s==0 else "")
                s=1
        else:

            des.append(-1)
            if q<m:
                c1n[q]=x[j]
                index_c1n.append(j)
                q+=1
            else:
```

```
            x1n.append(x[j])
            pylab.plot(x[j,0],x[j,1],'rX',label='Class -1'if t==0 else "")
            t=1


def RBF_PTA():
    epoch=0
    flag=1
    beta = 1.3
    variance = beta*np.var(x)
    while(flag):
        errors=0
        actop=[]

        for i in range(n):
            g=0
            for j in range(2*m):
                rbf= math.exp(-((np.linalg.norm(x[i]-c_union[j])**2)/(2*(variance**2))))
                g= (w[j]*rbf)+g
            g=g+theta
            actop.append(signum(g))

        for i in range(n):
            for j in range(2*m):
                rbf= math.exp(-((np.linalg.norm(x[i]-c_union[j])**2)/(2*(variance**2))))
                w[j]=w[j]+(eta*rbf*(des[i]-actop[i]))

            if des[i]!=actop[i]:
                errors+=1
```

```
        epoch+=1
        print("Epoch",epoch,"errors",errors,"accuracy",(n-errors),"%")
        if errors < 1 :
            flag=0

    h=0
    for x1 in range(resolution):
        for x2 in range(resolution):
            g=0
            x1t=x1/resolution
            x2t=x2/resolution
            arr=np.array([x1t,x2t])
            for j in range(2*m):
                rbf= math.exp(-((np.linalg.norm(arr-c_union[j])**2)/(2*(variance**2))))
                g= (w[j]*rbf)+g
            g=g+theta
            if(g < 0.05 and g > -0.05):
                pylab.plot(x1t,x2t,'b.',markersize=3,label='Decision Boundary' if h==0 else "")
                h=1
        print("epoch",x1)
    pylab.legend(loc='best')
    pylab.show()


m=10 # No. of centers in each class
x1=[]
x1n=[]
des=[]
```

```
index_c1=[]
index_c1n=[]
c1= np.zeros((m,2))
c1n= np.zeros((m,2))

sun_mountain(m)
x1=np.matrix(np.array(x1))
x1n=np.matrix(np.array(x1n))

c1_final=kmeans(c1,x1)
c1n_final=kmeans(c1n,x1n)

for i in range(m):
    pylab.plot(c1_final[i,0],c1_final[i,1],'yo',label='Class +1 Centers' if i==0 else "")
    pylab.plot(c1n_final[i,0],c1n_final[i,1],'ko',label='Class -1 Centers'if i==0 else "")

update_center(index_c1,c1_final)
update_center(index_c1n,c1n_final)

c_union=np.concatenate((c1_final,c1n_final))

w=weights(m)

RBF_PTA()
```