ECE 559 NEURAL NETWORKS ASSIGNMENT 8

NAME:LAKSHMI SRIDEVI UIN:668383906

1. RBF NETWORK FOR K=20

Importing libraries

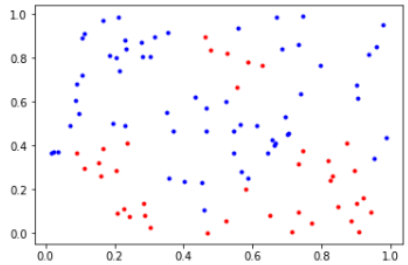
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
import random
```

Initializing variables

In [2]:
```
x=np.random.rand(2,100)
d=np.zeros((100,1))
d1=np.zeros((100,1))
D=np.zeros((100,1))
Cp=[]
Cn=[]
```

Plotting the given data points

In [3]:
```
X,Y=np.meshgrid(np.linspace(0,1,100),np.linspace(0,1,100))
sun=np.square(X-0.8)+np.square(Y-0.5)-np.square(0.15)
#plt.contour(Y,X,sun,[0])

y=np.linspace(0,1,100)
mount=((1/5)*np.sin(10*y))+0.3
#plt.plot(y,mount)

for i in range(len(x[0])):

    d[i][0]=((1/5)*np.sin(10*x[0][i]))+0.3
    d1[i][0]=np.square(x[1][i]-0.8)+np.square(x[0][i]-0.5)

    if x[1][i] < d[i][0] or d1[i][0] < np.square(0.15):
        D[i][0]=1
        Cp.append((x[0][i],x[1][i]))
        cp=np.asarray(Cp)
        plt.plot(x[0][i],x[1][i],'r.')

    else:
        D[i][0]=-1
        Cn.append((x[0][i],x[1][i]))
        cn=np.asarray(Cn)
        plt.plot(x[0][i],x[1][i],'b.')
```



Picking the centroid from the given set of datapoints

In [4]:
```
index_p=np.random.choice(len(cp),10,replace=0)
print(index_p)

index_n=np.random.choice(len(cn),10,replace=0)
print(index_n)
```

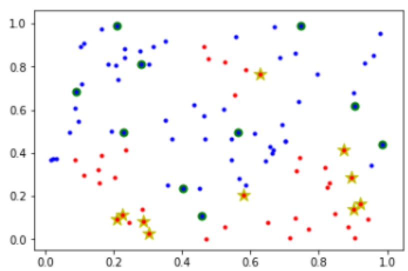
```
[32 36  9 18  7 30 31 14  0 34]
[ 3 31 41 30 33 19 28 40  2 26]
```

In [5]:
```
center_p=np.zeros((10,2))
center_n=np.zeros((10,2))
for i in range(10):
        center_p[i]=Cp[index_p[i]]
        center_n[i]=Cn[index_n[i]]
```

Plotting The chosen centrod points

In [6]:
```
plt.plot(cp[:,0],cp[:,1],'r.')
plt.scatter(center_p[:,0],center_p[:,1],marker='*',c='y',s=150)
plt.plot(cn[:,0],cn[:,1],'b.')
plt.scatter(center_n[:,0],center_n[:,1],marker='o',c='g',s=50)
```

Out[6]: <matplotlib.collections.PathCollection at 0x1e53b28b278>



Finding the new cluster centers for the positive class

In [7]:
```
k_p=10
center_p_old=np.zeros(center_p.shape)
center_p_new=center_p
cluster_p=np.zeros(len(cp))
dist_p=np.zeros((len(cp),k_p))
error_p = np.linalg.norm(center_p_new - center_p_old)

while error_p != 0:
    # Measure the distance to every center
    for i in range(k_p):
        dist_p[:,i] = np.linalg.norm(cp - center_p_new[i], axis=1)
    # Assign all training data to closest center
    cluster_p = np.argmin(dist_p, axis = 1)

    center_p_old = center_p_new
    # Calculate mean for every cluster and update the center
    for i in range(k_p):
        center_p_new[i] = np.mean(cp[cluster_p == i], axis=0)
    error_p = np.linalg.norm(center_p_new - center_p_old)
print('New cluster centers:',center_p_new)
print('Updated clusters',cluster_p)
```

```
New cluster centers: [[0.83719803 0.07111055]
 [0.28514692 0.11024899]
 [0.16827221 0.30596912]
 [0.38583172 0.01426605]
 [0.58440672 0.11377916]
 [0.81988667 0.28753747]
 [0.9193406  0.16370677]
 [0.8089592  0.39475176]
 [0.53898096 0.79552125]
 [0.22629527 0.08491042]]
Updated clusters [8 2 0 2 2 0 5 4 0 2 8 2 4 9 7 8 0 5 3 5 2 3 8 0 0 7 8 1 0 8 5 6 0 2 9 2 1
 5 4]
```

Finding the new cluster centers for the negative class

In [8]:

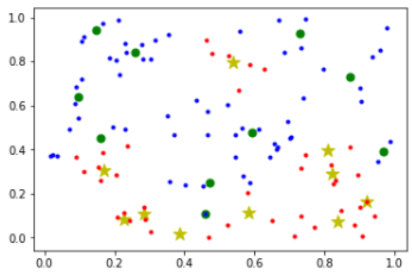
```
k_n=10
center_n_old=np.zeros(center_n.shape)
center_n_new=center_n
cluster_n=np.zeros(len(cn))
dist_n=np.zeros((len(cn),k_n))
error_n = np.linalg.norm(center_n_new - center_n_old)

while error_n != 0:
    # Measure the distance to every center
    for i in range(k_n):
        dist_n[:,i] = np.linalg.norm(cn - center_n_new[i], axis=1)
    # Assign all training data to closest center
    cluster_n = np.argmin(dist_n, axis = 1)

    center_n_old = center_n_new
    # Calculate mean for every cluster and update the center
    for i in range(k_n):
        center_n_new[i] = np.mean(cn[cluster_n == i], axis=0)
    error_n = np.linalg.norm(center_n_new - center_n_old)
print('New cluster centers:',center_n_new)
print('Updated clusters',cluster_n)
```

```
New cluster centers: [[0.4581932  0.10651131]
 [0.09557216 0.63916909]
 [0.59208229 0.47519462]
 [0.14803881 0.94097195]
 [0.72823202 0.92843928]
 [0.16143654 0.45270432]
 [0.47292781 0.25118446]
 [0.25984404 0.83889265]
 [0.96911661 0.39074612]
 [0.87237019 0.72824571]]
Updated clusters [7 6 8 0 4 2 2 9 5 2 9 7 8 4 9 7 1 2 1 5 4 2 3 7 3 7 9 5 6 3 3 1 2 4 2 4 2
 1 5 9 7 2 2 5 9 7 7 7 6 6 2 5 5 2 2 2 2 4 6 7 5]
```

Plotting the updated centroids

In [9]:
```
plt.plot(cp[:,0],cp[:,1],'r.')
plt.scatter(center_p_new[:,0],center_p_new[:,1],marker='*',c='y',s=150)
plt.plot(cn[:,0],cn[:,1],'b.')
plt.scatter(center_n[:,0],center_n[:,1],marker='o',c='g',s=50)
```

Out[9]: <matplotlib.collections.PathCollection at 0x1e53b2f2e48>



Concatenating the centers of both positive and negative classes

In [10]:
```
center=np.row_stack((center_p_new,center_n_new))
print(center)
```

```
[[0.83719803 0.07111055]
 [0.28514692 0.11024899]
 [0.16827221 0.30596912]
 [0.38583172 0.01426605]
 [0.58440672 0.11377916]
 [0.81988667 0.28753747]
 [0.9193406  0.16370677]
 [0.8089592  0.39475176]
 [0.53898096 0.79552125]
 [0.22629527 0.08491042]
 [0.4581932  0.10651131]
 [0.09557216 0.63916909]
 [0.59208229 0.47519462]
 [0.14803881 0.94097195]
 [0.72823202 0.92843928]
 [0.16143654 0.45270432]
 [0.47292781 0.25118446]
 [0.25984404 0.83889265]
 [0.96911661 0.39074612]
 [0.87237019 0.72824571]]
```

Computing the RBF function

In [28]:
```
x_mu=np.zeros((100,len(center)))
for i in range(100):
    for j in range(len(center)):
        x_mu[i][j]=np.linalg.norm(x[:,i]-center[j,:])
rbf=np.exp(-16*np.square(x_mu))
```

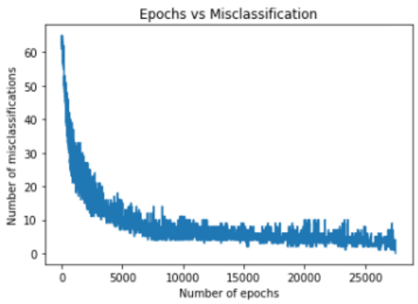
Perceptron training algorithm

In [32]:
```
K=20
W=np.zeros((K,1))
W=np.random.normal(size=(K,1))              #Initializing the weight vector
bias=np.random.randn(1)
errors=0                                    #Initializing the misclassifications
Mc=[]                                       #Storing the misclassification values after every epoch
epoch=0
lr=0.01
#rbf=np.row_stack([rbf_p,rbf_n])
```

In [35]:
```
while(epoch==0 or Mc[epoch-1] > 0):
#     errors=0
    for i in range(len(rbf)):
        #errors=0
        y=np.sign(np.matmul(W.reshape(1,20),rbf[i].reshape(20,1))+bias)
        #Y=np.heaviside(y,1)
        print('actual',y)
        E=D[i]-y
        print('Error',E)
        Q=E*rbf[i].reshape(20,1)
        W=W+lr*Q
        bias=bias+lr*E
        errors = 0
        for j in range(len(rbf)):
            Y=np.sign(np.matmul(W.reshape(1,20),rbf[j].reshape(20,1))+bias)
            if Y != D[j]:
                errors=errors+1
#                 print('count',errors)

        Mc.append(errors)
        print('Misclassification',Mc[epoch])
        epoch=epoch+1
```

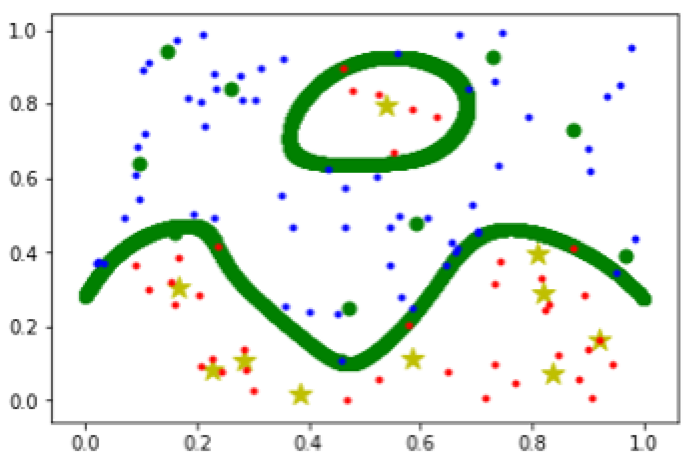
Plotting Number of epochs against Misclassifications

In [36]:
```
plt.plot(Mc)
plt.xlabel('Number of epochs')
plt.ylabel('Number of misclassifications')
plt.title('Epochs vs Misclassification')
plt.show()
```



Plotting the Decision Boundary

In [42]:
```
plt.plot(cp[:,0],cp[:,1],'r.')
plt.scatter(center_p_new[:,0],center_p_new[:,1],marker='*',c='y',s=150)
plt.plot(cn[:,0],cn[:,1],'b.')
plt.scatter(center_n[:,0],center_n[:,1],marker='o',c='g',s=50)
K_mu=0

for i in np.linspace(0,1,1000):
    for j in np.linspace(0,1,1000):
        g=0
        for k in range(len(center)):
            K_mu=np.linalg.norm(np.asarray([i,j])-center[k,:])
            RBF=np.exp(-16*np.square(K_mu))
            g=g+np.multiply(W[k],RBF)
        g=g+bias
        print('G:',g)
        if np.abs(g)<0.005:
            plt.scatter(i,j,c='g')
```



2. RBF NETWORK FOR K=4

Importing Libraries

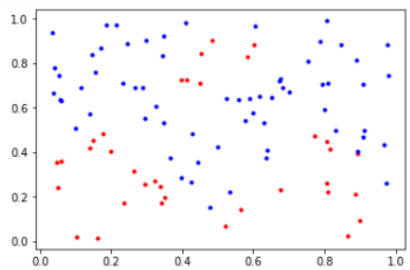
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
import random
```

Initializing variables

In [2]:
```
x=np.random.rand(2,100)
d=np.zeros((100,1))
d1=np.zeros((100,1))
D=np.zeros((100,1))
Cp=[]
Cn=[]
```

Plotting the given data points

In [3]:
```
X,Y=np.meshgrid(np.linspace(0,1,100),np.linspace(0,1,100))
sun=np.square(X-0.8)+np.square(Y-0.5)-np.square(0.15)
#plt.contour(Y,X,sun,[0])

y=np.linspace(0,1,100)
mount=((1/5)*np.sin(10*y))+0.3
#plt.plot(y,mount)

for i in range(len(x[0])):

    d[i][0]=((1/5)*np.sin(10*x[0][i]))+0.3
    d1[i][0]=np.square(x[1][i]-0.8)+np.square(x[0][i]-0.5)

    if x[1][i] < d[i][0] or d1[i][0] < np.square(0.15):
        D[i][0]=1
        Cp.append((x[0][i],x[1][i]))
        cp=np.asarray(Cp)
        plt.plot(x[0][i],x[1][i],'r.')

    else:
        D[i][0]=-1
        Cn.append((x[0][i],x[1][i]))
        cn=np.asarray(Cn)
        plt.plot(x[0][i],x[1][i],'b.')
```



Picking the centroid from the given set of points

In [4]:
```
index_p=np.random.choice(len(cp),2,replace=0)
print(index_p)

index_n=np.random.choice(len(cn),2,replace=0)
print(index_n)
```

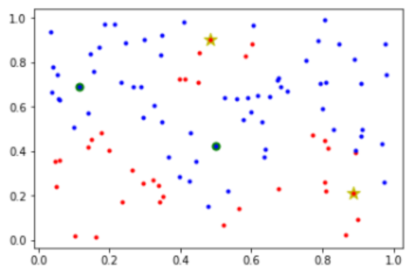
```
[17  6]
[63 10]
```

In [5]:
```
center_p=np.zeros((2,2))
center_n=np.zeros((2,2))
for i in range(2):
        center_p[i]=Cp[index_p[i]]
        center_n[i]=Cn[index_n[i]]
```

Plotting the centroids

In [6]:
```
plt.plot(cp[:,0],cp[:,1],'r.')
plt.scatter(center_p[:,0],center_p[:,1],marker='*',c='y',s=150)
plt.plot(cn[:,0],cn[:,1],'b.')
plt.scatter(center_n[:,0],center_n[:,1],marker='o',c='g',s=50)
```

Out[6]:
```
<matplotlib.collections.PathCollection at 0x1ef9669c320>
```



Finding the cluster centers for positive class

In [7]:
```
k_p=2
center_p_old=np.zeros(center_p.shape)
center_p_new=center_p
cluster_p=np.zeros(len(cp))
dist_p=np.zeros((len(cp),k_p))
error_p = np.linalg.norm(center_p_new - center_p_old)

while error_p != 0:
    # Measure the distance to every center
    for i in range(k_p):
        dist_p[:,i] = np.linalg.norm(cp - center_p_new[i], axis=1)
    # Assign all training data to closest center
    cluster_p = np.argmin(dist_p, axis = 1)

    center_p_old = center_p_new
    # Calculate mean for every cluster and update the center
    for i in range(k_p):
        center_p_new[i] = np.mean(cp[cluster_p == i], axis=0)
    error_p = np.linalg.norm(center_p_new - center_p_old)
print('New cluster centers:',center_p_new)
print('Updated clusters',cluster_p)
```

```
New cluster centers: [[0.29813113 0.57743738]
 [0.57296773 0.21630267]]
Updated clusters [0 0 0 1 0 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 0 0 1 0 1 1 1 1 1 1 0 1 0 0 0]
```

Finding cluster centers for the negative class

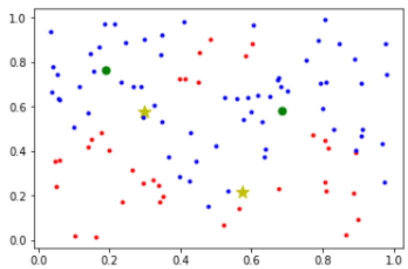
In [8]:
```
k_n=2
center_n_old=np.zeros(center_n.shape)
center_n_new=center_n
cluster_n=np.zeros(len(cn))
dist_n=np.zeros((len(cn),k_n))
error_n = np.linalg.norm(center_n_new - center_n_old)

while error_n != 0:
    # Measure the distance to every center
    for i in range(k_n):
        dist_n[:,i] = np.linalg.norm(cn - center_n_new[i], axis=1)
    # Assign all training data to closest center
    cluster_n = np.argmin(dist_n, axis = 1)

    center_n_old = center_n_new
    # Calculate mean for every cluster and update the center
    for i in range(k_n):
        center_n_new[i] = np.mean(cn[cluster_n == i], axis=0)
    error_n = np.linalg.norm(center_n_new - center_n_old)
print('New cluster centers:',center_n_new)
print('Updated clusters',cluster_n)
```

```
New cluster centers: [[0.19039576 0.76472406]
 [0.68544756 0.5807676 ]]
Updated clusters [0 1 0 1 0 1 1 0 1 1 1 0 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0
 0 1 0 0 1 0 1 1 1 1 0 0 1 1 1 0 0 1 0 1 1 1 0 0 1 1 0 0]
```

Plotting the updated centers

In [9]:
```
plt.plot(cp[:,0],cp[:,1],'r.')
plt.scatter(center_p_new[:,0],center_p_new[:,1],marker='*',c='y',s=150)
plt.plot(cn[:,0],cn[:,1],'b.')
plt.scatter(center_n[:,0],center_n[:,1],marker='o',c='g',s=50)
```

Out[9]:
```
<matplotlib.collections.PathCollection at 0x1ef96703ef0>
```



Concatenating the updated centers of positive and negative class

In [10]:
```
center=np.row_stack((center_p_new,center_n_new))
print(center)
```

```
[[0.29813113 0.57743738]
 [0.57296773 0.21630267]
 [0.19039576 0.76472406]
 [0.68544756 0.5807676 ]]
```

Computing the RBF function

In [11]:
```
x_mu=np.zeros((100,len(center)))
for i in range(100):
    for j in range(len(center)):
        x_mu[i][j]=np.linalg.norm(x[:,i]-center[j,:])
rbf=np.exp(-16*np.square(x_mu))
```

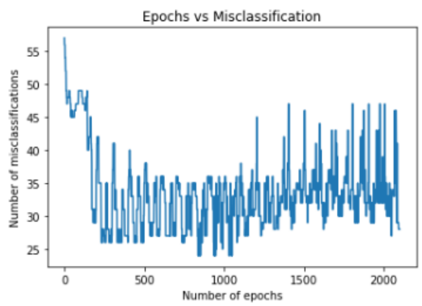
Perceptron training algorithm

In [13]:
```
K=4
W=np.zeros((K,1))
W=np.random.normal(size=(K,1))               #Initializing the weight vector
bias=np.random.randn(1)
errors=0                                  #Initializing the misclassifications
Mc=[]                                     #Storing the misclassification values after every epoch
epoch=0
lr=0.01
#rbf=np.row_stack([rbf_p,rbf_n])
```

In [15]:
```
while(epoch==0 or Mc[epoch-1] > 28):
#     errors=0
    for i in range(len(rbf)):
        #errors=0
        y=np.sign(np.matmul(W.reshape(1,4),rbf[i].reshape(4,1))+bias)
        #Y=np.heaviside(y,1)
        print('actual',y)
        E=D[i]-y
        print('Error',E)
        Q=E*rbf[i].reshape(4,1)
        W=W+lr*Q
        bias=bias+lr*E
        errors = 0
        for j in range(len(rbf)):
            Y=np.sign(np.matmul(W.reshape(1,4),rbf[j].reshape(4,1))+bias)
            if Y != D[j]:
                errors=errors+1
#                 print('count',errors)

        Mc.append(errors)
        print('Misclassification',Mc[epoch])
        epoch=epoch+1
```

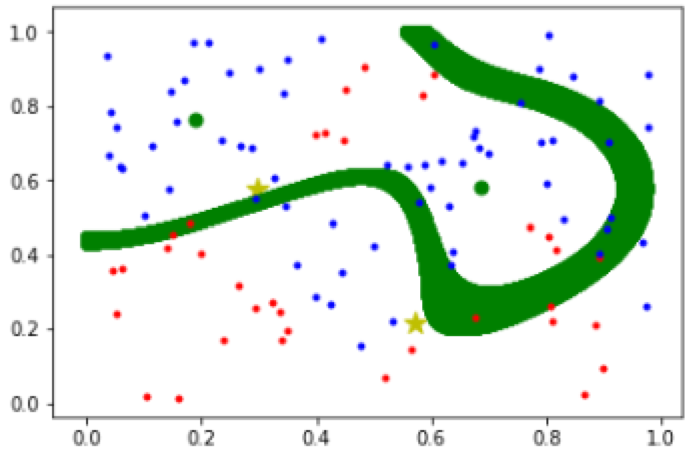
Plotting number of epochs against misclassification

In [20]:
```
plt.plot(Mc)
plt.xlabel('Number of epochs')
plt.ylabel('Number of misclassifications')
plt.title('Epochs vs Misclassification')
plt.show()
```



Plotting the decision boundary

In [21]:
```
plt.plot(cp[:,0],cp[:,1],'r.')
plt.scatter(center_p_new[:,0],center_p_new[:,1],marker='*',c='y',s=150)
plt.plot(cn[:,0],cn[:,1],'b.')
plt.scatter(center_n[:,0],center_n[:,1],marker='o',c='g',s=50)
K_mu=0

for i in np.linspace(0,1,1000):
    for j in np.linspace(0,1,1000):
        g=0
        for k in range(len(center)):
            K_mu=np.linalg.norm(np.asarray([i,j])-center[k,:])
            RBF=np.exp(-16*np.square(K_mu))
            g=g+np.multiply(W[k],RBF)
        g=g+bias
        print('G:',g)
        if np.abs(g)<0.005:
            plt.scatter(i,j,c='g')
```



Observations:

1. Gaussian Radial Basis Function was used in designing this RBF network.

   $\emptyset(||X - Ci||) = e^{-\beta(||X - Ci||^2)}$, where i=1,2,..,total number of centers

2. It is seen that the network generated an accurate data classification boundary for the data points with 20 centers. As we decrease the number of clusters to 4, the network doesnt show a converging behavior, thereby not providing an accurate classification boundary.