LAB – 07 – RADIAL BASIS FUNCTION NETWORK

1. Implement RBF network for classification. Use your own dataset
2. Compare the performance of RBF with Multi Layer Perceptron

RBF network code:-

```
import numpy as np
import pcn             #using perceptron network
import kmeans          # and kmeans clustering algorithm

class rbf:
    """ The Radial Basis Function network
    Parameters are number of RBFs, and their width, how to train the network
    (pseudo-inverse or kmeans) and whether the RBFs are normalised"""

    def __init__(self,inputs,targets,nRBF,sigma=0,usekmeans=0,normalise=0):
        self.nin = np.shape(inputs)[1]
        self.nout = np.shape(targets)[1]
        self.ndata = np.shape(inputs)[0]
        self.nRBF = nRBF
        self.usekmeans = usekmeans
        self.normalise = normalise

        if usekmeans:
            self.kmeansnet = kmeans.kmeans(self.nRBF,inputs)

        self.hidden = np.zeros((self.ndata,self.nRBF+1))

        if sigma==0:
            # Set width of Gaussians
            d = (inputs.max(axis=0)-inputs.min(axis=0)).max()
            self.sigma = d/np.sqrt(2*nRBF)
        else:
            self.sigma = sigma

        self.perceptron = pcn.pcn(self.hidden[:,:-1],targets)

        # Initialise network
        self.weights1 = np.zeros((self.nin,self.nRBF))
```

```
    def rbftrain(self,inputs,targets,eta=0.25,niterations=100):
        if self.usekmeans==0:
            # Version 1: set RBFs to be datapoints
            indices = range(self.ndata)
            np.random.shuffle(indices)
            for i in range(self.nRBF):
                self.weights1[:,i] = inputs[indices[i],:]
        else:
            # Version 2: use k-means
            self.weights1 = np.transpose(self.kmeansnet.kmeanstrain(inputs))

        for i in range(self.nRBF):
            self.hidden[:,i] = np.exp(-np.sum((inputs - np.ones((1,self.nin))*self.weights1[:,i])**2,axis=1)/(2*self.sigma**
        if self.normalise:
            self.hidden[:,:-1] /= np.transpose(np.ones((1,np.shape(self.hidden)[0]))*self.hidden[:,:-1].sum(axis=1))

        # Call Perceptron without bias node (since it adds its own)
        self.perceptron.pcntrain(self.hidden[:,:-1],targets,eta,niterations)

    def rbffwd(self,inputs):

        hidden = np.zeros((np.shape(inputs)[0],self.nRBF+1))

        for i in range(self.nRBF):
            hidden[:,i] = np.exp(-np.sum((inputs - np.ones((1,self.nin))*self.weights1[:,i])**2,axis=1)/(2*self.sigma**2))

        if self.normalise:
            hidden[:,:-1] /= np.transpose(np.ones((1,np.shape(hidden)[0]))*hidden[:,:-1].sum(axis=1))

        # Add the bias
        hidden[:,-1] = -1
        outputs = self.perceptron.pcnfwd(hidden)
        return outputs
```

```python
    def confmat(self,inputs,targets):
        """Confusion matrix"""

        outputs = self.rbffwd(inputs)
        nClasses = np.shape(targets)[1]

        if nClasses==1:
            nClasses = 2
            outputs = np.where(outputs>0,1,0)
        else:
            # 1-of-N encoding
            outputs = np.argmax(outputs,1)
            targets = np.argmax(targets,1)

        cm = np.zeros((nClasses,nClasses))
        for i in range(nClasses):
            for j in range(nClasses):
                cm[i,j] = np.sum(np.where(outputs==i,1,0)*np.where(targets==j,1,0))

        output = cm
        print("Confusion matrix is:")
        print(cm)
        print("Percentage Correct: ", np.trace(cm) / np.sum(cm) * 100)
        return output
```
[37] ✓ 0.1s Python

Using the banknote dataset:-

The Banknote Dataset involves predicting whether a given banknote is authentic given a number of measures taken from a photograph.

It is a binary (2-class) classification problem. The number of observations for each class is not balanced. There are 1,372 observations with 4 input variables and 1 output variable. The variable names are as follows:

1. Variance of Wavelet Transformed image (continuous).
2. Skewness of Wavelet Transformed image (continuous).
3. Kurtosis of Wavelet Transformed image (continuous).
4. Entropy of image (continuous).
5. Class (0 for authentic, 1 for inauthentic).

The baseline performance of predicting the most prevalent class is a classification accuracy of approximately 50%.

[UCI Machine Learning Repository: banknote authentication Data Set]

```python
iris = np.loadtxt('banknote.csv',delimiter=',')
iris[:,:4] = iris[:,:4]-iris[:,:4].mean(axis=0)
imax = np.concatenate((iris.max(axis=0)*np.ones((1,5)),iris.min(axis=0)*np.ones((1,5))),axis=0).max(axis=0)
iris[:,:4] = iris[:,:4]/imax[:4]
print (iris[0:5,:])
```
[47] ✓ 0.1s Python

```
[[ 0.49880026  0.61144219 -0.25438505  0.20451374  0.        ]
 [ 0.64342405  0.56622605 -0.23328978 -0.07427406  0.        ]
 [ 0.53704115 -0.4135054   0.03185603  0.3565094   0.        ]
 [ 0.47298296  0.68911749 -0.32721727 -0.6598847   0.        ]
 [-0.01635021 -0.57824013  0.19202762  0.05571211  0.        ]]
```

```python
iris.shape
```
[58] ✓ 0.9s Python

```
(1372, 5)
```

```python
target = np.zeros((np.shape(iris)[0], 2))
indices = np.where(iris[:,4]==0)
target[indices,0] = 1
indices = np.where(iris[:,4]==1)
target[indices,1] = 1
```

```python
order = np.arange(np.shape(iris)[0])
np.random.shuffle(order)
iris = iris[order,:]
target = target[order,:]
```

[49] ✓ 0.8s Python

```python
train = iris[::2,0:4]
traint = target[::2]
valid = iris[1::4,0:4]
validt = target[1::4]
test = iris[3::4,0:4]
testt = target[3::4]
```

[50] ✓ 0.7s Python

```python
print (train.max(axis=0), train.min(axis=0))
```

[51] ✓ 0.1s Python

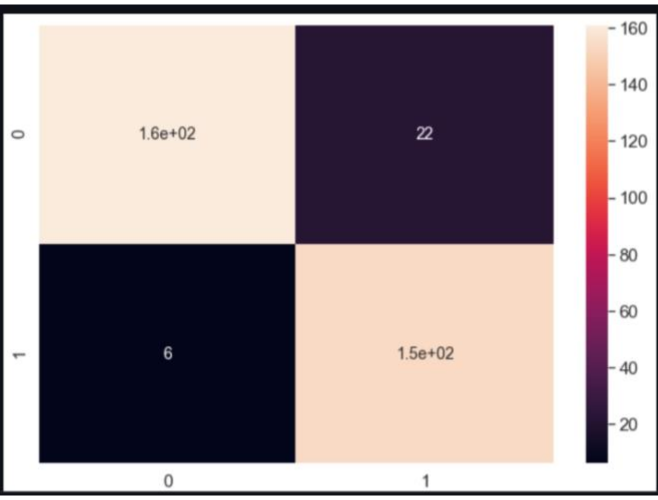
```
[1.         1.         0.9805079   0.92117889] [-1.16973236 -1.41444409 -0.40284444 -1.803395   ]
```

```python
net = rbf(train,traint,5,1,1)

net.rbftrain(train,traint,0.25,5000)
print("Train data:-")
net.confmat(train,traint)
print("Test data:-")
cm = net.confmat(test,testt)
```

[60] ✓ 0.8s Python

```
Train data:-
Confusion matrix is:
[[364.  37.]
 [ 29. 256.]]
Percentage Correct:  90.37900874635568
Test data:-
Confusion matrix is:
[[161.  22.]
 [  6. 154.]]
Percentage Correct:  91.83673469387756
```

Performance metrics of the network:-



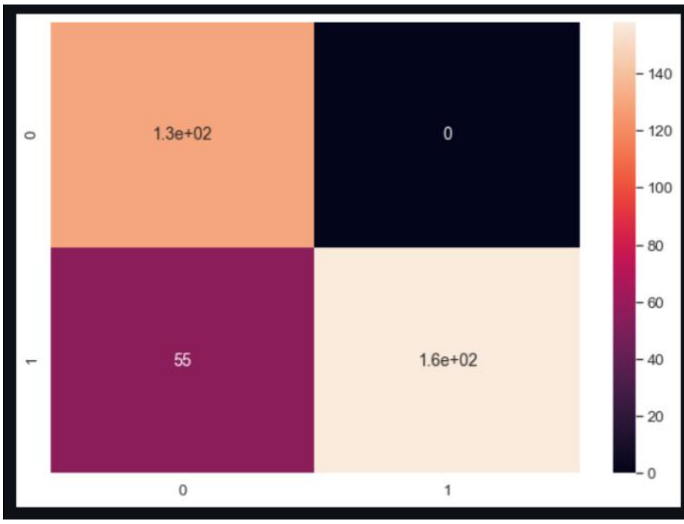
```python
from sklearn.metrics import classification_report
targets=testt
inputs = test
nClasses = np.shape(targets)[1]
outputs = net.rbffwd(inputs)
if nClasses==1:
    nClasses = 2
    outputs = np.where(outputs>0,1,0)
else:
    # 1-of-N encoding
    outputs = np.argmax(outputs,1)
    targets = np.argmax(targets,1)


print(classification_report(targets, outputs))
```

```
              precision    recall  f1-score   support

           0       0.88      0.96      0.92       167
           1       0.96      0.88      0.92       176

    accuracy                           0.92       343
   macro avg       0.92      0.92      0.92       343
weighted avg       0.92      0.92      0.92       343
```

Now comparing the network with MLP for the same data:-

```python
# Train the network
import MLP as mlp
net = mlp.mlp(train,traint,20, outtype='logistic')
net.earlystopping(train,traint,valid,validt,0.1)
cm = net.confmat(test,testt)
```

```
1
Iteration:  0  Error:  172.25374267512274
2
Iteration:  0  Error:  186.5038862009149
3
Iteration:  0  Error:  186.50163225703525
Stopped 102.01329341965177 102.01294511540866 102.00591661968596
Confusion matrix is:
[[130.   0.]
 [ 55. 158.]]
Percentage Correct:  83.96501457725948
```



```python
from sklearn.metrics import classification_report
targets=testt
inputs = np.concatenate((test, -np.ones((np.shape(test)[0], 1))), axis=1)
nclasses = np.shape(targets)[1]
output = net.mlpfwd(inputs)
if nclasses == 1:
    nclasses = 2
    output = np.where(output > 0.5, 1, 0)
else:
    # 1-of-N encoding
    output = np.argmax(output, 1)
    targets = np.argmax(targets, 1)


print(classification_report(targets, output))
```

0.2s Python

```
              precision    recall  f1-score   support

           0       1.00      0.70      0.83       185
           1       0.74      1.00      0.85       158

    accuracy                           0.84       343
   macro avg       0.87      0.85      0.84       343
weighted avg       0.88      0.84      0.84       343
```

Here we can clearly see that using RBF network has given a significantly better accurate output for the same training data. Also notable is that MLP was fully precise when classifying the negative outputs.