*Name: Ayush Chanchal*

*Sap id: 500097569*

*Batch: B6*

*EXPERIMENT: 10*

Q. Explain RBF (Radial Basis Function)

: = A radial basis function (RBF) neural network is a type of artificial neural network that uses radial basis functions as activation functions. It typically consists of three layers: an input layer, a hidden layer, and an output layer. The hidden layer applies a radial basis function, usually a Gaussian function, to the input. The output layer then linearly combines these outputs to generate the final output. RBF neural networks are highly versatile and are extensively used in pattern classification tasks, function approximation, and a variety of machine learning applications. They are especially known for their ability to handle non-linear problems effectively.

**Structure of RBF neural networks**

An RBF neural network typically comprises three layers:

• Input layer: This layer simply transmits the inputs to the neurons in the hidden layer.

• Hidden layer: Each neuron in this layer applies a radial basis function to the inputs it receives.

• Output layer: Each neuron in this layer computes a weighted sum of the outputs from the hidden layer, resulting in the final output.

Source Code:

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.datasets import make\_classification

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import accuracy\_score

import scipy.spatial.distance as distance

class RadialBasisFunctionNeuralNetwork:

def \_\_init\_\_(self, num\_of\_rbf\_units=10):

self.num\_of\_rbf\_units = num\_of\_rbf\_units

def \_rbf\_unit(self, rbf\_center, point\_in\_dataset):

return np.exp(-self.beta \* distance.cdist([point\_in\_dataset], [rbf\_center],'euclidean')\*\*2).flatten()[0]

def \_construct\_interpolation\_matrix(self, input\_dataset):

interpolation\_matrix = np.zeros((len(input\_dataset), self.num\_of\_rbf\_units))

for idx, point\_in\_dataset in enumerate(input\_dataset):

for center\_idx, rbf\_center in enumerate(self.rbf\_centers):

interpolation\_matrix[idx, center\_idx] = self.\_rbf\_unit(rbf\_center, point\_in\_dataset)

return interpolation\_matrix

def train\_model(self, input\_dataset, target\_dataset):

self.kmeans\_clustering = KMeans(n\_clusters=self.num\_of\_rbf\_units,

random\_state=0).fit(input\_dataset)

self.rbf\_centers = self.kmeans\_clustering.cluster\_centers\_

self.beta = 1.0 / (2.0 \* (self.kmeans\_clustering.inertia\_ / input\_dataset.shape[0]))

interpolation\_matrix = self.\_construct\_interpolation\_matrix(input\_dataset)

self.model\_weights =np.linalg.pinv(interpolation\_matrix.T.dot(interpolation\_matrix)).dot(interpolation\_matrix.T).dot(target\_dataset)

def predict(self, input\_dataset):

interpolation\_matrix = self.\_construct\_interpolation\_matrix(input\_dataset)

predicted\_values = interpolation\_matrix.dot(self.model\_weights)

return predicted\_values

if \_\_name\_\_ == "\_\_main\_\_":

# Generating a simple classification dataset

input\_dataset, target\_dataset = make\_classification(n\_samples=500, n\_features=2,n\_informative=2, n\_redundant=0, n\_classes=2)

# Initializing and training the RBF neural network

rbf\_neural\_network = RadialBasisFunctionNeuralNetwork(num\_of\_rbf\_units=20)

rbf\_neural\_network.train\_model(input\_dataset, target\_dataset)

# Predicting the target values

predictions = rbf\_neural\_network.predict(input\_dataset)

# Converting continuous output to binary labels

binary\_predictions = np.where(predictions > 0.5, 1, 0)

# print("Accuracy: {}".format(accuracy\_score(target\_dataset, binary\_predictions)))

print(f"Accuracy: {accuracy\_score(target\_dataset, binary\_predictions)\*100}%")

# Plotting the results

plt.scatter(input\_dataset[:, 0], input\_dataset[:, 1], c=binary\_predictions, cmap='viridis',alpha=0.7)

plt.scatter(rbf\_neural\_network.rbf\_centers[:, 0], rbf\_neural\_network.rbf\_centers[:, 1], c='red')

plt.title('Classification Result')

plt.show()

Screenshot:

