**Task 11**

**Assuming a set of documents that need to be classified, use the Radial basis function Classifier model to perform this task. Calculate the classification rate, accuracy, precision, and recall for your data set.**

**Tools: Google co-lab, Python, Scikitlearn, Anaconda navigator**

**Aim:**

To our in-depth exploration of Radial Basis Function Networks (RBFNs) using Python 3, looking to understand the basics or an experienced coder aiming to harness the power of RBFNs.

Algorithms:

**What Are Radial Basis Function Networks (RBFNs)?**

* Radial Basis Function Networks, or RBFNs, are a type of artificial neural network that is primarily used for function approximation, classification, and clustering tasks. These networks are particularly well-suited for solving problems where data is nonlinear or non-continuous.
* The core idea behind RBFNs is to use radial basis functions as activation functions. These functions are centered at specific points in the input space and have the ability to transform input data into a higher-dimensional space, making it easier to separate and classify.

**RBFNs in Action**

* To better understand RBFNs, let’s explore a practical example. Imagine you’re working on a project to classify different types of flowers based on their petal length and width. You have a dataset with various flower samples, and you want to create a model that can accurately classify these flowers.
* Here’s where RBFNs come into play. With their ability to transform the input data into a higher-dimensional space, RBFNs can be an excellent choice for this classification task. By selecting appropriate radial basis functions and training the network, you can build a model that can effectively distinguish between different flower types.

**Setting Up Your Environment**

Before we dive into the code, it’s essential to set up your Python environment. We recommend using Jupyter Notebook for its interactive capabilities, which are great for learning and experimenting.

First, make sure you have Python 3 installed on your system. You can download and install it from the official Python website.

Next, install Jupyter Notebook using pip:

pip install jupyter numpy scipy scikit-learn matplotlib

Once you’ve installed Jupyter Notebook, you can launch it from your terminal by running:

jupyter notebook

Now, let’s begin our exploration of RBFNs with Python 3.

**Importing Libraries**

In Python, we have a wealth of libraries and tools at our disposal. For this tutorial, we’ll be using the following libraries:

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_classification

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import StandardScaler

from scipy.spatial.distance import cdist

* numpy: For numerical operations and array handling.
* matplotlib: To create insightful plots.
* scikit-learn: To generate a sample dataset, perform train-test splits, and evaluate our RBFN model.
* StandardScaler: To standardize our dataset.
* scipy.spatial.distance.cdist: For calculating pairwise distances between data points.

**Generating a Sample Dataset**

Let’s create a sample dataset for our flower classification example. We’ll use scikit-learn to generate a synthetic dataset with two features and two classes. This dataset will serve as our training and testing data.

X, y = make\_classification(n\_samples=300, n\_features=2, n\_classes=2, n\_clusters\_per\_class=2, n\_redundant=0, random\_state=42)

In this code snippet, we generate a dataset with 300 samples, 2 features, 2 classes, and 2 clusters per class. The random\_state parameter ensures reproducibility.

Now, let’s visualize the dataset to get a better understanding of our data distribution:

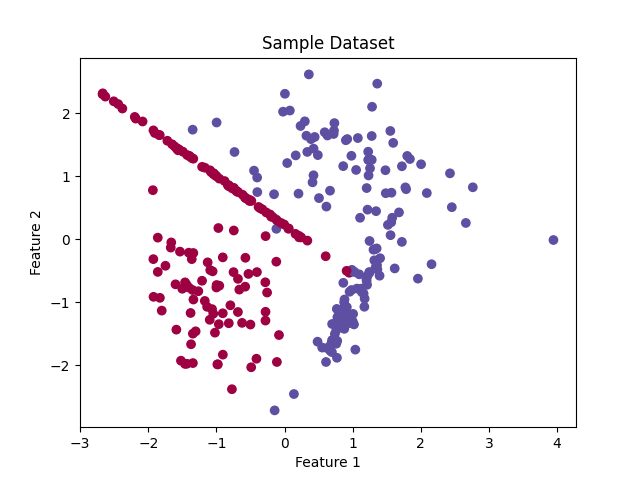
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral)

plt.title("Sample Dataset")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()



The resulting plot will show how our dataset is distributed, with different colors representing different classes.

**Preprocessing the Data**

Before we dive into building our RBFN, it’s crucial to preprocess the data. Standardization, or feature scaling, ensures that all features have the same scale, which is essential for the proper functioning of RBFNs.

scaler = StandardScaler()

X = scaler.fit\_transform(X)

By using the StandardScaler from scikit-learn, we standardize our data, which makes it easier for our RBFN to learn and converge efficiently.

**Defining Radial Basis Functions**

Now, we come to the heart of our RBFN implementation: the radial basis functions. These functions are responsible for transforming our input data.

In this example, we’ll use Gaussian radial basis functions. The Gaussian RBF is defined as:

[ \phi(x) = \exp \left( -\frac{|x – \mu|^2}{2\sigma^2} \right) ]

Where:

* (x) is the input data point.
* (\mu) is the center of the RBF.
* (\sigma) is the width of the RBF.

Let’s define the Gaussian RBF in Python:

def gaussian\_rbf(x, center, sigma):

return np.exp(-cdist(x, center, 'sqeuclidean') / (2 \* sigma\*\*2))

The gaussian\_rbf function takes input data (x), center points, and the width (\sigma). It calculates the Gaussian RBF for each data point.

**Choosing RBF Centers and Width**

Selecting the appropriate RBF centers and width is crucial for the success of your RBFN model. The centers should be chosen carefully based on your data distribution, and the width should be set to control the influence of each RBF.

Let’s randomly select the centers for our RBFs and set a default width:

n\_centers = 10 # Number of RBF centers

center\_indices = np.random.choice(X.shape[0], n\_centers, replace=False)

rbf\_centers = X[center\_indices]

rbf\_width = 1.0

In this example, we choose 10 random data points as our RBF centers. The width is set to 1.0 initially, but you can experiment with different values to see how they affect your model.

**Building the RBFN**

With our data preprocessed and our RBFs defined, it’s time to build our Radial Basis Function Network. We’ll create a simple two-layer network:

1. The first layer consists of the Gaussian RBFs.
2. The second layer is a linear layer, which combines the outputs of the RBFs.

Let’s put it all together:

def rbf\_layer(X, rbf\_centers, rbf\_width):

return gaussian\_rbf(X, rbf\_centers, rbf\_width)

def rbfn\_predict(X, rbf\_centers, rbf\_width, weights):

rbf\_outputs = rbf\_layer(X, rbf\_centers, rbf\_width)

return rbf

\_outputs @ weights

In the rbf\_layer function, we compute the outputs of our Gaussian RBFs. The rbfn\_predict function takes these RBF outputs and weights to predict the final outcome.

**Training the RBFN**

Training an RBFN involves finding the optimal weights that minimize the error between the predicted values and the actual target values. In our example, we’ll use scikit-learn’s train\_test\_split to create training and testing sets, and then use linear regression to find the optimal weights.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

rbf\_outputs\_train = rbf\_layer(X\_train, rbf\_centers, rbf\_width)

# Perform linear regression

from sklearn.linear\_model import LinearRegression

lr = LinearRegression()

lr.fit(rbf\_outputs\_train, y\_train)

# Make predictions on the test set

rbf\_outputs\_test = rbf\_layer(X\_test, rbf\_centers, rbf\_width)

y\_pred = lr.predict(rbf\_outputs\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, (y\_pred >= 0.5).astype(int))

print(f"Accuracy: {accuracy \* 100:.2f}%")

**Accuracy: 96.67%**

In this code, we split our data into training and testing sets, compute RBF outputs for the training and testing data, and then perform linear regression to find the optimal weights. Finally, we evaluate the model’s accuracy on the test set.

**Visualizing the Results**

To get a better grasp of how well our RBFN performs, let’s visualize the decision boundary it has learned:

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.01), np.arange(y\_min, y\_max, 0.01))

Z = rbfn\_predict(np.c\_[xx.ravel(), yy.ravel()], rbf\_centers, rbf\_width, lr.coef\_)

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.8)

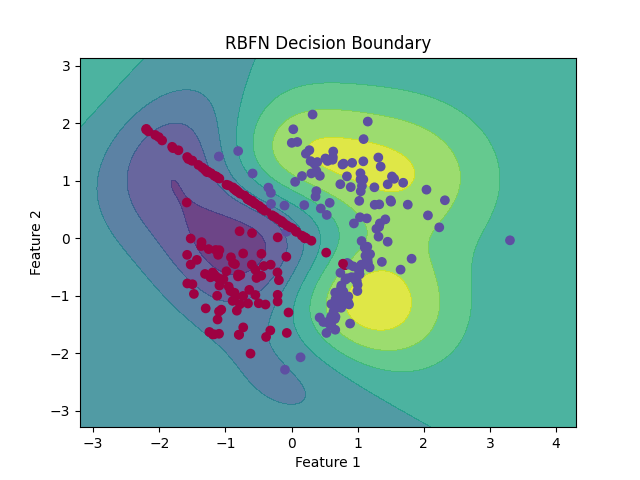
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral)

plt.title("RBFN Decision Boundary")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()



This code will create a contour plot that illustrates the decision boundary learned by our RBFN. It should help you visualize how well the network classifies the data points.

**Result**

Thus the Radial Basis Function Networks (RBFNs) using Python 3. In this programme covered the fundamentals, created a sample dataset, preprocessed the data, defined and implemented RBFs, built and trained an RBFN, and visualized the results.