**CS6301 MACHINE LEARNING LAB WEEK – 5 RBF**

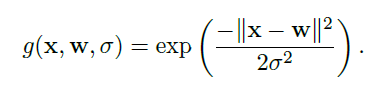
**SRIHARI. S – 2018103601**

**Date**: 22-03-2021 Monday

**Aim**: To implement Radial Basis Function Networks with different datasets and measure the performance metrics.

**ALGORITHM:**

* Position the RBF centres by either:
  + using the *k*-means algorithm to initialise the positions of the RBF centres OR
  + setting the RBF centres to be randomly chosen datapoints
* Calculate the actions of the RBF nodes using Equation
* **Gaussian function:**

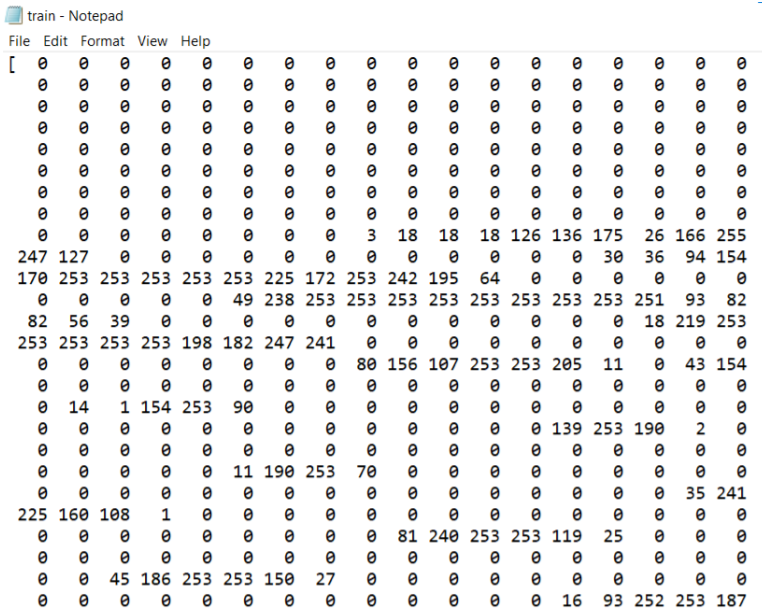


* Train the output weights by either:
* using the Perceptron OR
* computing the pseudo-inverse of the activations of the RBF centres

**Dataset-1:** MNIST Dataset

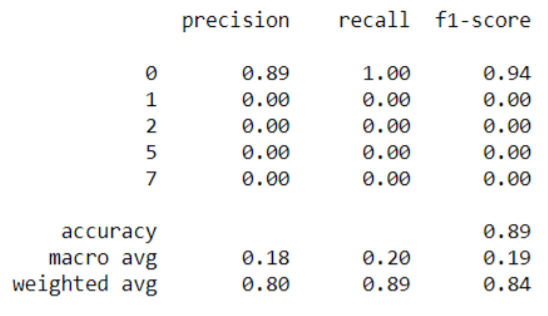
The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems which contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset.

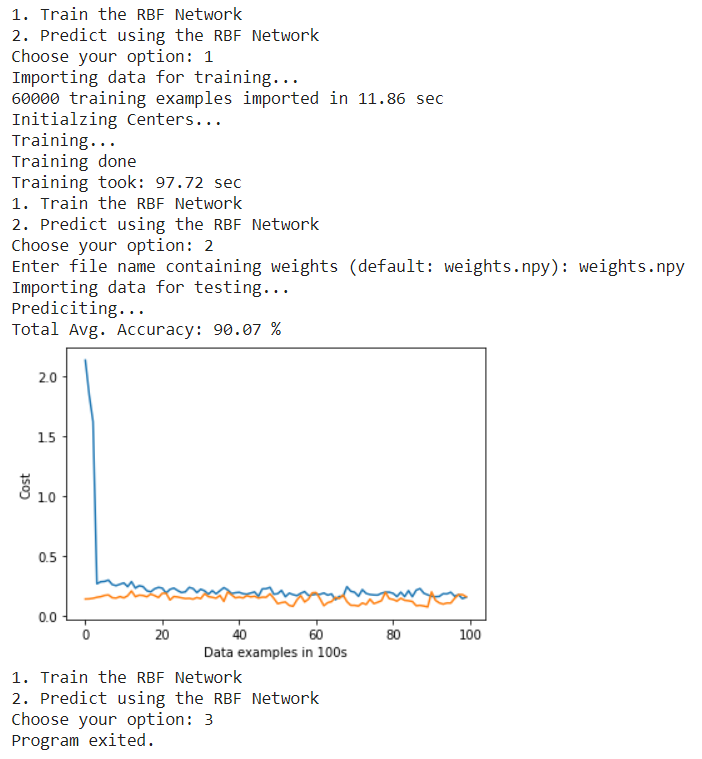
**Input:**



**Output:**

**Precision and Recall**





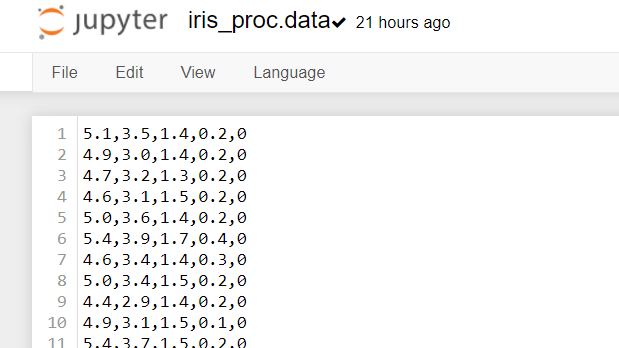
**Dataset**: **Iris Dataset**

**Url**: <https://archive.ics.uci.edu/ml/datasets/iris>

**Description**: The **Iris Dataset** contains four features (length and width of sepals and petals) of 50 samples of three species of **Iris** (**Iris** setosa, **Iris** virginica and **Iris** versicolor).

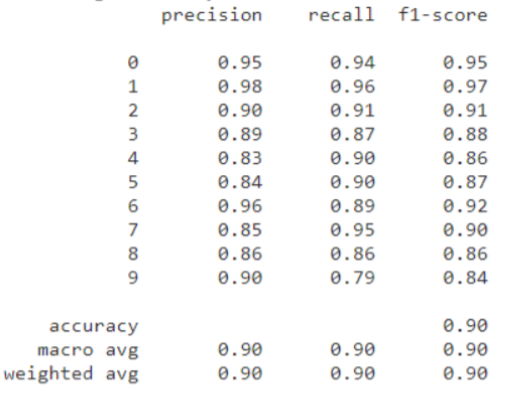
**Input**: The following 4 attributes

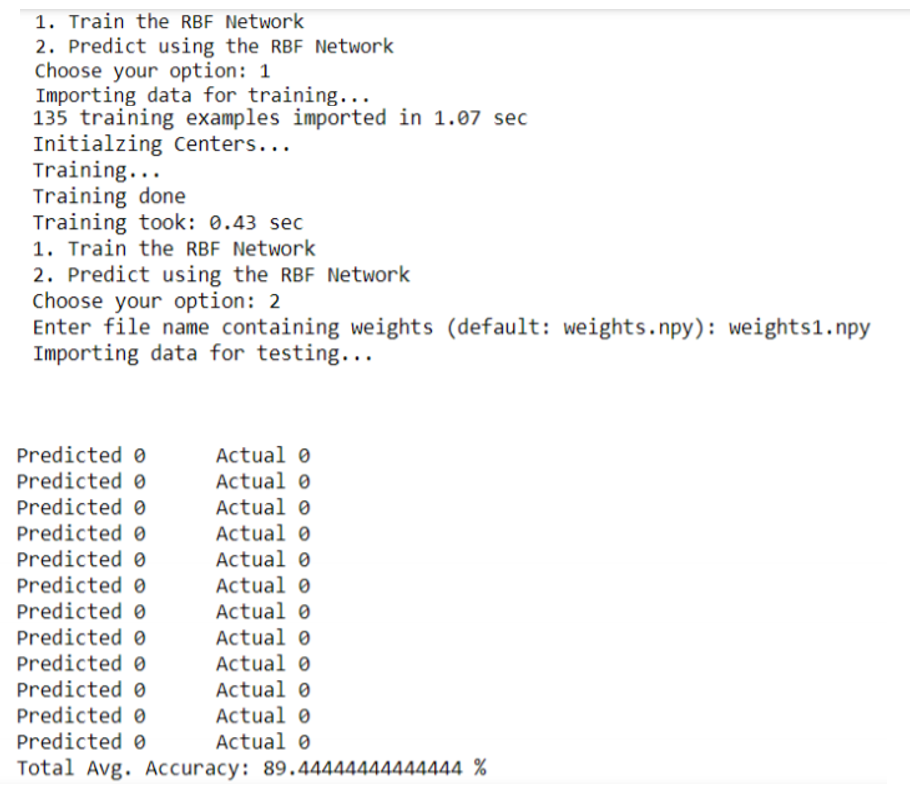
* sepal length in cm,
* sepal width in cm,
* petal length in cm,
* petal width in cm,

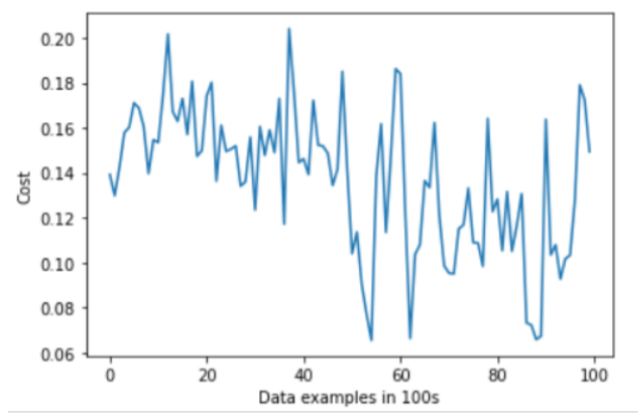


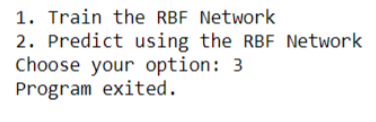
**Output**: decision: Multiclass classification among 3 classes of flowers: Iris Setosa, Iris Versicolour, Iris Virginica.

**Precision and Recall**









**TABULAR INFERENCE:**

|  |  |  |
| --- | --- | --- |
| **DATASET** | **MNIST** | **IRIS** |
| **PRECISION** | 89 | 90 |
| **RECALL** | 90 | 79 |
| **F1-SCORE** | 94 | 84 |
| **ACCURACY** | 90.07 % | 89.44 % |

The Radial basis function neural networks are demonstrated on different datasets and their performance metrics are measured. The RBF approach requires less time for model development since no repetition is required to reach the optimum model parameters.

**Code:**

import numpy as np

import time

import re

import sys

import math

import matplotlib.pyplot as plt

np.set\_printoptions(threshold=sys.maxsize, suppress=True)

np.random.seed(1)

def inputXFromFile(filename, sampleSize):

inputArray = np.zeros(shape=(sampleSize, 784))

with open(filename, "r") as file:

for i in range(sampleSize):

inputList = []

for \_ in range(44): # 44 lines of each example in file

line = file.readline().strip("[").replace("]", "")

inputList += line.split()

inputArray[i] = inputList

return np.divide(inputArray, 255)

def inputYFromFile(filename, sampleSize):

inputArray = np.zeros(shape=(sampleSize, 10))

with open(filename, "r") as file:

for i in range(sampleSize):

value = file.readline()

if not value:

break

inputArray[i][int(value)] = 1

return inputArray

def kMeansClustering(K, sampleData):

randIndices = np.random.choice(sampleData.shape[0], size=K, replace=False)

centeriods = sampleData[randIndices, :]

dataSize = 10000

data = sampleData[:dataSize]

for i in range(15):

centeriodSums = np.zeros(shape=centeriods.shape)

centeriodSumsCounter = np.zeros(shape=K)

for x in data:

index = np.argmin(np.square(centeriods - x).sum(axis=1))

centeriodSums[index] += x

centeriodSumsCounter[index] += 1

for i in range(K):

centeriods[i] = centeriodSums[i] / centeriodSumsCounter[i]

return centeriods

class Network:

def \_\_init\_\_(self):

self.XSize = 0

self.HSize = 300

self.OSize = 10

self.X = []

self.C = []

self.Y = []

self.W = []

self.trainErrors = []

self.testErrors = []

def loadData(self, filenameX, filenameY, sampleSize):

self.X = inputXFromFile(filenameX, sampleSize)

self.Y = inputYFromFile(filenameY, sampleSize)

self.XSize = sampleSize

def initializeCenters(self, K, useKMeans):

print("Initializing Centers...")

self.HSize = K # Since centriods is equal to hidden layer neurons

if useKMeans:

self.C = kMeansClustering(K, self.X)

else:

self.C = self.X[: self.HSize]

def train(self, epochs=1, learnRate=0.5, K=300, useKMeans=False):

self.initializeCenters(K, useKMeans)

self.W = np.random.uniform(-1, 1, (self.HSize, self.OSize))

self.trainErrors = np.zeros(shape=self.XSize) # Preallocating numpy array

print("Training...")

for \_ in range(epochs):

for i, x in enumerate(self.X):

HLayer = rbf(x, self.C)

output = np.dot(HLayer, self.W) # + self.B

error = output - self.Y[i]

self.W = self.W - (learnRate \* np.outer(HLayer, error))

self.trainErrors[i] = 0.5 \* sum(error \*\* 2)

print("Training done")

np.save("weights", self.W)

np.save("centers", self.C)

def predict(self):

self.testErrors = np.zeros(shape=self.XSize) # Preallocating numpy array

print("Predicting...")

totalAvg = count = correctCount = 0.0

for count, x in enumerate(self.X):

HLayer = rbf(x, self.C)

output = np.dot(HLayer, self.W) # + self.B

o = np.argmax(output)

y = np.argmax(self.Y[count])

if o == y:

correctCount += 1

error = output - self.Y[count]

self.testErrors[count] = 0.5 \* sum(error \*\* 2)

totalAvg = (correctCount \* 100.0) / (count + 1)

print(f"Total Avg. Accuracy: {totalAvg} %")

def rbf(x, C, beta=0.05):

H = np.zeros(shape=(np.shape(C)[0]))

for i, c in enumerate(C): # For each neuron in H layer

H[i] = math.exp((-1 \* beta) \* np.dot(x - c, x - c))

return H

def plotLearningCurves(trainErrors, testErrors):

avgSize = 100

if type(trainErrors) is np.ndarray:

Jtrain = trainErrors.reshape(-1, avgSize).mean(axis=1)

plt.plot(Jtrain, label='Training Cost')

Jtest = testErrors.reshape(-1, avgSize).mean(axis=1)

plt.plot(Jtest, label='Test Cost')

plt.xlabel(f"Data examples in {avgSize}s")

plt.ylabel("Cost")

plt.show()

start = time.time()

trainDataSize = 60000

testDataSize = 10000

myNetwork = Network()

while True:

print("1. Train the RBF Network\n2. Predict using the RBF Network")

userInput = input("Choose your option: ")

if userInput == "1":

print("Importing data for training...")

startTime = time.time()

myNetwork.loadData("train.txt", "train-labels.txt", trainDataSize)

print(

f"{trainDataSize} training examples imported in {time.time()-startTime:.2f} sec"

)

startTrainingTime = time.time()

myNetwork.train(epochs=1, learnRate=0.3, K=100, useKMeans=True)

print(f"Training took: {time.time()-startTrainingTime:.2f} sec")

elif userInput == "2":

filename = input("Enter file name containing weights (default: weights.npy): ")

myNetwork.W = np.load(filename)

myNetwork.C = np.load("centers.npy")

print("Importing data for testing...")

myNetwork.loadData("test.txt", "test-labels.txt", testDataSize)

myNetwork.predict()

plotLearningCurves(myNetwork.trainErrors[:10000], myNetwork.testErrors[:10000])

else:

break

print("Program exited.")