CSE6363 MACHINE LEARNING

**ASSIGNMENT-1**

MADHULIKA REDDY LEKKALA

1001718053

The given Problem statement was executed, and the final result is recorded for all various types of inputs and the accuracies:

**RANDOM SPLIT:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **RANDOM** | **EUCLIDIAN** | **POLYNOMIAL** | **RADIAL** | **SIGMOID** |
| **IRIS.DATA** | **97.7960%** | **97.7960%** | **97.8160%** | **97.8160%** |
| **YEAST.DATA** | **48.769%** | **49.30%** | **50.099%** | **48.90%** |
| **WDBC.DATA** | **92.146%** | **92.146%** | **37.69%** | **91.099%** |

**K-FOLD:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **K-FOLD** | **EUCLIDIAN** | **POLYNOMIAL** | **RADIAL** | **SIGMOD** |
| **IRIS** | **96.33** | **97.33** | **94.666** | **94.73** |
| **YEAST** | **52.9729** | **53.10** | **54.1891** | **53.108** |
| **WDBC** | **91.071** | **91.07%** | **62.32%** | **74.285%** |

**CODE:**

import csv

import random

import math

import operator

import numpy as np

from random import seed

from random import randrange

#Loading the dataset

def LoadingData(file, split,method, trainingSet=[] ,testSet=[],dataset1=[]):

with open(file) as infile, open('csv.data', 'w') as outfile:

for line in infile:

outfile.write(" ".join(line.split()).replace(' ', ','))

outfile.write("\n")

with open('csv.data', 'rt') as csvfile:

lines = csv.reader(csvfile)

dataset = list(lines)

if (file=='wdbc.data'):

for z in dataset:

z[1], z[31] = z[31], z[1]

for x in range(len(dataset)-1):

if (file=='iris.data'):

for y in range(4):

m=y

dataset[x][m] = float(dataset[x][m])

if (file=='yeast.data'):

for y in range(8):

m=y

dataset[x][m+1] = float(dataset[x][m+1])

if (file=='wdbc.data'):

for y in range(30):

m=y

dataset[x][m+1] = float(dataset[x][m+1])

if(method=="randomSplit"):

if random.random() < split:

trainingSet.append(dataset[x])

else:

testSet.append(dataset[x])

if(method=="kfold"):

dataset1.append(dataset[x])

#defining the k-fold cross validation

def cross\_validation\_split(dataset, folds=3):

dataset\_split = list()

dataset\_copy = list(dataset)

fold\_size = int(len(dataset) / folds)

for i in range(folds):

fold = list()

while len(fold) < fold\_size:

index = randrange(len(dataset\_copy))

fold.append(dataset\_copy.pop(index))

dataset\_split.append(fold)

return dataset\_split

#defining the Euclidean distance

def euclideanDistance(data,instance1, instance2, length):

distance = 0

if (data=='iris.data'):

for x in range(length):

distance += pow((instance1[x] - instance2[x]), 2)

if (data=='yeast.data' or data=='wdbc.data'):

for x in range(length-1):

y=x

distance += pow((instance1[y+1] - instance2[y+1]), 2)

return math.sqrt(distance)

#defining the polynomial distance

def polynomialKernel(data,instance1, instance2, length):

distance = 0

xx=0.0

yy=0.0

xy=0.0

if (data=='iris.data'):

for x in range(length):

xy = xy + instance1[x]\*instance2[x]

yy = yy + instance2[x]\*instance2[x]

xx = xx + instance1[x]\*instance1[x]

if (data=='yeast.data' or data=='wdbc.data'):

for x in range(length-1):

y=x

xy = xy + instance1[y+1]\*instance2[y+1]

yy = yy + instance2[y+1]\*instance2[y+1]

xx = xx + instance1[y+1]\*instance1[y+1]

distance=pow(1 +math.sqrt(xx -2\*xy + yy),3);

return distance

#defining the radial distance

def radialDistance(data,instance1, instance2, length):

distance = 0

xMy=0.0

sigma=0.97

if (data=='iris.data'):

for x in range(length):

xMy = xMy + abs(instance1[x] - instance2[x])

if (data=='yeast.data' or data=='wdbc.data'):

for x in range(length-1):

y=x

xMy = xMy + abs(instance1[y+1] - instance2[y+1])

distance=2-2\*math.exp(-(math.pow(xMy,2)/math.pow(sigma,2)))

return distance

#defining the sigmoid distance

def sigmoidDistance(data,instance1, instance2, length):

distance = 0

xx=0.0

yy=0.0

xy=0.0

if (data=='iris.data'):

for x in range(length):

xy = xy + instance1[x]\*instance2[x]

yy = yy + instance2[x]\*instance2[x]

xx = xx + instance1[x]\*instance1[x]

if (data=='yeast.data' or data=='wdbc.data'):

for x in range(length-1):

y=x

xy = xy + instance1[y+1]\*instance2[y+1]

yy = yy + instance2[y+1]\*instance2[y+1]

xx = xx + instance1[y+1]\*instance1[y+1]

distance= np.tanh(0.3\*math.sqrt(xx -2\*xy + yy)+0.7)

return distance

def getNeighbors(distance\_method,data,trainingSet, testInstance, k):

distances=[]

length = len(testInstance)-1

for x in range(len(trainingSet)):

if(distance\_method=="euclidean"):

dist = euclideanDistance(data,testInstance, trainingSet[x], length)

if(distance\_method=="polynomial"):

dist = polynomialKernel(data,testInstance, trainingSet[x], length)

if(distance\_method=="radialDistance"):

dist = radialDistance(data,testInstance, trainingSet[x], length)

if(distance\_method=="sigmoidDistance"):

dist = sigmoidDistance(data,testInstance, trainingSet[x], length)

distances.append((trainingSet[x], dist))

distances.sort(key=operator.itemgetter(1))

neighbors = []

for x in range(k):

neighbors.append(distances[x][0])

return neighbors

def getResponse(neighbors):

classVotes = {}

for x in range(len(neighbors)):

response = neighbors[x][-1]

if response in classVotes:

classVotes[response] += 1

else:

classVotes[response] = 1

sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)

return sortedVotes[0][0]

def getAccuracy(testSet, predictions):

correct = 0

for x in range(len(testSet)):

if testSet[x][-1] == predictions[x]:

correct += 1

return (correct/float(len(testSet))) \* 100.0

def main():

# prepare data

trainingSet=[]

testSet=[]

dataset1=[]

avg\_accuracy=0

seed(1)

method = input("Method")

distance\_method=input("Distance Method")

data = input("Enter the dataset name: ")

k = input("Enter the value of k : ")

split = 0.67

if(method=="randomSplit"):

LoadingData(data, split,method,trainingSet, testSet,dataset1)

print('Train set: ' + repr(len(trainingSet)))

print('Test set: ' + repr(len(testSet)))

predictions=[]

k=int(k)

for x in range(len(testSet)):

neighbors = getNeighbors(distance\_method,data,trainingSet, testSet[x], k)

result = getResponse(neighbors)

predictions.append(result)

print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1]))

accuracy = getAccuracy(testSet, predictions)

print('Accuracy: ' + repr(accuracy) + '%')

if(method=="kfold"):

LoadingData(data, split,method,trainingSet, testSet,dataset1)

dataset\_split=cross\_validation\_split(dataset1,10)

for x in range(10):

for y in range(int(len(dataset1)/10)):

testSet.append(dataset\_split[x][y])

for z in range(10):

if(z!=x):

for m in range(int(len(dataset1)/10)):

trainingSet.append(dataset\_split[z][m])

predictions=[]

k=int(k)

for x in range(len(testSet)):

neighbors = getNeighbors(distance\_method,data,trainingSet, testSet[x], k)

result = getResponse(neighbors)

predictions.append(result)

print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1]))

accuracy = getAccuracy(testSet, predictions)

avg\_accuracy= avg\_accuracy+accuracy

print('Accuracy: ' + repr(accuracy) + '%')

trainingSet=[]

testSet=[]

print('Avg\_Accuracy'+repr(avg\_accuracy/10)+'%')

main()

**REFERENCES:**

[**https://www.researchgate.net/profile/Kai\_Yu7/publication/220578072\_Kernel\_Nearest\_Neighbor\_Algorithm/links/02e7e533620f1ac895000000/Kernel-Nearest-Neighbor-Algorithm.pdf**](https://www.researchgate.net/profile/Kai_Yu7/publication/220578072_Kernel_Nearest_Neighbor_Algorithm/links/02e7e533620f1ac895000000/Kernel-Nearest-Neighbor-Algorithm.pdf)

[**https://machinelearningmastery.com/implement-resampling-methods-scratch-python/**](https://machinelearningmastery.com/implement-resampling-methods-scratch-python/)

**https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python**