657 Assignment 2 Q3

July 3, 2021

[2]:
```python
# Importing libraries as
import numpy as np
import pandas as pd
import random
import math
from numpy import linalg as lin
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
```

[6]:
```python
# Generating the data set

#Fix the randomizer to get same values each time
random.seed(50)

data = []

while len(data) < 441:
    i = random.randint(0, 20)
    j = random.randint(0, 20)
    x1 = round(((0.2*i) - 2), 2)
    x2 = round(((0.2*j) - 2), 2)
    if [x1, x2] not in data:
        data.append([x1,x2])

inputs = pd.DataFrame(data)
inputs.describe()
```

[6]:

	0	1
count	4.410000e+02	4.410000e+02
mean	2.718914e-17	-2.895139e-18
std	1.212436e+00	1.212436e+00
min	-2.000000e+00	-2.000000e+00
25%	-1.000000e+00	-1.000000e+00
50%	0.000000e+00	0.000000e+00
75%	1.000000e+00	1.000000e+00
max	2.000000e+00	2.000000e+00

```
[7]: # Generate data lables for the inputs
# wehre F(x1, x2) = +1 if x1^2 + x2^2 <= 1
#                 = -1 if x1^2 + x2^2 > 1
# Using the above relation find all the data labels for 441 input samples

out = []
x1_column = 0
x2_column = 1

for i in range(len(inputs)):
    if (((inputs.loc[i, x1_column])**2 + (inputs.loc[i, x2_column])**2) > 1):
            out.append(-1)
    else:
        out.append(1)

outputs = pd.DataFrame(out)

# Generate a single data-set with inputs and outputs
dataset = pd.concat([inputs,outputs], axis = 1)
```

```
[12]: # Split the dataset into 80-20 Train-test split

train_data, test_data, train_label, test_label = train_test_split(dataset.iloc[:
 ↪,:2], dataset.iloc[:,2:3], test_size=0.2,                                     ␣
 ↪random_state=50)
# Convert the splitted dataset into array for better computation
train_data = np.asarray(train_data)
test_data = np.asarray(test_data)
train_label = np.asarray(train_label)
test_label = np.asarray(test_label)

print("Shape of Training data is:   {} ".format(train_data.shape))
print("Shape of Training labels is: {} \n".format(train_label.shape))

print("Shape of Test data is: {} ".format(test_data.shape))
print("Shape of Test data is: {}".format(test_label.shape))
```

```
Shape of Training data is:   (352, 2)
Shape of Training labels is: (352, 1)

Shape of Test data is: (89, 2)
Shape of Test data is: (89, 1)
```

```
[39]: ### get_Centers ###

# Calculate the centers for various input choices
```

```
# Choice are as follows:
#    choice = 0 : All the training data points are considered as centers
#    choice = 1 : 150 random train data points are conisdered as centers
#    choice = 2 : Using k-means clustering 150 cluster_cneters are considered␣
↪as centers
# Returns centers for RBF comppputation

def get_Centers(choice, inputs, hidden_nodes):
    # apply cases as per questions
    centers = []
    for i in range(hidden_nodes):
        if choice == 0:
            centers.append(inputs[i])

        if choice == 1:
            for i in range(hidden_nodes):
                #print("randomizer:", i)
                rand_indx = random.randint(0, inputs.shape[0]-1)
                #print("random value is: {} for index no {}".format(rand_indx,␣
↪i))
                centers.append(inputs[rand_indx])

        if choice == 2:
            k_means = KMeans(n_clusters = hidden_nodes, random_state = 50)
            k_means.fit(inputs)
            centers = k_means.cluster_centers_

    return centers



### train_data_Gaussian ###
# Compute the G matrix for training dataset
# Returns G matrix for training data

def train_data_Gaussian(centers, train_data, sigma, hidden_nodes):
    # length of training data
    in_length = train_data.shape[0]
    #centers = np.asarray(centers)

    #initialize G matrices
    g = np.empty((in_length, hidden_nodes), dtype = float)

    #Computing the G function
    for i in range(in_length):
        for j in range(hidden_nodes):
            num = lin.norm(train_data[i] - centers[j])
```

```
            g[i][j] = math.exp(- math.pow(num,2) / (2*(math.pow(sigma,2))))
    return g



### get_weights ###
# Returns weights after coomputing Weights = G+.D

def get_Weights(gaussian_func, train_labels):

    # Compute G+ matrix where G+ = (GT G)-1 GT
    GT_G = np.dot(gaussian_func.T, gaussian_func)
    GT_G_inv = lin.pinv(GT_G)
    G_plus = np.dot(GT_G_inv, gaussian_func.T)

    # Compute Weight matrix as W = G+ dot D
    # G+ is psuedo inverse of Gaussian kernel function
    # D is train labels

    weights = np.dot(G_plus, train_labels)
    return weights


def get_RBF_output(centers, tst_data, sigma, hidden_nodes):
    # length of training data
    in_length = tst_data.shape[0]

    #initialize G matrices
    outs = np.empty((in_length, hidden_nodes), dtype = float)


    #Computing the G function
    for i in range(in_length):
        for j in range(hidden_nodes):
            numr = lin.norm(tst_data[i] - centers[j])
            outs[i][j] = math.exp(- math.pow(numr,2) / (2*(math.pow(sigma,2))))
    return outs


### get_accuracy ###
# Check if there is variation in predicted and actual data labels size(length)
# if not then comupute the average accuracy
# Returns accuracy for each data sample whihc is passed in the arguments

def get_accuracy(actual, predicted):
    accuracy = []
    if not (len(actual) == len(predicted)):
```

```
        print('Size of predicted and true labels not equal.')
        return 0.0

    corr = 0
    for i in range(0,len(actual)):
        corr += 1 if (actual[i] == predicted[i]).all() else 0
        accur = corr/len(actual)*100
    return accur



### get_MSE ###
# Mean Squared Error = (Predicted output - Actual output)^2 / Total number of␣
↪samples(i.e. taking the mean of error)
# Returns MSE for a given data sample which is passed in the arguments

def get_MSE(actual, predicted):
    MSE_RBF = []
    MSE = 0
    for i in range(len(predicted)):
        MSE = MSE + (predicted[i]-actual[i])**2
    Mean_Sq_err = MSE/len(predicted)
    MSE_RBF.append(Mean_Sq_err)
    return Mean_Sq_err
```

[40]:

```
### compute_RBF ###

# Steps:
# 1. get_Centers to identify the center points for RBF network
# 2. Compute the Gaussian kernel for the training dataset 80% of complete␣
↪dataset
# 3. compute weight matrix using dot product of G+ . D where D is data lables␣
↪for the sample inputs
# 4. predict train labels to check the accuracy and mse for training the␣
↪network
# 5. Compute gaussian kernel function for test data set 20% of the complete␣
↪dataset
# 6. predict test labels and check accuracy and mse of testing

# Returns predicted train and test labels

def compute_RBF(train_inputs, hidden_nodes, train_labels, sigma, test_data,␣
↪choice):
    centers = get_Centers(choice, train_inputs, hidden_nodes)
    G = train_data_Gaussian(centers, train_inputs, sigma, hidden_nodes)
    W = get_Weights(G, train_labels)
    predict_train_labels = np.dot(G, W)
    G_output = get_RBF_output(centers, test_data, sigma, hidden_nodes)
```

```
    predict_test_labels = np.dot(G_output, W)
    return predict_train_labels, predict_test_labels
```

[29]:
```
# Set spread parameters to iterate the training process of the RBF network and␣
↪compute the accuracy and mse for different spread values
spread_parameter = [0.001, 0.002, 0.01, 0.02, 0.03, 0.04,0.05, 0.1, 0.2, 0.3, 0.
↪4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

0.1 Part 1

0.1.1 RBF NN using all the points in the training set as centers of the RB functions and based on Gaussian kernel functions with constant spread function

[30]:
```
#Center choice = 0 := Compute RBF with centers as inputs i.e. number of␣
↪ceneters = number of inputs
Q3_part1 = 0

# Set the number of hidden nodes: as we are selecting all the training samples␣
↪as centers the size o hidden layer = lenght of train data
hidden_nodes = len(train_data)

# Initialize all the accuracy and mse values as 0
accur_train, accur_test, MSE_train, MSE_test = 0, 0, 0, 0

# Initialize the arrays for storing Train and Test data's Accuracy and MSE
accuracy_train = []
accuracy_test = []
mse_train =[]
mse_test = []


for sigma in spread_parameter:
    training_outcome, predict_out = compute_RBF( train_data, hidden_nodes,␣
↪train_label, sigma, test_data, Q3_part1)
    training_outcome = np.sign(training_outcome)
    predict_out = np.sign(predict_out)

    # Compute Accuracy
    accur_train = get_accuracy(train_label, training_outcome)
    accuracy_train.append(accur_train)
    accur_test = get_accuracy(test_label, predict_out)
    accuracy_test.append(accur_test)

    # Compute MSE
    MSE_train = get_MSE(train_label, training_outcome)
    mse_train.append(MSE_train)
    MSE_test = get_MSE(test_label, predict_out)
```

```
    mse_test.append(MSE_test)

    print("For spread parameter - {} \n 1. Train Accuracy is: {} and MSE = {}\n␣
↪2. Test Accuracy is: {} and MSE = {} \n\n".
          format(sigma, accur_train, MSE_train, accur_test, MSE_test))
```

```
For spread parameter - 0.001
 1. Train Accuracy is: 100.0 and MSE = [0.]
 2. Test Accuracy is: 0.0 and MSE = [1.]


For spread parameter - 0.002
 1. Train Accuracy is: 100.0 and MSE = [0.]
 2. Test Accuracy is: 0.0 and MSE = [1.]


For spread parameter - 0.01
 1. Train Accuracy is: 100.0 and MSE = [0.]
 2. Test Accuracy is: 95.50561797752809 and MSE = [0.17977528]


For spread parameter - 0.02
 1. Train Accuracy is: 100.0 and MSE = [0.]
 2. Test Accuracy is: 94.3820224719101 and MSE = [0.2247191]


For spread parameter - 0.03
 1. Train Accuracy is: 100.0 and MSE = [0.]
 2. Test Accuracy is: 91.01123595505618 and MSE = [0.35955056]


For spread parameter - 0.04
 1. Train Accuracy is: 100.0 and MSE = [0.]
 2. Test Accuracy is: 91.01123595505618 and MSE = [0.35955056]


For spread parameter - 0.05
 1. Train Accuracy is: 100.0 and MSE = [0.]
 2. Test Accuracy is: 92.13483146067416 and MSE = [0.31460674]


For spread parameter - 0.1
 1. Train Accuracy is: 100.0 and MSE = [0.]
 2. Test Accuracy is: 92.13483146067416 and MSE = [0.31460674]


For spread parameter - 0.2
 1. Train Accuracy is: 100.0 and MSE = [0.]
```