SML Assignment – 3 (Documentation)

First we import the necessary libraries that are numpy , tensorflow and random.

*#Question - 1*

import numpy as np

import tensorflow as tf

import random

Then we load the data set from the provided link to the google api as follows:

*#loading the data*

link = "https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz"

path = tf.keras.utils.get\_file('mnist.npz',link)

data = np.load(path)

x\_coord\_train , y\_coord\_train = data["x\_train"], data["y\_train"]

x\_coord\_test , y\_coord\_test = data["x\_test"], data["y\_test"]

Now , selecting the 3 classes 0 , 1 and 2 from the dataset as follows:

*#choosing the classes 0 , 1 , 2*

selected\_classes = [0, 1, 2] *# making the array for the classes we want*

select\_val\_train = np.isin(y\_coord\_train, selected\_classes) *# using the np.isin function to get the selected classes*

select\_val\_test = np.isin(y\_coord\_test, selected\_classes) *# using the np.isin function to get the selected classes*

x\_selected\_train = x\_coord\_train[select\_val\_train] *#selected x\_train data -- the value of n is close to 18000*

y\_selected\_train = y\_coord\_train[select\_val\_train] *#selected y\_train data -- the value of n is close to 18000*

x\_selected\_test = x\_coord\_test[select\_val\_test] *#selected x\_train data -- the value of n is close to 3000*

y\_selected\_test = y\_coord\_test[select\_val\_test] *#selected y\_train data -- the value of n is close to 3000*

This will have the datapoints which will be belonging to only one of this classes that is 0 , 1 or 2.

Now ,we are taking 10 dimensions from the dataset , both train and the test to get the desired dataset that is one having only the first 10 dimensions based on the eigenvalues of the matrix (using PCA) .

Now , we will use this reduced dimension dataset to get all the answers .

We do this as following :

p = 10

x\_selected\_train = x\_selected\_train.reshape(x\_selected\_train.shape[0], -1) *# making the selected train in a 2 dimensional array*

x\_selected\_test = x\_selected\_test.reshape(x\_selected\_test.shape[0], -1) *# making the selected test in a 2 dimensional array*

*# print(x\_selected\_train.shape)*

*# print(y\_selected\_train.shape)*

*# print(x\_selected\_test.shape)*

*# print(y\_selected\_test.shape)*

*# print(x\_selected\_train.shape)*

x\_selected\_train = x\_selected\_train.T

mean\_of\_X = np.mean(x\_selected\_train , *axis*=1 , *keepdims*=True)

*# print(mean\_of\_X.shape)*

X\_centralized = x\_selected\_train - mean\_of\_X

*# print(x\_selected\_train.shape[1] - 1)*

S = (X\_centralized @ X\_centralized.T) / (x\_selected\_train.shape[1] - 1)

S\_eigenvalues, S\_eigenvectors = np.linalg.eig(S)

sorted\_S = np.argsort(S\_eigenvalues)[::-1]

S\_eigenvalues = S\_eigenvalues[sorted\_S]

S\_eigenvectors = S\_eigenvectors[:, sorted\_S]

U = S\_eigenvectors

Up = U[:, :p]

Y = Up.T @ X\_centralized *# reducing the dimension of the dataset*

*#print(Y.shape)*

*# seeing all the 10 dimensions for the first split*

x\_selected\_test = x\_selected\_test.T

*#  mean\_of\_X = np.mean(x\_selected\_test , axis=1 , keepdims=True)*

*# print(mean\_of\_X.shape)*

X\_centralized = x\_selected\_test - mean\_of\_X

*# print(x\_selected\_test.shape[1] - 1)*

S = (X\_centralized @ X\_centralized.T) / (x\_selected\_test.shape[1] - 1)

S\_eigenvalues, S\_eigenvectors = np.linalg.eig(S)

sorted\_S = np.argsort(S\_eigenvalues)[::-1]

S\_eigenvalues = S\_eigenvalues[sorted\_S]

S\_eigenvectors = S\_eigenvectors[:, sorted\_S]

U = S\_eigenvectors

Up = U[:, :p]

Y\_test = Up.T @ X\_centralized *# reducing the dimension of the dataset*

Now , we need to learn the tree and make 3 terminal nodes such that we divide the dataset into 3 different regions and then predict the class of each datapoint in the dataset based on that.

For splitting, we are finding the mean of the dataset of that dimension (say i ) and then based on that we are splitting the datapoints on two bases that is either the value of a datapoint at that dimension will be less than the mean or will be greater than the mean value .

We do it using the function mean as follows:

*# Y[i][j] -- i is the dimension and j is the data point*

def mean(*i*,*Y*):

    sum = 0

    cnt = 0

    arr = [];

    for j in range(len(*Y*[*i*])):

        arr.append(*Y*[*i*][j])

        sum+=*Y*[*i*][j]

        cnt+=1

    arr.sort()

*# return (arr[0]+arr[-1])/2*

    return sum/cnt

We use the two\_partitions function to divide the datapoints into two sets , less than mean and more than mean .

def two\_partitions(*i*,*Y*):

    less\_than\_mean = []

    greater\_than\_mean = []

    mean\_i = mean(*i*,*Y*)

    for j in range(len(*Y*[*i*])):

        if *Y*[*i*][j] < mean\_i:

            less\_than\_mean.append(j)

        else:

            greater\_than\_mean.append(j)

    return less\_than\_mean , greater\_than\_mean

Finally , to get the split , we calculate the gini index of each split we get , if we do at the ith dimension and then take the lowest of them as follows :

We form a function that calculates the gini index for cut at that dimension:

def gini\_index(*two\_partitions*):

    left = [0]\*3

    right = [0]\*3

    less = *two\_partitions*[0]

    greater = *two\_partitions*[1]

    for i in less:

        left[y\_selected\_train[i]] += 1

    for i in greater:

        right[y\_selected\_train[i]] += 1

*# print(left , right )*

    sum\_left = sum(left)

    sum\_right = sum(right)

    gini\_left = 0

    gini\_right = 0

    for i in left :

        gini\_left += ((i/sum\_left)\*(1-(i/sum\_left)))

    for i in right:

        gini\_right += ((i/sum\_right)\*(1-(i/sum\_right)))

    return (gini\_left)\*(len(less)/(len(less)+len(greater))) + (gini\_right)\*(len(greater)/(len(less)+len(greater)))

Now, using these three functions to get the first cut as follows :

import random

def learn\_tree(*Y*, *Y\_test*):

*# getting where should we cut for the first split*

    min\_gini\_dim = 0

    for i in range(p):

*# print(gini\_index(two\_partitions(i,Y)))*

*# print(mean(i,Y))*

        if (gini\_index(two\_partitions(i,*Y*)) < gini\_index(two\_partitions(min\_gini\_dim,*Y*))):

            min\_gini\_dim = i

*# print("end=====================")*

    min\_gini\_mean = mean(min\_gini\_dim,*Y*)

This will give us the first cut correctly.

Now , moving on to the second cut , we randomly select any of the two parts that is either taking the one less than mean or the one more than mean .

Now , similarly we make the mean function for the datapoints having value at that dimension less than the mean and more than the mean.

def mean\_second\_cut\_lessmean(*i*,*gini\_min\_mean* , *gini\_min*,*Y*):

    sum = 0

    cnt = 0

    arr = []

    for j in range(len(*Y*[*i*])):

        if *Y*[*gini\_min*][j] < *gini\_min\_mean*:

            sum+=*Y*[*i*][j]

            arr.append(*Y*[*i*][j])

            cnt+=1

    arr.sort()

    return sum/cnt

*# return (arr[0]+arr[-1])/2*

def mean\_second\_cut\_moremean(*i*,*gini\_min\_mean* , *gini\_min*,*Y*):

    sum = 0

    cnt = 0

    arr = []

    for j in range(len(*Y*[*i*])):

        if *Y*[*gini\_min*][j] >= *gini\_min\_mean*:

            sum+=*Y*[*i*][j]

            cnt+=1

            arr.append(*Y*[*i*][j])

    arr.sort()

*# return (arr[0]+arr[-1])/2*

    return sum/cnt

Similarly , we need to make the two\_partitions function for them also as follows:

def two\_partitions\_second\_cut\_lessmean(*i* , *gini\_min\_mean* , *gini\_min*,*Y*):

    less\_than\_mean = []

    greater\_than\_mean = []

    mean\_i = mean\_second\_cut\_lessmean(*i* , *gini\_min\_mean* , *gini\_min*,*Y*)

    for j in range(len(*Y*[*i*])):

        if *Y*[*i*][j] < mean\_i and *Y*[*gini\_min*][j] < *gini\_min\_mean*:

            less\_than\_mean.append(j)

        elif *Y*[*i*][j] >= mean\_i and *Y*[*gini\_min*][j] < *gini\_min\_mean*:

            greater\_than\_mean.append(j)

    return less\_than\_mean , greater\_than\_mean

def two\_partitions\_second\_cut\_greatermean(*i* , *gini\_min\_mean* , *gini\_min*,*Y*):

    less\_than\_mean = []

    greater\_than\_mean = []

    mean\_i = mean\_second\_cut\_moremean(*i* , *gini\_min\_mean* , *gini\_min*,*Y*)

    for j in range(len(*Y*[*i*])):

        if *Y*[*i*][j] < mean\_i and *Y*[*gini\_min*][j] >= *gini\_min\_mean*:

            less\_than\_mean.append(j)

        elif *Y*[*i*][j] >= mean\_i and *Y*[*gini\_min*][j] >= *gini\_min\_mean*:

            greater\_than\_mean.append(j)

    return less\_than\_mean , greater\_than\_mean

The function gini\_index will remain the same for this operation too .

We similarly calculate the split for the less than mean and more than mean , and then randomly select one of them to give the second cut at another dimension .

min\_gini\_dim\_second\_cut\_lessmean = 0

    for i in range(p):

*# print(gini\_index(two\_partitions\_second\_cut\_lessmean(i,min\_gini\_mean,min\_gini\_dim,Y)))*

        if gini\_index(two\_partitions\_second\_cut\_lessmean(i,min\_gini\_mean,min\_gini\_dim,*Y*)) < gini\_index(two\_partitions\_second\_cut\_lessmean(min\_gini\_dim\_second\_cut\_lessmean,min\_gini\_mean,min\_gini\_dim,*Y*)) :

            min\_gini\_dim\_second\_cut\_lessmean = i

*# print("end=====================")*

*# print(min\_gini\_dim\_second\_cut\_lessmean)*

    min\_gini\_dim\_second\_cut\_greatermean = 0

    for i in range(p):

*# print(gini\_index(two\_partitions\_second\_cut\_greatermean(i,min\_gini\_mean,min\_gini\_dim,Y)))*

        if (gini\_index(two\_partitions\_second\_cut\_greatermean(i,min\_gini\_mean,min\_gini\_dim,*Y*)) < gini\_index(two\_partitions\_second\_cut\_greatermean(min\_gini\_dim\_second\_cut\_greatermean,min\_gini\_mean,min\_gini\_dim,*Y*))):

            min\_gini\_dim\_second\_cut\_greatermean = i

Randomly selecting now ,

x = random.randint(0,1)

    min\_gini\_dim\_second\_cut = 0 ; mean\_second\_cut = 0

*# if (gini\_index(two\_partitions\_second\_cut\_greatermean(min\_gini\_dim\_second\_cut\_greatermean,min\_gini\_mean,min\_gini\_dim,Y)) > gini\_index(two\_partitions\_second\_cut\_lessmean(min\_gini\_dim\_second\_cut\_lessmean,min\_gini\_mean,min\_gini\_dim,Y))):*

    if x == 1:

        min\_gini\_dim\_second\_cut = min\_gini\_dim\_second\_cut\_lessmean

        mean\_second\_cut = mean\_second\_cut\_lessmean(min\_gini\_dim\_second\_cut\_lessmean , min\_gini\_mean , min\_gini\_dim,*Y*)

    else :

        min\_gini\_dim\_second\_cut = min\_gini\_dim\_second\_cut\_greatermean

        mean\_second\_cut = mean\_second\_cut\_moremean(min\_gini\_dim\_second\_cut\_greatermean , min\_gini\_mean , min\_gini\_dim, *Y*)

    print("The first cut is at dimension:",min\_gini\_dim , "\nThe second cut is at dimension:" , min\_gini\_dim\_second\_cut)

Now, if we get the left one for the second cut or right one for the second cut , we make to cases and do the needful to detect the class in which the datapoint is predicted . We also calculate the accuracy based on the predicted class of the datapoint and the original class of the datapoint .

For class wise accuracy we store the class in which that datapoint is belonging as per the predicted class in a array and the original one similarly.

    if x == 0 :

        R1 = [] *# region 1*

        R2 = [] *# region 2*

        R3 = [] *# region 3*

        for j in range(len(*Y*[0])):

            if *Y*[min\_gini\_dim][j] < min\_gini\_mean :

                R1.append(j)

            elif *Y*[min\_gini\_dim][j] >= min\_gini\_mean and *Y*[min\_gini\_dim\_second\_cut][j] < mean\_second\_cut :

                R2.append(j)

            else :

                R3.append(j)

        R1\_class\_count = [0]\*3

        R2\_class\_count = [0]\*3

        R3\_class\_count = [0]\*3

        for i in R1:

            R1\_class\_count[y\_selected\_train[i]] += 1

        for i in R2:

            R2\_class\_count[y\_selected\_train[i]] += 1

        for i in R3:

            R3\_class\_count[y\_selected\_train[i]] += 1

*#print(np.argmax(R1\_class\_count) , np.argmax(R2\_class\_count) , np.argmax(R3\_class\_count))*

        class\_R1 = np.argmax(R1\_class\_count)

        class\_R2 = np.argmax(R2\_class\_count)

        class\_R3 = np.argmax(R3\_class\_count)

        print("Class of Region-1 is :",class\_R1, "\nClass of Region-2 is :",class\_R2, "\nClass of Region-3 is :",class\_R3)

*# if  Y\_test[min\_gini\_dim][0] < min\_gini\_mean :*

*#     print(class\_R1)*

*# elif Y\_test[min\_gini\_dim][0] >= min\_gini\_mean and Y\_test[min\_gini\_dim\_second\_cut][0] < mean\_second\_cut:*

*#     print(class\_R2)*

*# else:*

*#     print(class\_R3)*

        for j in range(len(*Y\_test*[0])):

            if *Y\_test*[min\_gini\_dim][j] < min\_gini\_mean :

                tree\_wise\_prediction[j] = class\_R1

                if y\_selected\_test[j] == class\_R1:

                    cnt+=1

                    predicted\_class[class\_R1] += 1

            elif *Y\_test*[min\_gini\_dim][j] >= min\_gini\_mean and *Y\_test*[min\_gini\_dim\_second\_cut][j] < mean\_second\_cut :

                tree\_wise\_prediction[j] = class\_R2

                if y\_selected\_test[j] == class\_R2:

                    cnt+=1

                    predicted\_class[class\_R2] += 1

            else :

                if y\_selected\_test[j] == class\_R3:

                    tree\_wise\_prediction[j] = class\_R3

                    cnt+=1

                    predicted\_class[class\_R3] += 1

            real\_class[y\_selected\_test[j]] += 1

    else:

        R1 = [] *# region 1*

        R2 = [] *# region 2*

        R3 = [] *# region 3*

        for j in range(len(*Y*[0])):

            if *Y*[min\_gini\_dim][j] >= min\_gini\_mean  :

                R1.append(j)

            elif *Y*[min\_gini\_dim][j] < min\_gini\_mean and *Y*[min\_gini\_dim\_second\_cut][j] >= mean\_second\_cut :

                R2.append(j)

            else :

                R3.append(j)

        R1\_class\_count = [0]\*3

        R2\_class\_count = [0]\*3

        R3\_class\_count = [0]\*3

        for i in R1:

            R1\_class\_count[y\_selected\_train[i]] += 1

        for i in R2:

            R2\_class\_count[y\_selected\_train[i]] += 1

        for i in R3:

            R3\_class\_count[y\_selected\_train[i]] += 1

*#print(np.argmax(R1\_class\_count) , np.argmax(R2\_class\_count) , np.argmax(R3\_class\_count))*

        class\_R1 = np.argmax(R1\_class\_count)

        class\_R2 = np.argmax(R2\_class\_count)

        class\_R3 = np.argmax(R3\_class\_count)

        print("Class of Region-1 is :",class\_R1, "\nClass of Region-2 is :",class\_R2, "\nClass of Region-3 is :",class\_R3)

        predicted\_class = [0]\*3 ; real\_class = [0]\*3

*# if  Y\_test[min\_gini\_dim][0] >= min\_gini\_mean :*

*#     print(class\_R1)*

*# elif Y\_test[min\_gini\_dim][0] < min\_gini\_mean and Y\_test[min\_gini\_dim\_second\_cut][0] >= mean\_second\_cut:*

*#     print(class\_R2)*

*# else:*

*#     print(class\_R3)*

        for j in range(len(*Y\_test*[0])):

            if *Y\_test*[min\_gini\_dim][j] >= min\_gini\_mean :

                tree\_wise\_prediction[j] = class\_R1

                if y\_selected\_test[j] == class\_R1:

                    cnt+=1

                    predicted\_class[class\_R1] += 1

            elif *Y\_test*[min\_gini\_dim][j] < min\_gini\_mean and *Y\_test*[min\_gini\_dim\_second\_cut][j] >= mean\_second\_cut :

                tree\_wise\_prediction[j] = class\_R2

                if y\_selected\_test[j] == class\_R2:

                    cnt+=1

                    predicted\_class[class\_R2] += 1

            else :

                tree\_wise\_prediction[j] = class\_R3

                if y\_selected\_test[j] == class\_R3:

                    cnt+=1

                    predicted\_class[class\_R3] += 1

            real\_class[y\_selected\_test[j]] += 1

    print("Overall accuracy is:",(cnt / len(*Y\_test*[0] )\*100),"%")

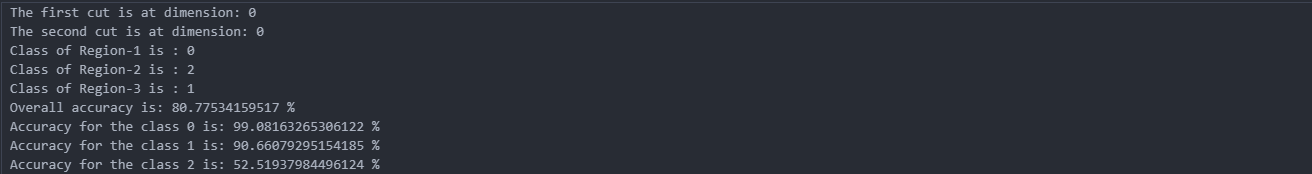
    for i in range(3):

        print("Accuracy for the class" , i , "is:" , (predicted\_class[i]/real\_class[i])\*100,"%")

    return tree\_wise\_prediction

original\_prediction = learn\_tree(Y , Y\_test)

The output of this will be one of the two :



Or

A black rectangular object with white text

Description automatically generated

Now, we need to use bagging by choosing points randomly from the dataset and create 5 datasets , where we can choose one point more than one time .

We use the previously reduced dataset and do the following :

count\_datasets = 5 ; Y = Y.T

*# print(len(Y))*

total\_dataset\_size = len(Y)

bagged\_datasets = [] ; stored\_y\_train = y\_selected\_train ; stored\_indices = []

for i in range(count\_datasets):

    indices = np.random.choice(total\_dataset\_size, *size*=total\_dataset\_size, *replace*=True) *# randomly selecting any indices  from the dataset*

    stored\_indices.append(indices)

    bagged\_dataset = Y[indices]

    bagged\_dataset = bagged\_dataset.T

    bagged\_datasets.append(bagged\_dataset)

*#print(len(indices)) ;*

Y = Y.T ; *#print(len(stored\_indices[0])) ;*

Now , we call the learn\_tree function for each of the dataset created and then predict the class of datapoints from each of the datasets :

print("----------------------------------------------------------")

for no , dataset in enumerate(bagged\_datasets):

*# print(no)*

*# print(no + 1)*

    predicted\_class = [0] \* len(indices)

    y\_selected\_train = []

    for x in stored\_indices[no]:

        y\_selected\_train.append(stored\_y\_train[x])

*# print(len(y\_selected\_train))*

*# print(dataset.shape)*

    predicted\_class = learn\_tree(dataset , Y\_test)

    dataset\_predictions.append(predicted\_class)

    print("----------------------------------------------------------")

Now , for a datapoint we need to predict the class based on the majority element , which we can do as follows:

final\_predictions = []

for j in range(len(Y\_test[0])):

    cnt\_0 = 0 ; cnt\_1 = 0 ; cnt\_2 = 0

    for i in range(count\_datasets):

        if dataset\_predictions[i][j] == 0 :

            cnt\_0 += 1

        elif dataset\_predictions[i][j] == 1 :

            cnt\_1 += 1

        else:

            cnt\_2 += 1

*# print(cnt\_0 , cnt\_1 , cnt\_2)*

    if cnt\_0 >= cnt\_1 and cnt\_0 >= cnt\_2:

        final\_predictions.append(0)

    elif cnt\_1 >= cnt\_0 and cnt\_1 >= cnt\_2:

        final\_predictions.append(1)

    elif cnt\_2 >= cnt\_0 and cnt\_2 >= cnt\_1:

        final\_predictions.append(2)

Now , we predict the overall accuracy and class wise accuracy similarly as we did previously :

print("Overall Accuracy is coming out to be:",(cnt / len(Y\_test[0])) \* 100,"%")

predicted\_class\_final = [0]\*3 ; real\_class\_final = [0]\*3

for i in range(len(Y\_test[0])):

    if final\_predictions[i] == 0 and y\_selected\_test[i] == 0:

        predicted\_class\_final[0] += 1

    if y\_selected\_test[i] == 0:

        real\_class\_final[0] += 1

    if final\_predictions[i] == 1 and y\_selected\_test[i] == 1:

        predicted\_class\_final[1] += 1

    if y\_selected\_test[i] == 1:

        real\_class\_final[1] += 1

    if final\_predictions[i] == 2 and y\_selected\_test[i] == 2:

        predicted\_class\_final[2] += 1

    if y\_selected\_test[i] == 2:

        real\_class\_final[2] += 1

*# Now the class wise accuracy*

for i in range(3):

    print("Accuracy for the class" , i , "is:" , (predicted\_class\_final[i]/real\_class\_final[i])\*100,"%")

print(*end*="----------------------------------------------------------")

By this we get the one of the following outputs as :

----------------------------------------------------------

The first cut is at dimension: 0

The second cut is at dimension: 0

Class of Region-1 is : 0

Class of Region-2 is : 2

Class of Region-3 is : 1

Overall accuracy is: 80.87067047982205 %

Accuracy for the class 0 is: 99.08163265306122 %

Accuracy for the class 1 is: 90.57268722466961 %

Accuracy for the class 2 is: 52.90697674418605 %

----------------------------------------------------------

The first cut is at dimension: 0

The second cut is at dimension: 0

Class of Region-1 is : 0

Class of Region-2 is : 2

Class of Region-3 is : 1

Overall accuracy is: 80.71178900540197 %

Accuracy for the class 0 is: 99.08163265306122 %

Accuracy for the class 1 is: 90.48458149779735 %

Accuracy for the class 2 is: 52.51937984496124 %

----------------------------------------------------------

The first cut is at dimension: 0

The second cut is at dimension: 1

Class of Region-1 is : 1

Class of Region-2 is : 2

Class of Region-3 is : 0

Overall accuracy is: 77.59771210676834 %

Accuracy for the class 0 is: 87.55102040816325 %

Accuracy for the class 1 is: 99.91189427312776 %

Accuracy for the class 2 is: 43.604651162790695 %

----------------------------------------------------------

The first cut is at dimension: 0

The second cut is at dimension: 0

Class of Region-1 is : 0

Class of Region-2 is : 2

Class of Region-3 is : 1

Overall accuracy is: 80.743565300286 %

Accuracy for the class 0 is: 99.08163265306122 %

Accuracy for the class 1 is: 90.48458149779735 %

Accuracy for the class 2 is: 52.616279069767444 %

----------------------------------------------------------

The first cut is at dimension: 0

The second cut is at dimension: 1

Class of Region-1 is : 1

Class of Region-2 is : 2

Class of Region-3 is : 0

Overall accuracy is: 77.37527804258023 %

Accuracy for the class 0 is: 87.95918367346938 %

Accuracy for the class 1 is: 99.91189427312776 %

Accuracy for the class 2 is: 42.53875968992248 %

----------------------------------------------------------

----------------------Bagging-Final-Result-------------------------

Overall Accuracy is coming out to be: 80.77534159517 %

Accuracy for the class 0 is: 99.08163265306122 %

Accuracy for the class 1 is: 90.57268722466961 %

Accuracy for the class 2 is: 52.616279069767444 %

End of Document