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_d
im,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_gin
i_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_great
ermean,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_grea
termean,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 :

```
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: 80.77534159517 %
Accuracy for the class 0 is: 99.08163265306122 %
Accuracy for the class 1 is: 90.66079295154185 %
Accuracy for the class 2 is: 52.51937984496124 %
```

Or

```
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: 77.3117254528122 %
Accuracy for the class 0 is: 87.34693877551021 %
Accuracy for the class 1 is: 99.91189427312776 %
Accuracy for the class 2 is: 42.92635658914728 %
```

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