February-May 2021 Semester

CS5691: Pattern recognition and Machine Learning

Programming Assignment 2

Codes

<u>Team 32</u>

Varun Srinivas Venkatesh

MM17B036

Hrishikesh Kambale

ME18B142

S Sabesh Vishwanath

MM18B112

Task 1 : Naive-Bayes Classifier

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn.metrics import accuracy_score
from matplotlib.lines import Line2D


#Import Data
data_train = pd.read_csv('train.csv', header = None, names =
['Variable1', 'Variable2', 'Class'])
dev_data = pd.read_csv('dev.csv', header = None, names =
['Variable1', 'Variable2', 'Class'])

#Test-Validate Split
shuffled = dev_data.sample(frac=1, random_state=10)
test_subset,val_subset = np.split(shuffled.sample(frac=1,
random_state=4),[int(0.5*len(dev_data))])

#Test-Train-Validate data
x_test = test_subset.drop('Class',axis = 1).to_numpy()
y_test = test_subset['Class'].to_numpy()

x_val = val_subset.drop('Class',axis = 1).to_numpy()
y_val = val_subset['Class'].to_numpy()

x_train = data_train.drop('Class',axis = 1).to_numpy()
y_train = data_train['Class'].to_numpy()

#Split test data by class
df0 = data_train[data_train['Class'] == 0].drop('Class',axis = 1)
df1 = data_train[data_train['Class'] == 1].drop('Class',axis = 1)
df2 = data_train[data_train['Class'] == 2].drop('Class',axis = 1)
df3 = data_train[data_train['Class'] == 3].drop('Class',axis = 1)

#Function to Calculate Probab
def post_prob_ref(x,mean,cov_mat):
    temp1 = np.linalg.det(cov_mat)
```

```
    cov_mat_inv = np.linalg.inv(cov_mat)
    temp2 = x - mean
    temp3 = np.transpose(temp2)
    temp4 = (-0.5)*np.dot(temp3,np.dot(cov_mat_inv,temp2))
    probab = (temp1**-0.5)*(math.exp(temp4))
    return probab

#Calculate mean specific to class
mean0 = np.transpose(np.array(df0.mean()))
mean1 = np.transpose(np.array(df1.mean()))
mean2 = np.transpose(np.array(df2.mean()))
mean3 = np.transpose(np.array(df3.mean()))
mean = [mean0.copy(),mean1.copy(),mean2.copy(),mean3.copy()]

del shuffled, test_subset, val_subset,mean0,mean1,mean2,mean3

#%%
#Part 1: Covariance Matrix is same and equal to sigma^2 * I
independant = data_train.drop('Class', axis = 1)
sigma_squared = independant.var()
Cov_mat_basic = np.array([[sigma_squared[0],0],[0,sigma_squared[1]]])
y_test_pred_basic = []
y_val_pred_basic = []
y_train_pred_basic = []

for x in x_test:
    post_prob = [post_prob_ref(x,i,Cov_mat_basic) for i in mean]
    largest = max(post_prob)
    y_test_pred_basic.append(post_prob.index(largest))

for x in x_val:
    post_prob = [post_prob_ref(x,i,Cov_mat_basic) for i in mean]
    largest = max(post_prob)
    y_val_pred_basic.append(post_prob.index(largest))

for x in x_train:
    post_prob = [post_prob_ref(x,i,Cov_mat_basic) for i in mean]
    largest = max(post_prob)
    y_train_pred_basic.append(post_prob.index(largest))
```

```
train_accuracy_basic = accuracy_score(y_train, y_train_pred_basic)
test_accuracy_basic = accuracy_score(y_test, y_test_pred_basic)
val_accuracy_basic = accuracy_score(y_val, y_val_pred_basic)

del independant,sigma_squared,x,post_prob

#Part 2: Covariance Matrix is same and equal to C
independant = data_train.drop('Class', axis = 1).to_numpy()
Cov_mat_full = np.cov(independant,rowvar = False)
y_test_pred_full = []
y_val_pred_full = []
y_train_pred_full = []

for x in x_test:
    post_prob = [post_prob_ref(x,i,Cov_mat_full) for i in mean]
    largest = max(post_prob)
    y_test_pred_full.append(post_prob.index(largest))

for x in x_val:
    post_prob = [post_prob_ref(x,i,Cov_mat_full) for i in mean]
    largest = max(post_prob)
    y_val_pred_full.append(post_prob.index(largest))

for x in x_train:
    post_prob = [post_prob_ref(x,i,Cov_mat_full) for i in mean]
    largest = max(post_prob)
    y_train_pred_full.append(post_prob.index(largest))

train_accuracy_full = accuracy_score(y_train, y_train_pred_full)
test_accuracy_full = accuracy_score(y_test, y_test_pred_full)
val_accuracy_full = accuracy_score(y_val, y_val_pred_full)

del independant,post_prob,x

#Part 3: Covariance Matrix is different for all classes
Cov_mat0 = np.cov(df0.to_numpy(),rowvar = False)
Cov_mat1 = np.cov(df1.to_numpy(),rowvar = False)
Cov_mat2 = np.cov(df2.to_numpy(),rowvar = False)
```

```python
Cov_mat3 = np.cov(df3.to_numpy(),rowvar = False)
Cov_mat =
[Cov_mat0.copy(),Cov_mat1.copy(),Cov_mat2.copy(),Cov_mat3.copy()]
del Cov_mat0,Cov_mat1,Cov_mat2,Cov_mat3

y_test_pred = []
y_val_pred = []
y_train_pred = []

for x in x_test:
    post_prob = [post_prob_ref(x,mean[i],Cov_mat[i]) for i in
range(4)]
    largest = max(post_prob)
    y_test_pred.append(post_prob.index(largest))

for x in x_val:
    post_prob = [post_prob_ref(x,mean[i],Cov_mat[i]) for i in
range(4)]
    largest = max(post_prob)
    y_val_pred.append(post_prob.index(largest))

for x in x_train:
    post_prob = [post_prob_ref(x,mean[i],Cov_mat[i]) for i in
range(4)]
    largest = max(post_prob)
    y_train_pred.append(post_prob.index(largest))

train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
val_accuracy = accuracy_score(y_val, y_val_pred)

del x

#Confusion Matrix
from sklearn.metrics import confusion_matrix
y = np.append(y_train,np.append(y_test,y_val))
y_pred = np.append(y_train_pred,np.append(y_test_pred,y_val_pred))
confusion_matrix = confusion_matrix(y, y_pred)
```

```
#%%
#Plot
x1 = np.linspace(-13,13,300)
x2 = np.linspace(-13,13,300)
z = np.zeros([300,300])
z_basic = np.zeros([300,300])
z_full = np.zeros([300,300])

for i in range(300):
    for j in range(300):
        x = np.array([x1[i],x2[j]])
        post_prob = [post_prob_ref(x,mean[k],Cov_mat[k]) for k in
range(4)]
        largest = max(post_prob)
        z[i][299-j] = post_prob.index(largest)

for i in range(300):
    for j in range(300):
        x = np.array([x1[i],x2[j]])
        post_prob = [post_prob_ref(x,mean[k],Cov_mat_basic) for k in
range(4)]
        largest = max(post_prob)
        z_basic[i][299-j] = post_prob.index(largest)

for i in range(300):
    for j in range(300):
        x = np.array([x1[i],x2[j]])
        post_prob = [post_prob_ref(x,mean[k],Cov_mat_full) for k in
range(4)]
        largest = max(post_prob)
        z_full[i][299-j] = post_prob.index(largest)

#Colormap
cmap = plt.cm.coolwarm
custom_lines = [Line2D([0], [0], color=cmap(0), lw=4),
                Line2D([0], [0], color=cmap(0.333), lw=4),
                Line2D([0], [0], color=cmap(0.666), lw=4),
                Line2D([0], [0], color=cmap(1), lw=4)]
#Plot 1
```

```
fig, ax = plt.subplots()
plt.scatter(data_train['Variable1'],data_train['Variable2'],c =
data_train['Class'])
im = ax.imshow(z,extent = [-13,13,-13,13],
interpolation='none',origin = 'upper',cmap = 'coolwarm')
ax.legend(custom_lines, ['Class 0', 'Class 1', 'Class 2', 'Class 3'])
ax.set_xlabel('Variable 1')
ax.set_ylabel('Variable 2')
ax.set_title('Decision Region for Case (c)')
plt.show()

#Plot 2
fig, ax = plt.subplots()
plt.scatter(data_train['Variable1'],data_train['Variable2'],c =
data_train['Class'])
im = ax.imshow(z_basic,extent = [-13,13,-13,13],
interpolation='none',origin = 'upper',cmap = 'coolwarm')
ax.legend(custom_lines, ['Class 0', 'Class 1', 'Class 2', 'Class 3'])
ax.set_xlabel('Variable 1')
ax.set_ylabel('Variable 2')
ax.set_title('Decision Region for Case (a)')
plt.show()

#Plot 3
fig, ax = plt.subplots()
plt.scatter(data_train['Variable1'],data_train['Variable2'],c =
data_train['Class'])
im = ax.imshow(z_full,extent = [-13,13,-13,13],
interpolation='none',origin = 'upper',cmap = 'coolwarm')
ax.legend(custom_lines, ['Class 0', 'Class 1', 'Class 2', 'Class 3'])
ax.set_xlabel('Variable 1')
ax.set_ylabel('Variable 2')
ax.set_title('Decision Region for Case (b)')
plt.show()
```

Task 2 : Bayes Classifier with GMM

```
# -*- coding: utf-8 -*-
"""Assg2_Bayes_classifier_GMM_Models.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1nVeTULXdkg5-ND3pzUs5k-CnDpBR
8lVU

## Importing Libraries
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
import sympy as sym
import warnings
warnings.filterwarnings("ignore")

"""## Extracting Dataset 1B"""

def split(x,y):
  np.random.seed(42)
  shuffle_data=np.zeros((x.shape))
  shuffle_target=np.zeros(len(y))
  shuffle_index=np.random.permutation(len(x))
  for i in range(len(x)):
      shuffle_data[i]=x[shuffle_index[i]]
      shuffle_target[i]=y[shuffle_index[i]]
  half=int(len(x)/2)
  split1=shuffle_data[:half]
  split2=shuffle_data[half:]
  target1=shuffle_target[:half]
  target2=shuffle_target[half:]
  return split1,target1,split2,target2
```

```
df = pd.read_csv('1B_train.csv',header=None)
df_val=pd.read_csv('1B_dev.csv',header=None)
print(df.shape)
df.head(5)

df[2].value_counts()    #Equal class distribution  for given 3
classes

x_orig=np.array(df.iloc[:,0:2])             # Converting into Numpy
array
y_orig=np.array(df.iloc[:,-1])
xval=np.array(df_val.iloc[:,0:2])             # Converting into Numpy
array
yval=np.array(df_val.iloc[:,-1])

def normalise(x):                            # MIN-MAX SCALING ON
INPUT VARIABLES
   return (x - np.min(x,axis=0))/(np.max(x, axis=0) - np.min(x,
axis=0))
   #return x
x_orig =normalise(x_orig)
xval = normalise(xval)

xval,yval,xtest,ytest=split(xval,yval)
x0=x_orig[0:200]
x1=x_orig[200:400]
x2=x_orig[400:600]

print(x_orig.shape)
print(y_orig.shape)

"""## Extracting Dataset 2A"""

df_coast = pd.read_csv('coast_train.csv')
df_coast_val=pd.read_csv('coast_dev.csv')

df_highway = pd.read_csv('highway_train.csv')
df_highway_val=pd.read_csv('highway_dev.csv')
```

```
df_insidecity = pd.read_csv('insidecity_train.csv')
df_insidecity_val=pd.read_csv('insidecity_dev.csv')

df_street = pd.read_csv('street_train.csv')
df_street_val=pd.read_csv('street_dev.csv')

df_tallbuilding = pd.read_csv('tallbuilding_train.csv')
df_tallbuilding_val=pd.read_csv('tallbuilding_dev.csv')

x_coast=normalise(np.array(df_coast.iloc[:,1:]))
x_coast_val=normalise(np.array(df_coast_val.iloc[:,1:]))
y_coast=np.zeros(len(x_coast))
y_coast_val=np.zeros(len(x_coast_val))

x_highway=normalise(np.array(df_highway.iloc[:,1:]))
x_highway_val=normalise(np.array(df_highway_val.iloc[:,1:]))
y_highway=np.ones(len(x_highway))
y_highway_val=np.ones(len(x_highway_val))

x_insidecity=normalise(np.array(df_insidecity.iloc[:,1:]))
x_insidecity_val=normalise(np.array(df_insidecity_val.iloc[:,1:]))
y_insidecity=2*np.ones(len(x_insidecity))
y_insidecity_val=2*np.ones(len(x_insidecity_val))

x_street=normalise(np.array(df_street.iloc[:,1:]))
x_street_val=normalise(np.array(df_street_val.iloc[:,1:]))
y_street=3*np.ones(len(x_street))
y_street_val=3*np.ones(len(x_street_val))

x_tallbuilding=normalise(np.array(df_tallbuilding.iloc[:,1:]))
x_tallbuilding_val=normalise(np.array(df_tallbuilding_val.iloc[:,1:])
)
y_tallbuilding=4*np.ones(len(x_tallbuilding))
y_tallbuilding_val=4*np.ones(len(x_tallbuilding_val))

xval_2A=np.vstack((x_coast_val,x_highway_val,x_insidecity_val,x_stree
t_val,x_tallbuilding_val))
yval_2A=np.hstack((y_coast_val,y_highway_val,y_insidecity_val,y_stree
```

```
t_val,y_tallbuilding_val))
xval_2A,yval_2A,xtest_2A,ytest_2A=split(xval_2A,yval_2A)
xorig_2A=np.vstack((x_coast,x_highway,x_insidecity,x_street,x_tallbui
lding))
yorig_2A=np.hstack((y_coast,y_highway,y_insidecity,y_street,y_tallbui
lding))

"""## Extracting Dataset 2B"""

def convert_numpy(file_path):                        # FUNCTION TO
CONVERT PANDAS FRAME INTO NUMPY ARRAY
    df= pd.read_csv(file_path,header=None)
    new= df[0].str.split(" ", n = 23, expand = True)
    new.drop(df.index[len(df)-1],inplace=True)
    x=new.values
    x=x.astype('float64')
    return x
coast_train_path='/content/coast_combine_train.jpg_color_edh_entropy'
# All THESE ARE COMBINED FILES FOR ALL IMAGE FOR GIVEN TRAIN/VAL SET
FOR EACH CLASS
coast_val_path='/content/coast_combine_dev'
highway_train_path='/content/highway_combine_train'
highway_val_path='/content/highway_combine_dev'
insidecity_train_path='/content/insidecity_combine_train'
insidecity_val_path='/content/insidecity_combine_dev'
street_train_path='/content/street_combine_train'
street_val_path='/content/street_combine_dev'
tallbuilding_train_path='/content/tallbuilding_combine_train'
tallbuilding_val_path='/content/tallbuilding_combine_dev'

x_coast2b=convert_numpy(coast_train_path)
# ALL DATA CONVERTED INTO NUMPY ARRAY
x_coast_val2b=convert_numpy(coast_val_path)
x_highway2b=convert_numpy(highway_train_path)
x_highway_val2b=convert_numpy(highway_val_path)
x_insidecity2b=convert_numpy(insidecity_train_path)
x_insidecity_val2b=convert_numpy(insidecity_val_path)
x_street2b=convert_numpy(street_train_path)
x_street_val2b=convert_numpy(street_val_path)
```

```
x_tallbuilding2b=convert_numpy(tallbuilding_train_path)
x_tallbuilding_val2b=convert_numpy(tallbuilding_val_path)
print('No. of images in coast train=',len(x_coast2b)/36)
print('No. of images in coast val=',len(x_coast_val2b)/36)
print('No. of images in highway train=',len(x_highway2b)/36)
print('No. of images in highway val=',len(x_highway_val2b)/36)
print('No. of images in insidecity train=',len(x_insidecity2b)/36)
print('No. of images in insidecity val=',len(x_insidecity_val2b)/36)
print('No. of images in street train=',len(x_street2b)/36)
print('No. of images in street val=',len(x_street_val2b)/36)
print('No. of images in tallbuilding
train=',len(x_tallbuilding2b)/36)
print('No. of images in tallbuilding
val=',len(x_tallbuilding_val2b)/36)

x_coast2b=x_coast2b.reshape((251,36,23))
x_coast_val2b=x_coast_val2b.reshape((73,36,23))
x_highway2b=x_highway2b.reshape((182,36,23))
x_highway_val2b=x_highway_val2b.reshape((52,36,23))
x_insidecity2b=x_insidecity2b.reshape((215,36,23))
x_insidecity_val2b=x_insidecity_val2b.reshape((62,36,23))
x_street2b=x_street2b.reshape((204,36,23))
x_street_val2b=x_street_val2b.reshape((58,36,23))
x_tallbuilding2b=x_tallbuilding2b.reshape((249,36,23))
x_tallbuilding_val2b=x_tallbuilding_val2b.reshape((71,36,23))

y_coast2b=np.zeros(int(len(x_coast2b)))
y_coast_val2b=np.zeros(int(len(x_coast_val2b)))
y_highway2b=np.ones(int(len(x_highway2b)))
y_highway_val2b=np.ones(int(len(x_highway_val2b)))
y_insidecity2b=2*np.ones(int(len(x_insidecity2b)))
y_insidecity_val2b=2*np.ones(int(len(x_insidecity_val2b)))
y_street2b=3*np.ones(int(len(x_street2b)))
y_street_val2b=3*np.ones(int(len(x_street_val2b)))
y_tallbuilding2b=4*np.ones(int(len(x_tallbuilding2b)))
y_tallbuilding_val2b=4*np.ones(int(len(x_tallbuilding_val2b)))

xval_2b=np.vstack((x_coast_val2b,x_highway_val2b,x_insidecity_val2b,x
_street_val2b,x_tallbuilding_val2b))
```

```
yval_2b=np.hstack((y_coast_val2b,y_highway_val2b,y_insidecity_val2b,y
_street_val2b,y_tallbuilding_val2b))
xval_2b,yval_2b,xtest_2b,ytest_2b=split(xval_2b,yval_2b)
xorig_2b=np.vstack((x_coast2b,x_highway2b,x_insidecity2b,x_street2b,x
_tallbuilding2b))
yorig_2b=np.hstack((y_coast2b,y_highway2b,y_insidecity2b,y_street2b,y
_tallbuilding2b))

"""## Plotting Training pts for 1B Data"""

u1=x_orig[:,0]
u2=x_orig[:,1]
v=y_orig
fig = plt.figure(figsize=(8,4))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(u1,u2,v, c='r')

ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('f')
ax.set_title('Plotting Training DataPoints ')
plt.show()

"""# Modelling
## Model:1 Bayes Classifier with GMM, Full Covariance matrix
"""

class GMM_FULL_COVARIANCE:
  def __init__(self,k,d):
    self.k=k
    self.d=d
  def initialisation(self,x):
    d=self.d
    k=self.k
    class kmeans_clustering:
                def __init__(self,k,d):
                    self.k=int(k)
                    self.d=int(d)
```

```
        def initialize_centroids(self,x):
            #returns k centroids from the initial points
            centroids = x.copy()
            np.random.shuffle(centroids)
            return centroids[:self.k]

        def closest_centroid(self,x,centroids):
            #returns an array containing the index to the
nearest centroid for each point
            self.distances = np.sqrt(((x - centroids[:,
np.newaxis])**2).sum(axis=2))
            return np.argmin(self.distances, axis=0)

        def zi_ni_ui(self,x,assign_cluster_index):
            z=np.ones((len(x),self.k))
            n=np.ones((1,self.k))
            u=np.ones((self.k,self.d))
            w_q=np.ones((1,self.k))

            for j in range(self.k):
              for i in range(len(x)):
                  if (assign_cluster_index[i]==j):
                    z[i,j]=1
                  else:
                    z[i,j]=0
              n[0,j]=np.sum(z[:,j])

u[j,:]=(np.sum(z[:,j].reshape(-1,1)*x,axis=0))/n[0,j]
              w_q[0,j]=n[0,j]/len(x)

            return w_q, z, u ,n

        def covariance_calc(self,x,z,u,n):
           covariance=np.zeros((self.k,self.d,self.d))
           for j in range(self.k):
              for i in range(len(x)):

covariance[j,:,:]+=(z[i,j]*((x[i,:]-u[j,:]).reshape(1,-1).T)@((x[i,:]
-u[j,:]).reshape(1,-1)))
```

```
                      covariance[j,:,:]=covariance[j,:,:]/n[0,j]

                   return covariance

    def clustering(d,x,k):
          clus=kmeans_clustering(k,d)
          centroids=clus.initialize_centroids(x)
          zprev=np.zeros((len(x),k))
          znext=np.ones((len(x),k))
          count=0
          while (np.linalg.norm(znext-zprev)>0.001):
              count=count+1
              assign_cluster_index=clus.closest_centroid(x,centroids)

w_q,zprev,centroids,n_q=clus.zi_ni_ui(x,assign_cluster_index)
              assign_cluster_index=clus.closest_centroid(x,centroids)

w_q,znext,centroids,n_q=clus.zi_ni_ui(x,assign_cluster_index)
          covariance=clus.covariance_calc(x,znext,centroids,n_q)
          return w_q,centroids,covariance
    return clustering(d,x,k)

  def normal(self,x,u,c):
      k=self.k
      d=self.d

prob_density=1/((2*3.14159)**(d/2)*(np.linalg.det(c))**0.5)*np.exp(-0
.5*((x-u).reshape(1,-1))@(np.linalg.inv(c))@((x-u).reshape(1,-1).T))
      return prob_density

  def E_STEP_gamma_nq(self,w,u,covariance,x):
      k=self.k
      d=self.d
      gamma_q=np.zeros((len(x),self.k))
      def normal(x,u,c):

prob_density=1/((2*3.14159)**(d/2)*np.linalg.det(c)**0.5)*np.exp(-0.5
*((x-u).reshape(1,-1))@(np.linalg.inv(c))@((x-u).reshape(1,-1).T))
        return prob_density
```

```
      def denominator_sum(i):
        sum=0
        for j in range(k):
          sum=sum+w[0,j]*normal(x[i,:],u[j,:],covariance[j,:,:])
        return sum

      for i in range(len(x)):
        for j in range(k):

gamma_q[i,j]=w[0,j]*normal(x[i,:],u[j,:],covariance[j,:,:])/denominat
or_sum(i)
      return  gamma_q

  def M_STEP_expectation_maximization(self,gamma,x):
    k=self.k
    d=self.d
    n_q=np.sum(gamma,axis=0)
    #print('nq',n_q)
    wq_new=n_q/len(x)
    cq_new=np.zeros((k,d,d))
    uq_new=np.zeros((k,d))
    for j in range(k):
      for i in range(len(x)):
           uq_new[j,:]+=gamma[i,j]*x[i,:]

#cq_new[j,:,:]+=(gamma[i,j]*((x[i,:]-uq_new[j,:]).reshape(1,-1).T) @
((x[i,:]-uq_new[j,:]).reshape(1,-1)))
      uq_new[j,:]/=n_q[j]
      #cq_new[j,:,:]/=n_q[j]
    for j in range(k):
      for i in range(len(x)):

cq_new[j,:,:]+=(gamma[i,j]*((x[i,:]-uq_new[j,:]).reshape(1,-1).T) @
((x[i,:]-uq_new[j,:]).reshape(1,-1)))
      cq_new[j,:,:]/=n_q[j]
    return wq_new,uq_new,cq_new

def final_optimization_full_covariance(k,d,x):
    gmm=GMM_FULL_COVARIANCE(k,d)
```

```
    wprev,uprev,cprev=gmm.initialisation(x)
    gamma_prev=gmm.E_STEP_gamma_nq(wprev,uprev,cprev,x)
    #print(gamma_prev)
    log_likeli_prev=0
    log_likeli_next=1000
    count=0
    while ((log_likeli_next-log_likeli_prev)>=1):
        count+=1
        log_likeli_prev=0
        prob_density_sum1=0
        for i in range(len(x)):
          for j in range(k):

prob_density_sum1+=wprev[0,j]*gmm.normal(x[i,:],uprev[j,:],cprev[j,:,
:])
          log_likeli_prev+=np.log(prob_density_sum1)
        #print('PREVIOUS',log_likeli_prev)


wnew,unew,cnew=gmm.M_STEP_expectation_maximization(gamma_prev,x)
        wnew=wnew.reshape(1,-1)
        gamma_next=gmm.E_STEP_gamma_nq(wnew,unew,cnew,x)

        log_likeli_next=0
        prob_density_sum2=0
        for i in range(len(x)):
           for j in range(k):

prob_density_sum2+=wnew[0,j]*gmm.normal(x[i,:],unew[j,:],cnew[j,:,:])
           log_likeli_next+=np.log(prob_density_sum2)
        wprev=wnew
        uprev=unew
        cprev=cnew
        gamma_prev=gamma_next
    wnew=np.squeeze(wnew)
        #print('count',count)
    return wnew,unew,cnew

def per_class_GMM_full_covariance(w,u,c,x):
```

```
    k=u.shape[0]
    d=x.shape[1]
    if (w.ndim==0):
        w=w.reshape(1,1)
    else:
      pass
    gmm=GMM_FULL_COVARIANCE(k,d)
    prob=np.zeros((len(x),k))
    prob_net=np.zeros(len(x))
    for i in range(len(x)):
         for j in range(k):
           prob[i,j]=w[j]*gmm.normal(x[i,:],u[j,:],c[j,:,:])
         prob_net[i]=np.sum(prob[i,:])
    return prob_net

def accuracy_score(ytrue,ypred):
  count=0
  for i in range(len(ytrue)):
        if (ytrue[i]==ypred[i]):
          count+=1
        else:
          pass
  accuracy=count/len(ytrue)
  return accuracy

"""### 1.1 Predicting on Data 1B (GMM_FULL_COVARIANCE CASE)"""

def predict_GMM_FULL_COVARIANCE(k,d,x_predict,xtrain):
# PREDICTING ON DATASET 1B

wstar0,ustar0,cstar0=final_optimization_full_covariance(k,d,xtrain[0:
200])

wstar1,ustar1,cstar1=final_optimization_full_covariance(k,d,xtrain[20
0:400])

wstar2,ustar2,cstar2=final_optimization_full_covariance(k,d,xtrain[40
0:600])
```

```
prob0=per_class_GMM_full_covariance(wstar0,ustar0,cstar0,x_predict)

prob1=per_class_GMM_full_covariance(wstar1,ustar1,cstar1,x_predict)

prob2=per_class_GMM_full_covariance(wstar2,ustar2,cstar2,x_predict)
    prob0=prob0.reshape(-1,1)
    prob1=prob1.reshape(-1,1)
    prob2=prob2.reshape(-1,1)
    prob_net=np.column_stack([prob0,prob1,prob2])
    ypred=np.zeros(len(x_predict))
    for i in range(len(x_predict)):
      ypred[i]=np.argmax(prob_net[i,:])
    return ypred

ypred_train=predict_GMM_FULL_COVARIANCE(6,2,x_orig,x_orig)   # No. of
gaussians=plug different values for testing , X dimension=2 ,
TRAINING SET
ypred_val=predict_GMM_FULL_COVARIANCE(6,2,xval,x_orig)       # No. of
gaussians=plug different values for testing  , X dimension=2 ,
VALIDATION SET
ypred_test=predict_GMM_FULL_COVARIANCE(6,2,xtest,x_orig)
accuracy_train=round(accuracy_score(y_orig, ypred_train),4)
accuracy_val=round(accuracy_score(yval, ypred_val),4)
accuracy_test=round(accuracy_score(ytest, ypred_test),4)
print('Train Accuracy: ',accuracy_train)
print('Validation Accuracy: ',accuracy_val)
print('Test Accuracy: ',accuracy_test)
from sklearn.metrics import confusion_matrix
training=confusion_matrix(y_orig, ypred_train)
val=confusion_matrix(yval, ypred_val)
test=confusion_matrix(ytest, ypred_test)
print('Confusion matrix for Training Data: ',training)
print('Confusion matrix for Validation Data: ',val)
print('Confusion matrix for Test Data: ', test)

"""# Plotting For 1B Data  (Full_covariance_GMM)
## Code for plotting 1.Decision region 2.Training pts 3.Level
curves(Ellipses) for GMM_Full Covariance case on data 1B
"""
```

```
k=6
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
def multivariate_gaussian(pos, mu, Sigma):
    """Return the multivariate Gaussian distribution on array pos.
    pos is an array constructed by packing the meshed arrays of
variables
    x_1, x_2, x_3, ..., x_k into its _last_ dimension. """
    n = mu.shape[0]
    Sigma_det = np.linalg.det(Sigma)
    Sigma_inv = np.linalg.inv(Sigma)
    N = np.sqrt((2*np.pi)**n * Sigma_det)
    # This einsum call calculates (x-mu)T.Sigma-1.(x-mu) in a
vectorized way across all the input variables.
    fac = np.einsum('...k,kl,...l->...', pos-mu, Sigma_inv, pos-mu)
    return np.exp(-fac / 2) / N

wstar0,ustar0,cstar0=final_optimization_full_covariance(k,2,x_orig[0:
200])
wstar1,ustar1,cstar1=final_optimization_full_covariance(k,2,x_orig[20
0:400])
wstar2,ustar2,cstar2=final_optimization_full_covariance(k,2,x_orig[40
0:600])

X = np.linspace(0,1,100)
Y = np.linspace(0,1,100)
X, Y = np.meshgrid(X, Y)

# Pack X and Y into a single 3-dimensional array
pos = np.empty(X.shape + (2,))
pos[:, :, 0] = X
pos[:, :, 1] = Y

fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111)
# The distribution on the variables X, Y packed into pos.
```

```python
#for i in range(k):
Z0=np.zeros((k,100,100))
Z1=np.zeros((k,100,100))
Z2=np.zeros((k,100,100))
prob0=np.zeros((100,100))
prob1=np.zeros((100,100))
prob2=np.zeros((100,100))
for i in range(k):
      Z0[i] = multivariate_gaussian(pos,ustar0[i],cstar0[i])
      Z1[i] = multivariate_gaussian(pos,ustar1[i],cstar1[i])
      Z2[i] = multivariate_gaussian(pos,ustar2[i],cstar2[i])
      prob0+=wstar0[i]*Z0[i]
      prob1+=wstar1[i]*Z1[i]
      prob2+=wstar2[i]*Z2[i]
for i in range(100):
  for j in range(100):
      p0=prob0[i,j]
      p1=prob1[i,j]
      p2=prob2[i,j]
      n=np.argmax([p0,p1,p2])
      if (n==0):
         plt.scatter(i/100,j/100,c='lightcoral')
      elif (n==1):
         plt.scatter(i/100,j/100,c='lightgreen')
      else:
         plt.scatter(i/100,j/100,c='lightblue')
      print('i={}, j={}'.format(i,j))

for i in range(k):
      Z1 = multivariate_gaussian(pos,ustar0[i],cstar0[i])
      Z2 = multivariate_gaussian(pos,ustar1[i],cstar1[i])
      Z3 = multivariate_gaussian(pos,ustar2[i],cstar2[i])
      class1 = ax.contour(X, Y, Z1, 5,  cmap='RdGy')
      class2 = ax.contour(X, Y, Z2, 5,  cmap='RdGy')
      class3 = ax.contour(X, Y, Z3, 5,  cmap='RdGy')

plt.axes().set_aspect('equal')
plt.title('Plot of Best Bayes_GMM_Full_covariance Model for 
Q={}'.format(k))
```

```
plt.scatter(x_orig[0:200,0],x_orig[0:200,1] ,c='r',label='Class 0')
plt.scatter(x_orig[200:400,0],x_orig[200:400,1] ,c='g',label='Class
1')
plt.scatter(x_orig[400:600,0],x_orig[400:600,1] ,c='b',label='Class
2')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
plt.legend(fontsize='medium')

plt.show()

"""### Predicting on 2A Data (GMM_FULL_COVARIANCE_CASE )"""

def
predict_GMM_FULL_COVARIANCE(k,d,x_predict,x_coast,x_highway,x_insidec
ity,x_street,x_tallbuilding):      # PREDCITING ON DATASET 2A

wstar0,ustar0,cstar0=final_optimization_full_covariance(k,d,x_coast)

wstar1,ustar1,cstar1=final_optimization_full_covariance(k,d,x_highway
)

wstar2,ustar2,cstar2=final_optimization_full_covariance(k,d,x_insidec
ity)

wstar3,ustar3,cstar3=final_optimization_full_covariance(k,d,x_street)

wstar4,ustar4,cstar4=final_optimization_full_covariance(k,d,x_tallbui
lding)

prob0=per_class_GMM_full_covariance(wstar0,ustar0,cstar0,x_predict)

prob1=per_class_GMM_full_covariance(wstar1,ustar1,cstar1,x_predict)

prob2=per_class_GMM_full_covariance(wstar2,ustar2,cstar2,x_predict)

prob3=per_class_GMM_full_covariance(wstar3,ustar3,cstar3,x_predict)

prob4=per_class_GMM_full_covariance(wstar4,ustar4,cstar4,x_predict)
```

```
    prob0=prob0.reshape(-1,1)
    prob1=prob1.reshape(-1,1)
    prob2=prob2.reshape(-1,1)
    prob3=prob3.reshape(-1,1)
    prob4=prob4.reshape(-1,1)
    prob_net=np.column_stack([prob0,prob1,prob2,prob3,prob4])
    ypred=np.zeros(len(x_predict))
    for i in range(len(x_predict)):
      ypred[i]=np.argmax(prob_net[i,:])
    return ypred

ypred_train_2A=predict_GMM_FULL_COVARIANCE(1,24,xorig_2A,x_coast,x_hi
ghway,x_insidecity,x_street,x_tallbuilding)   # No. of gaussians=plug
diff. values for testing , X dimension=23, TRAINING SET
ypred_val_2A=predict_GMM_FULL_COVARIANCE(1,24,xval_2A,x_coast,x_highw
ay,x_insidecity,x_street,x_tallbuilding)      # No. of gaussians=plug
diff. values for testing , X dimension=23, VALIDATION SET
ypred_test_2A=predict_GMM_FULL_COVARIANCE(1,24,xtest_2A,x_coast,x_hig
hway,x_insidecity,x_street,x_tallbuilding)      # No. of
gaussians=plug diff. values for testing , X dimension=23, TEST SET
accuracy_train=round(accuracy_score(yorig_2A, ypred_train_2A),4)
accuracy_val=round(accuracy_score(yval_2A, ypred_val_2A),4)
accuracy_test=round(accuracy_score(ytest_2A, ypred_test_2A),4)
print('Train Accuracy: ',accuracy_train)
print('Validation Accuracy: ',accuracy_val)
print('Test Accuracy: ',accuracy_test)
training=confusion_matrix(yorig_2A, ypred_train_2A)
val=confusion_matrix(yval_2A, ypred_val_2A)
test=confusion_matrix(ytest_2A, ypred_test_2A)
print('Confusion matrix for Training Data: ',training)
print('Confusion matrix for Validation Data: ',val)
print('Confusion matrix for Test Data: ', test)

"""## Model:2 Bayes Classifier with GMM, Diagonal Covariance
matrix"""

class GMM_DIAGONAL_COVARIANCE:
  def __init__(self,k,d):
    self.k=k
```

```
    self.d=d
  def initialisation(self,x):
    d=self.d
    k=self.k
    class kmeans_clustering:
                def __init__(self,k,d):
                    self.k=int(k)
                    self.d=int(d)

                def initialize_centroids(self,x):
                    #returns k centroids from the initial points
                    centroids = x.copy()
                    np.random.shuffle(centroids)
                    return centroids[:self.k]

                def closest_centroid(self,x,centroids):
                    #returns an array containing the index to the
nearest centroid for each point
                    self.distances = np.sqrt(((x - centroids[:,
np.newaxis])**2).sum(axis=2))
                    return np.argmin(self.distances, axis=0)

                def zi_ni_ui(self,x,assign_cluster_index):
                    z=np.ones((len(x),self.k))
                    n=np.ones((1,self.k))
                    u=np.ones((self.k,self.d))
                    w_q=np.ones((1,self.k))

                    for j in range(self.k):
                      for i in range(len(x)):
                          if (assign_cluster_index[i]==j):
                            z[i,j]=1
                          else:
                            z[i,j]=0
                      n[0,j]=np.sum(z[:,j])

u[j,:]=(np.sum(z[:,j].reshape(-1,1)*x,axis=0))/n[0,j]
                      w_q[0,j]=n[0,j]/len(x)
```

```
                    return w_q, z, u ,n

                def covariance_calc(self,x,z,u,n):
                   k=self.k
                   d=self.d
                   def identity(k,d):
                        i = np.identity(d)
                        e=[]
                        for j in range(k):
                          e.append(i)
                        e=np.array(e)
                        return e
                   sigma_sq=np.zeros(k)
                   cq_new=identity(k,d)

                   for j in range(k):
                      for i in range(len(x)):

#sigma_sq[j]+=(z[i,j]*((x[i,:]-u[j,:]).reshape(1,-1))@((x[i,:]-u[j,:]
).reshape(1,-1).T))

cq_new[j,:,:]+=(z[i,j]*((x[i,:]-u[j,:]).reshape(1,-1).T)@((x[i,:]-u[j
,:]).reshape(1,-1)))

#print('initialised_covariance_before_scaling',covariance)
                      cq_new[j,:,:]=cq_new[j,:,:]/n[0,j]

                   return cq_new

    def clustering(d,x,k):
          clus=kmeans_clustering(k,d)
          centroids=clus.initialize_centroids(x)
          zprev=np.zeros((len(x),k))
          znext=np.ones((len(x),k))
          count=0
          while (np.linalg.norm(znext-zprev)>0.001):
              count=count+1
              assign_cluster_index=clus.closest_centroid(x,centroids)
```

```
w_q,zprev,centroids,n_q=clus.zi_ni_ui(x,assign_cluster_index)
              assign_cluster_index=clus.closest_centroid(x,centroids)

w_q,znext,centroids,n_q=clus.zi_ni_ui(x,assign_cluster_index)
          covariance=clus.covariance_calc(x,znext,centroids,n_q)
          return w_q,centroids,covariance
    return clustering(d,x,k)

  def normal(self,x,u,c):
      k=self.k
      d=self.d

prob_density=1/((2*3.14159)**(d/2)*(np.linalg.det(c))**0.5)*np.exp(-0
.5*((x-u).reshape(1,-1))@(np.linalg.inv(c))@((x-u).reshape(1,-1).T))
      return prob_density

  def E_STEP_gamma_nq(self,w,u,covariance,x):
      k=self.k
      d=self.d
      gamma_q=np.zeros((len(x),self.k))
      def normal(x,u,c):

prob_density=1/((2*3.14159)**(d/2)*np.linalg.det(c)**0.5)*np.exp(-0.5
*((x-u).reshape(1,-1))@(np.linalg.inv(c))@((x-u).reshape(1,-1).T))
        return prob_density
      def denominator_sum(i):
        sum=0
        for j in range(k):
          sum=sum+w[0,j]*normal(x[i,:],u[j,:],covariance[j,:,:])
        return sum

      for i in range(len(x)):
        for j in range(k):

gamma_q[i,j]=w[0,j]*normal(x[i,:],u[j,:],covariance[j,:,:])/denominat
or_sum(i)
      return  gamma_q

  def M_STEP_expectation_maximization(self,gamma,x):
```

```
    k=self.k
    d=self.d
    n_q=np.sum(gamma,axis=0)
    wq_new=n_q/len(x)
    uq_new=np.zeros((k,d))
    def identity(k,d):
        i = np.identity(d)
        e=[]
        for j in range(k):
          e.append(i)
        e=np.array(e)
        return e

    sigma_sq=np.zeros(k)
    cq_new=identity(k,d)
    diagonalised_matrix=identity(k,d)

    for j in range(k):
      for i in range(len(x)):
            uq_new[j,:]+=gamma[i,j]*x[i,:]
      uq_new[j,:]/=n_q[j]

    for j in range(k):
      for i in range(len(x)):

#sigma_sq[j]+=(gamma[i,j]*((x[i,:]-uq_new[j,:]).reshape(1,-1)) @
((x[i,:]-uq_new[j,:]).reshape(1,-1).T))

cq_new[j,:,:]+=(gamma[i,j]*((x[i,:]-uq_new[j,:]).reshape(1,-1).T) @
((x[i,:]-uq_new[j,:]).reshape(1,-1)))
      cq_new[j,:,:]/=n_q[j]
      for r in range(d):
        for c in range(d):
            if (r==c):
               diagonalised_matrix[j,r,c]=cq_new[j,r,c]
            else:
               diagonalised_matrix[j,r,c]=0
    return wq_new,uq_new, diagonalised_matrix
```

```
def final_optimization_diagonal_covariance(k,d,x):
    gmm=GMM_DIAGONAL_COVARIANCE(k,d)
    wprev,uprev,cprev=gmm.initialisation(x)
    gamma_prev=gmm.E_STEP_gamma_nq(wprev,uprev,cprev,x)
    #print(gamma_prev)
    log_likeli_prev=0
    log_likeli_next=1000
    count=0
    while ((log_likeli_next-log_likeli_prev)>=1):
        count+=1
        log_likeli_prev=0
        prob_density_sum1=0
        for i in range(len(x)):
          for j in range(k):

prob_density_sum1+=wprev[0,j]*gmm.normal(x[i,:],uprev[j,:],cprev[j,:,
:])
          log_likeli_prev+=np.log(prob_density_sum1)
       # print('PREVIOUS',cprev)


wnew,unew,cnew=gmm.M_STEP_expectation_maximization(gamma_prev,x)
        wnew=wnew.reshape(1,-1)
        gamma_next=gmm.E_STEP_gamma_nq(wnew,unew,cnew,x)

        log_likeli_next=0
        prob_density_sum2=0
        for i in range(len(x)):
           for j in range(k):

prob_density_sum2+=wnew[0,j]*gmm.normal(x[i,:],unew[j,:],cnew[j,:,:])
           log_likeli_next+=np.log(prob_density_sum2)
        wprev=wnew
        uprev=unew
        cprev=cnew
        gamma_prev=gamma_next
    wnew=np.squeeze(wnew)
        #print('count',count)
    return wnew,unew,cnew
```

```
def per_class_GMM_diagonal_covariance(w,u,c,x):
   k=u.shape[0]
   d=x.shape[1]
   if (w.ndim==0):
       w=w.reshape(1,1)
   else:
     pass
   gmm=GMM_FULL_COVARIANCE(k,d)
   prob=np.zeros((len(x),k))
   prob_net=np.zeros(len(x))
   for i in range(len(x)):
        for j in range(k):
          prob[i,j]=w[j]*gmm.normal(x[i,:],u[j,:],c[j,:,:])
        prob_net[i]=np.sum(prob[i,:])
   return prob_net

"""### 2.1 Predicting on 1B Data (GMM_DIAGONAL_COVARIANCE_CASE)"""

def predict_GMM_DIAGONAL_COVARIANCE(k,d,x_predict,xtrain):
# PREDICTION ON DATASET 1B

wstar0,ustar0,cstar0=final_optimization_diagonal_covariance(k,d,xtrai
n[0:200])

wstar1,ustar1,cstar1=final_optimization_diagonal_covariance(k,d,xtrai
n[200:400])

wstar2,ustar2,cstar2=final_optimization_diagonal_covariance(k,d,xtrai
n[400:600])

prob0=per_class_GMM_diagonal_covariance(wstar0,ustar0,cstar0,x_predic
t)

prob1=per_class_GMM_diagonal_covariance(wstar1,ustar1,cstar1,x_predic
t)

prob2=per_class_GMM_diagonal_covariance(wstar2,ustar2,cstar2,x_predic
t)
```

```
    prob0=prob0.reshape(-1,1)
    prob1=prob1.reshape(-1,1)
    prob2=prob2.reshape(-1,1)
    prob_net=np.column_stack([prob0,prob1,prob2])
    ypred=np.zeros(len(x_predict))
    for i in range(len(x_predict)):
      ypred[i]=np.argmax(prob_net[i,:])
    return ypred

ypred_train=predict_GMM_DIAGONAL_COVARIANCE(5,2,x_orig,x_orig)
# No. of gaussians= Plug diff. values for testing  , X dimension=2
ypred_val=predict_GMM_DIAGONAL_COVARIANCE(5,2,xval,x_orig)
ypred_test=predict_GMM_DIAGONAL_COVARIANCE(5,2,xtest,x_orig)
accuracy_train=round(accuracy_score(y_orig, ypred_train),4)
accuracy_val=round(accuracy_score(yval, ypred_val),4)
accuracy_test=round(accuracy_score(ytest, ypred_test),4)
print('Train Accuracy: ',accuracy_train)
print('Validation Accuracy: ',accuracy_val)
print('Test Accuracy: ',accuracy_test)
training=confusion_matrix(y_orig, ypred_train)
val=confusion_matrix(yval, ypred_val)
test=confusion_matrix(ytest, ypred_test)
print('Confusion matrix for Training Data: ',training)
print('Confusion matrix for Validation Data: ',val)
print('Confusion matrix for Test Data: ', test)

"""#  Plotting for 1B data for Diagonal_covariance_GMM
## Code for plotting 1.Decision regions 2. Training pts 3. Level
curves for GMM _Diagonal covariance matrix case for dataset 1b
"""


k=5
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
def multivariate_gaussian(pos, mu, Sigma):
    n = mu.shape[0]
    Sigma_det = np.linalg.det(Sigma)
```

```
    Sigma_inv = np.linalg.inv(Sigma)
    N = np.sqrt((2*np.pi)**n * Sigma_det)
    # This einsum call calculates (x-mu)T.Sigma-1.(x-mu) in a
vectorized way across all the input variables.
    fac = np.einsum('...k,kl,...l->...', pos-mu, Sigma_inv, pos-mu)
    return np.exp(-fac / 2) / N

wstar0,ustar0,cstar0=final_optimization_diagonal_covariance(k,2,x_ori
g[0:200])
wstar1,ustar1,cstar1=final_optimization_diagonal_covariance(k,2,x_ori
g[200:400])
wstar2,ustar2,cstar2=final_optimization_diagonal_covariance(k,2,x_ori
g[400:600])

X = np.linspace(0,1,100)
Y = np.linspace(0,1,100)
X, Y = np.meshgrid(X, Y)

# Pack X and Y into a single 3-dimensional array
pos = np.empty(X.shape + (2,))
pos[:, :, 0] = X
pos[:, :, 1] = Y

fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111)

Z0=np.zeros((k,100,100))
Z1=np.zeros((k,100,100))
Z2=np.zeros((k,100,100))
prob0=np.zeros((100,100))
prob1=np.zeros((100,100))
prob2=np.zeros((100,100))
for i in range(k):
      Z0[i] = multivariate_gaussian(pos,ustar0[i],cstar0[i])
      Z1[i] = multivariate_gaussian(pos,ustar1[i],cstar1[i])
      Z2[i] = multivariate_gaussian(pos,ustar2[i],cstar2[i])
      prob0+=wstar0[i]*Z0[i]
      prob1+=wstar1[i]*Z1[i]
      prob2+=wstar2[i]*Z2[i]
```

```python
for i in range(100):
  for j in range(100):
      p0=prob0[i,j]
      p1=prob1[i,j]
      p2=prob2[i,j]
      n=np.argmax([p0,p1,p2])
      if (n==0):
         plt.scatter(i/100,j/100,c='lightcoral')
      elif (n==1):
         plt.scatter(i/100,j/100,c='lightgreen')
      else:
         plt.scatter(i/100,j/100,c='lightblue')
      print('i={}, j={}'.format(i,j))

for i in range(k):
      Z1 = multivariate_gaussian(pos,ustar0[i],cstar0[i])
      Z2 = multivariate_gaussian(pos,ustar1[i],cstar1[i])
      Z3 = multivariate_gaussian(pos,ustar2[i],cstar2[i])
      class1 = ax.contour(X, Y, Z1, 5,  cmap='RdGy')
      class2 = ax.contour(X, Y, Z2, 5,  cmap='RdGy')
      class3 = ax.contour(X, Y, Z3, 5,  cmap='RdGy')


plt.axes().set_aspect('equal')
plt.title('Plot of Best Bayes_GMM_Diagonal_covariance Model for 
Q={}'.format(k))
plt.scatter(x_orig[0:200,0],x_orig[0:200,1] ,c='r',label='Class 0')
plt.scatter(x_orig[200:400,0],x_orig[200:400,1] ,c='g',label='Class 
1')
plt.scatter(x_orig[400:600,0],x_orig[400:600,1] ,c='b',label='Class 
2')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
plt.legend(fontsize='medium')

plt.show()

"""### 2.2 Predicting on 2A Data (GMM_DIAGONAL_COVARIANCE_CASE)"""
```

```
def
predict_GMM_DIAGONAL_COVARIANCE(k,d,x_predict,x_coast,x_highway,x_ins
idecity,x_street,x_tallbuilding):              # PREDICTION ON
DATASET 1B

wstar0,ustar0,cstar0=final_optimization_diagonal_covariance(k,d,x_coa
st)

wstar1,ustar1,cstar1=final_optimization_diagonal_covariance(k,d,x_hig
hway)

wstar2,ustar2,cstar2=final_optimization_diagonal_covariance(k,d,x_ins
idecity)

wstar3,ustar3,cstar3=final_optimization_diagonal_covariance(k,d,x_str
eet)

wstar4,ustar4,cstar4=final_optimization_diagonal_covariance(k,d,x_tal
lbuilding)

prob0=per_class_GMM_diagonal_covariance(wstar0,ustar0,cstar0,x_predic
t)

prob1=per_class_GMM_diagonal_covariance(wstar1,ustar1,cstar1,x_predic
t)

prob2=per_class_GMM_diagonal_covariance(wstar2,ustar2,cstar2,x_predic
t)

prob3=per_class_GMM_diagonal_covariance(wstar3,ustar3,cstar3,x_predic
t)

prob4=per_class_GMM_diagonal_covariance(wstar4,ustar4,cstar4,x_predic
t)
    prob0=prob0.reshape(-1,1)
    prob1=prob1.reshape(-1,1)
    prob2=prob2.reshape(-1,1)
    prob3=prob3.reshape(-1,1)
    prob4=prob4.reshape(-1,1)
```

```
    prob_net=np.column_stack([prob0,prob1,prob2,prob3,prob4])
    ypred=np.zeros(len(x_predict))
    for i in range(len(x_predict)):
      ypred[i]=np.argmax(prob_net[i,:])
    return ypred

ypred_train_2A=predict_GMM_DIAGONAL_COVARIANCE(3,24,xorig_2A,x_coast,
x_highway,x_insidecity,x_street,x_tallbuilding)         # No. of
gaussians= Plug diff. values for testing  , X dimension=23
ypred_val_2A=predict_GMM_DIAGONAL_COVARIANCE(3,24,xval_2A,x_coast,x_h
ighway,x_insidecity,x_street,x_tallbuilding)
ypred_test_2A=predict_GMM_DIAGONAL_COVARIANCE(3,24,xtest_2A,x_coast,x
_highway,x_insidecity,x_street,x_tallbuilding)      # No. of
gaussians=plug diff. values for testing , X dimension=23, TEST SET
accuracy_train=round(accuracy_score(yorig_2A, ypred_train_2A),4)
accuracy_val=round(accuracy_score(yval_2A, ypred_val_2A),4)
accuracy_test=round(accuracy_score(ytest_2A, ypred_test_2A),4)
print('Train Accuracy: ',accuracy_train)
print('Validation Accuracy: ',accuracy_val)
print('Test Accuracy: ',accuracy_test)
training=confusion_matrix(yorig_2A, ypred_train_2A)
val=confusion_matrix(yval_2A, ypred_val_2A)
test=confusion_matrix(ytest_2A, ypred_test_2A)
print('Confusion matrix for Training Data: ',training)
print('Confusion matrix for Validation Data: ',val)
print('Confusion matrix for Test Data: ', test)

"""## Model:3 Bayes Classifier with KNN (Non_parametric_method)"""

class bayes_KNN:
   def __init__(self,k):
     self.k=k

   def algo(self,xpoint,x):
     k=self.k
     r=0
     #print('point entered',xpoint)
     knn=np.zeros(k)
     distance=np.zeros(len(x))
```

```
    for i in range(len(x)):
         distance[i]=np.linalg.norm((xpoint-x[i]))
    distance=np.sort(distance)
    for i in range(k):
         knn[i]=distance[i]

    knn=knn[::-1]       # Descending order in KNN points distance
wise
    #print(knn)
    r=knn[0]            # r max
    #print(r)
    return r

def bayes_knn_predict(k,x_predict_point,xtrain):          #
PREDICTING FUNCTION
    knn=bayes_KNN(k)
    ypred=np.zeros(x_predict_point.shape[0])
    r_vector=np.zeros(3)            # No. of classes=3
    for i in range(x_predict_point.shape[0]):
        r_vector[0]=knn.algo(x_predict_point[i],xtrain[0:200])
        r_vector[1]=knn.algo(x_predict_point[i],xtrain[200:400])
        r_vector[2]=knn.algo(x_predict_point[i],xtrain[400:600])
        ypred[i]=np.argmin(r_vector)
    return ypred

ypred_train=bayes_knn_predict(10,x_orig,x_orig)          # NEAREST
NEIGHBOURS= PLUG Diff hypere values for KNN's
ypred_val=bayes_knn_predict(10,xval,x_orig)
ypred_test=bayes_knn_predict(10,xtest,x_orig)
accuracy_train=round(accuracy_score(y_orig, ypred_train),4)
accuracy_val=round(accuracy_score(yval, ypred_val),4)
accuracy_test=round(accuracy_score(ytest, ypred_test),4)
print('Train Accuracy: ',accuracy_train)
print('Validation Accuracy: ',accuracy_val)
print('Test Accuracy: ',accuracy_test)
training=confusion_matrix(y_orig, ypred_train)
val=confusion_matrix(yval, ypred_val)
test=confusion_matrix(ytest, ypred_test)
print('Confusion matrix for Training Data: ',training)
```

```
print('Confusion matrix for Validation Data: ',val)
print('Confusion matrix for Test Data: ', test)

"""## PLOT Bayes with KNN DECISION BOUNDARY"""

k=5
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

X = np.linspace(0,1,100)
Y = np.linspace(0,1,100)
X, Y = np.meshgrid(X, Y)

# Pack X and Y into a single 3-dimensional array
pos = np.empty(X.shape + (2,))
pos[:, :, 0] = X
pos[:, :, 1] = Y

fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111)
def bayes_knn_predict_new(k,x_predict_point,xtrain):          #
PREDICTING FUNCTION
    knn=bayes_KNN(k)
    ypred=0
    r_vector=np.zeros(3)             # No. of classes=3
    r_vector[0]=knn.algo(x_predict_point,xtrain[0:200])
    r_vector[1]=knn.algo(x_predict_point,xtrain[200:400])
    r_vector[2]=knn.algo(x_predict_point,xtrain[400:600])
    ypred=np.argmin(r_vector)
    return ypred
for i in range(100):
  for j in range(100):
      current_point=np.array([i/100,j/100])
      y=bayes_knn_predict_new(10,current_point,x_orig)
      #print(y)
      if (y==0):
         plt.scatter(i/100,j/100,c='lightcoral')
```

```
      elif (y==1):
         plt.scatter(i/100,j/100,c='lightgreen')
      else:
         plt.scatter(i/100,j/100,c='lightblue')
      print('i={}, j={}'.format(i,j))



plt.axes().set_aspect('equal')
plt.title('Plot of Best Bayes with KNN Model for Knn={}'.format(k))
plt.scatter(x_orig[0:200,0],x_orig[0:200,1] ,c='r',label='Class 0')
plt.scatter(x_orig[200:400,0],x_orig[200:400,1] ,c='g',label='Class
1')
plt.scatter(x_orig[400:600,0],x_orig[400:600,1] ,c='b',label='Class
2')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
plt.legend(fontsize='medium')

plt.show()

"""## Model 4: GMM_Full_Covariance_Dataset_2B"""

from decimal import Decimal
def per_class_GMM_covariance_2b(w,u,c,x):
   k=u.shape[0]
   d=x.shape[2]
   if (w.ndim==0):
       w=w.reshape(1,1)
   else:
     pass
   k=int(len(w))
   gmm=GMM_FULL_COVARIANCE(k,d)
   prob_features=np.zeros(36)
   prob=np.zeros((36,k))
   prob_net=np.zeros(len(x))
   for t in range(x.shape[0]):
      for i in range(36):
            for j in range(k):
```

```
prob[i,j]=(w[j]*gmm.normal(x[t,i,:],u[j,:],c[j,:,:]))/(10**(32))
            prob_features[i]=np.sum(prob[i,:])
      prob_net[t]=np.prod(prob_features)
   return prob_net

def
predict_GMM_FULL_COVARIANCE_FOR_2B_DATA(k,d,x_predict,x_coast,x_highw
ay,x_insidecity,x_street,x_tallbuilding):      # PREDCITING ON
DATASET 2A
    x_coast=x_coast.reshape((int(x_coast.shape[0]*36)),23)
    x_highway=x_highway.reshape((int(x_highway.shape[0]*36)),23)

x_insidecity=x_insidecity.reshape((int(x_insidecity.shape[0]*36)),23)
    x_street=x_street.reshape((int(x_street.shape[0]*36)),23)

x_tallbuilding=x_tallbuilding.reshape((int(x_tallbuilding.shape[0]*36
)),23)


wstar0,ustar0,cstar0=final_optimization_full_covariance(k,d,x_coast)

wstar1,ustar1,cstar1=final_optimization_full_covariance(k,d,x_highway
)

wstar2,ustar2,cstar2=final_optimization_full_covariance(k,d,x_insidec
ity)

wstar3,ustar3,cstar3=final_optimization_full_covariance(k,d,x_street)

wstar4,ustar4,cstar4=final_optimization_full_covariance(k,d,x_tallbui
lding)

    prob0=per_class_GMM_covariance_2b(wstar0,ustar0,cstar0,x_predict)
    prob1=per_class_GMM_covariance_2b(wstar1,ustar1,cstar1,x_predict)
    prob2=per_class_GMM_covariance_2b(wstar2,ustar2,cstar2,x_predict)
    prob3=per_class_GMM_covariance_2b(wstar3,ustar3,cstar3,x_predict)
    prob4=per_class_GMM_covariance_2b(wstar4,ustar4,cstar4,x_predict)
```

```
    prob0=prob0.reshape(-1,1)
    prob1=prob1.reshape(-1,1)
    prob2=prob2.reshape(-1,1)
    prob3=prob3.reshape(-1,1)
    prob4=prob4.reshape(-1,1)
    prob_net=np.column_stack([prob0,prob1,prob2,prob3,prob4])

    ypred=np.zeros((len(x_predict)))
    for i in range(len(x_predict)):
      ypred[i]=np.argmax(prob_net[i,:])
    return ypred,prob_net

ypred_train_2b,prob_net_train=predict_GMM_FULL_COVARIANCE_FOR_2B_DATA
(1,23,xorig_2b,x_coast2b,x_highway2b,x_insidecity2b,x_street2b,x_tall
building2b)   # No. of gaussians=plug diff. values for testing , X
dimension=23, TRAINING SET
ypred_val_2b,prob_net_val=predict_GMM_FULL_COVARIANCE_FOR_2B_DATA(1,2
3,xval_2b,x_coast2b,x_highway2b,x_insidecity2b,x_street2b,x_tallbuild
ing2b)      # No. of gaussians=plug diff. values for testing , X
dimension=23, VALIDATION SET
ypred_test_2b,prob_net_test=predict_GMM_FULL_COVARIANCE_FOR_2B_DATA(1
,23,xtest_2b,x_coast2b,x_highway2b,x_insidecity2b,x_street2b,x_tallbu
ilding2b)      # No. of gaussians=plug diff. values for testing , X
dimension=23, TEST SET
accuracy_train=round(accuracy_score(yorig_2b, ypred_train_2b),4)
accuracy_val=round(accuracy_score(yval_2b, ypred_val_2b),4)
accuracy_test=round(accuracy_score(ytest_2b, ypred_test_2b),4)
print('Train Accuracy: ',accuracy_train)
print('Validation Accuracy: ',accuracy_val)
print('Test Accuracy: ',accuracy_test)

#from sklearn.metrics import confusion_matrix
training=confusion_matrix(yorig_2b, ypred_train_2b)
val=confusion_matrix(yval_2b, ypred_val_2b)
test=confusion_matrix(ytest_2b, ypred_test_2b)
print('Confusion matrix for Training Data: ',training)
print('Confusion matrix for Validation Data: ',val)
print('Confusion matrix for Test Data: ', test)
```

```
"""# Model5: GMM_Diagonal_covariance for data 2B"""

def
predict_GMM_DIAGONAL_COVARIANCE_FOR_2B_DATA(k,d,x_predict,x_coast,x_h
ighway,x_insidecity,x_street,x_tallbuilding):      # PREDCITING ON
DATASET 2A
    x_coast=x_coast.reshape((int(x_coast.shape[0]*36)),23)
    x_highway=x_highway.reshape((int(x_highway.shape[0]*36)),23)

x_insidecity=x_insidecity.reshape((int(x_insidecity.shape[0]*36)),23)
    x_street=x_street.reshape((int(x_street.shape[0]*36)),23)

x_tallbuilding=x_tallbuilding.reshape((int(x_tallbuilding.shape[0]*36
)),23)


wstar0,ustar0,cstar0=final_optimization_diagonal_covariance(k,d,x_coa
st)

wstar1,ustar1,cstar1=final_optimization_diagonal_covariance(k,d,x_hig
hway)

wstar2,ustar2,cstar2=final_optimization_diagonal_covariance(k,d,x_ins
idecity)

wstar3,ustar3,cstar3=final_optimization_diagonal_covariance(k,d,x_str
eet)

wstar4,ustar4,cstar4=final_optimization_diagonal_covariance(k,d,x_tal
lbuilding)

    prob0=per_class_GMM_covariance_2b(wstar0,ustar0,cstar0,x_predict)
    prob1=per_class_GMM_covariance_2b(wstar1,ustar1,cstar1,x_predict)
    prob2=per_class_GMM_covariance_2b(wstar2,ustar2,cstar2,x_predict)
    prob3=per_class_GMM_covariance_2b(wstar3,ustar3,cstar3,x_predict)
    prob4=per_class_GMM_covariance_2b(wstar4,ustar4,cstar4,x_predict)

    prob0=prob0.reshape(-1,1)
    prob1=prob1.reshape(-1,1)
```

```
    prob2=prob2.reshape(-1,1)
    prob3=prob3.reshape(-1,1)
    prob4=prob4.reshape(-1,1)
    prob_net=np.column_stack([prob0,prob1,prob2,prob3,prob4])

    ypred=np.zeros((len(x_predict)))
    for i in range(len(x_predict)):
      ypred[i]=np.argmax(prob_net[i,:])
    return ypred,prob_net

ypred_train_2b,prob_net_train=predict_GMM_DIAGONAL_COVARIANCE_FOR_2B_
DATA(2,23,xorig_2b,x_coast2b,x_highway2b,x_insidecity2b,x_street2b,x_
tallbuilding2b)   # No. of gaussians=plug diff. values for testing ,
X dimension=23, TRAINING SET
ypred_val_2b,prob_net_val=predict_GMM_DIAGONAL_COVARIANCE_FOR_2B_DATA
(2,23,xval_2b,x_coast2b,x_highway2b,x_insidecity2b,x_street2b,x_tallb
uilding2b)      # No. of gaussians=plug diff. values for testing , X
dimension=23, VALIDATION SET
ypred_test_2b,prob_net_test=predict_GMM_DIAGONAL_COVARIANCE_FOR_2B_DA
TA(2,23,xtest_2b,x_coast2b,x_highway2b,x_insidecity2b,x_street2b,x_ta
llbuilding2b)      # No. of gaussians=plug diff. values for testing ,
X dimension=23, TEST SET
accuracy_train=round(accuracy_score(yorig_2b, ypred_train_2b),4)
accuracy_val=round(accuracy_score(yval_2b, ypred_val_2b),4)
accuracy_test=round(accuracy_score(ytest_2b, ypred_test_2b),4)
print('Train Accuracy: ',accuracy_train)
print('Validation Accuracy: ',accuracy_val)
print('Test Accuracy: ',accuracy_test)
training=confusion_matrix(yorig_2b, ypred_train_2b)
val=confusion_matrix(yval_2b, ypred_val_2b)
test=confusion_matrix(ytest_2b, ypred_test_2b)
print('Confusion matrix for Training Data: ',training)
print('Confusion matrix for Validation Data: ',val)
print('Confusion matrix for Test Data: ', test)
```

Task 3 : KNN Classifier

```
# -*- coding: utf-8 -*-
"""KNN.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1DuNJPEwKOdcG5YGBd61tuZFEYCKF
886Y
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

columns = ['x1','x2','y']
df1 = pd.read_csv('1B_train.csv',header = None,names = columns)
df2 = pd.read_csv('1B_dev.csv',header = None,names = columns)


df1.coloumns = ['x1','x2','y']

def normalise_data(x):
        df_min_max_scaled = x.copy()

        # apply normalization techniques by Column 1
        column = ['x1' , 'x2']
        df_min_max_scaled[column] = (df_min_max_scaled[column] -
df_min_max_scaled[column].min()) / (df_min_max_scaled[column].max() -
df_min_max_scaled[column].min())

        # view normalized data
        return df_min_max_scaled
df1 = normalise_data(df1)
df2 = normalise_data(df2)
x_train = df1.iloc[:,0:2].values
y_train = df1.iloc[:,-1].values
```

```python
def train_test_split(dataset , test_size = 0.5):

    n_test = int(len(dataset)*test_size)
    test_set = dataset.sample(n_test)
    train_set = []
    for ind in dataset.index:
        if ind in test_set.index:
          continue
        train_set.append(dataset.iloc[ind])

    train_set = pd.DataFrame(train_set).astype(float).values.tolist()
    test_set = test_set.astype(float).values.tolist()

    return train_set, test_set

val_set , test_set = train_test_split(df2)
val_set = pd.DataFrame(val_set)
test_set = pd.DataFrame(test_set)

x_val = val_set.iloc[:,0:2].values
y_val = val_set.iloc[:,-1].values
x_test = test_set.iloc[:,0:2].values
y_test = test_set.iloc[:,-1].values

def train(X_train, y_train):

    return

from collections import Counter
def predict(X_train, y_train, x_test, k):

    distances = []
    Y = []

    for i in range(len(X_train)):

        distances.append([np.sqrt(np.sum(np.square(x_test -
X_train[i, :]))), i])
```

```python
    distances = sorted(distances)


    for i in range(k):
        index = distances[i][1]
        Y.append(y_train[index])

    return Counter(Y).most_common(1)[0][0]

def k_nearest_neighbor(X_train, y_train, X_test, k):

    train(X_train, y_train)


    Targets = []
    for i in range(len(X_test)):
        Targets.append(predict(X_train, y_train, X_test[i, :], k))

    return np.asarray(Targets)

Kne = k_nearest_neighbor(x_train,y_train,x_test,1)
Kne

y_pred = Kne

y_pred

def getAccuracy(actual,predicted):
      correct = 0
      for i in range(len(actual)):
            if actual[i] == predicted[i]:
                  correct += 1
      return (correct / float(len(actual))) * 100.00

Eval = getAccuracy(y_test , y_pred)
Eval

def create_conf_matrix(expected, predicted, n_classes):
```

```
    expected = expected.astype(int)
    predicted = predicted.astype(int)
    m = [[0] * n_classes for i in range(n_classes)]
    for pred, exp in zip(predicted, expected):
        m[pred][exp] += 1
    return m

CM = create_conf_matrix(y_test,y_pred,4)
CM

def Decision_Boundary_Regions(X, y, classifier, test_idx=None,
resolution=0.02):

   markers = ('s', 'x', 'o', '^')
   colors = ('red', 'blue', 'lightgreen', 'gray')

   cmap = ListedColormap(colors[:len(np.unique(y))])

   x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1



   x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
   xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),

   np.arange(x2_min, x2_max, resolution))
   D = k_nearest_neighbor(X,y,np.array([xx1.ravel(),
xx2.ravel()]).T,3)

   D = D.reshape(xx1.shape)
   plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)


   X_test, y_test = X[test_idx, :], y[test_idx]
   for idx, cl in enumerate(np.unique(y)):
      plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
               alpha=0.8, c=cmap(idx),
               marker=markers[idx], label=cl)
```

```
    if test_idx:
        X_test, y_test = X[test_idx, :], y[test_idx]
        plt.scatter(X_test[:, 0], X_test[:, 1], c='',
                    alpha=1.0, linewidth=1, marker='o',
                    s=55)



Decision_Boundary_Regions(x_train, y_train,
                      classifier=k_nearest_neighbor,
test_idx=range(105,150))
plt.xlim(-0.25, 1.25)
plt.ylim(-0.25, 1.25)

plt.xlabel("Variable x1")
plt.ylabel("Variable x2")
plt.title("Decision Boundary plot(k = 7)")
plt.legend(loc='upper left')
plt.show()
```