ASSIGNMENT 2

CS5691 Pattern Recognition and Machine Learning

CS5691 Assignment Code 2

Team Members:

BE17B007	N Sowmya Manojna
PH17B010	Thakkar Riya Anandbhai
PH17B011	Chaithanya Krishna Moorthy

Indian Institute of Technology, Madras

Contents

All codes excluding the modules are converted to .py files from IPython Notebooks

1 Dataset 1A

The code written for analyzing Dataset 1A is as follows:

```
#!/usr/bin/env python
# coding: utf-8
#########################################################################
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
get_ipython().run_line_magic('matplotlib', 'inline')

#########################################################################
from sklearn.metrics import confusion_matrix

#########################################################################
from sklearn.metrics import classification_report


# # Dataset 1a:
# ### Importing the train, test and cross validation data sets
#########################################################################
col_names=["x1","x2","y"]

#########################################################################
## Train data
data1a=pd.read_csv("train.csv",names=col_names)

#########################################################################
data1a.head()

#########################################################################
data1a.isnull().sum()

#########################################################################
data1a.describe()

#########################################################################
## Splitting the columns of train data

X1train=data1a["x1"]
X2train=data1a["x2"]
Ytrain=np.array(data1a["y"])
Xtrain=np.array(data1a.drop("y",axis=1))

#########################################################################
## group labels
data1a["y"].unique()

#########################################################################
## Importing the test and cross-validation data
data1a_dev=pd.read_csv("dev.csv",names=col_names)

#########################################################################
## Function to split a given dataset into test and cross-validation

def create_datasets(data,cv_size):
    data.sample(frac=1).reset_index(drop=True)
    data_cv=data[0:cv_size]
    data_test=data[cv_size:]
    return(data_cv,data_test)

#########################################################################
def euclidean(p1,p2):
    d=np.linalg.norm(np.array(p1)-np.array(p2))
    return d

########################################################################
def accuracy(y_pred,y_actual):
    true_count=0
    for i in range(len(y_pred)):
        if y_pred[i]==y_actual[i]:
            true_count+=1;
    return(true_count/len(y_pred))

########################################################################
data1a_dev.shape

########################################################################
## Splitting in the ratio 70:30 (cv:test)
data1a_cv,data1a_test=create_datasets(data1a_dev,84)


# ### Plotting the train data set
########################################################################
X_cv=np.array(data1a_cv.drop("y",axis=1))
Y_cv=np.array(data1a_cv["y"])
X_test=np.array(data1a_test.drop("y",axis=1))
Y_test=np.array(data1a_test["y"])

plt.figure()
plt.scatter(X1train[Ytrain==0],X2train[Ytrain==0],label="y=0")
plt.scatter(X1train[Ytrain==1],X2train[Ytrain==1],label="y=1")
plt.scatter(X1train[Ytrain==2],X2train[Ytrain==2],label="y=2")
plt.scatter(X1train[Ytrain==3],X2train[Ytrain==3],label="y=3")
plt.legend()
plt.xlabel("X1")
plt.ylabel("X2")
plt.title("Scatter plot of data 1a")
plt.savefig("Scatter plot of data_1a.jpg")
plt.show()


# # K Nearest Neighbour Classifier for dataset 1a:
########################################################################
def knn(x,y,test,k):
    distances=[]
    for i in range(len(x)):
        d=euclidean(x[i],test)
        l=(d,x[i],y[i])
        distances.append(l)
    distances.sort(key = lambda x:x[0])
    count=Counter()
    for i in distances[:k]:
        count[i[2]]+=1
    pred=count.most_common(1)[0][0]
    return(distances[:k],pred)



# ### KNN on given cross-validation and test datasets:
########################################################################
k_list=[1,7,15]
Accuracy_cv=[]
Accuracy_train=[]
Accuracy_test=[]

########################################################################
## iterating over k-values
for i in k_list:
    ycv_pred=[]
    for j in X_cv:
        ycv_pred.append(knn(Xtrain,Ytrain,j,i)[1])
    ytest_pred=[]
    for j in X_test:
        ytest_pred.append(knn(Xtrain,Ytrain,j,i)[1])
    ytrain_pred=[]
    for j in Xtrain:
        ytrain_pred.append(knn(Xtrain,Ytrain,j,i)[1])
    Accuracy_cv.append(accuracy(Y_cv,ycv_pred))
    Accuracy_test.append(accuracy(Y_test,ytest_pred))
    Accuracy_train.append(accuracy(Ytrain,ytrain_pred))

#########################################################################
accuracy_table_knn=pd.DataFrame(list(zip(k_list,Accuracy_train,Accuracy_cv,...
    Accuracy_test)),columns=["k-value", "Accuracy train","Accuracy CV","Accuracy ...
    test"])

#########################################################################
accuracy_table_knn

#########################################################################
cm=confusion_matrix(Ytrain,ytrain_pred,labels=[1.0,3.0,0.0,2.0])
cm2=confusion_matrix(Y_test,ytest_pred)

#########################################################################
from sklearn.metrics import ConfusionMatrixDisplay

#########################################################################
cmd=ConfusionMatrixDisplay(cm,display_labels=[0.0,1.0,2.0,3.0])
plt.figure()
cmd.plot()
plt.savefig("1a_cm_knn_train.jpg")

#########################################################################
cmd2=ConfusionMatrixDisplay(cm2,display_labels=[0.0,1.0,2.0,3.0])
plt.figure()
cmd2.plot()
plt.savefig("1a_cm_knn_test.jpg")


# # Naive Bayes Classifier:
#########################################################################
def seperate_by_classval(data):
    ## the target variable must be stored in a column named "y"
    class_vals=list(data["y"].unique())
    seperated=dict()
    features=data.drop('y',axis=1)
    Y=np.array(data["y"])
    ## creates a key value corresponding to each class label
    for i in class_vals:
        seperated[i]=features[Y==i];
    return(seperated)

#########################################################################
def priori(data):
    seperated_data=seperate_by_classval(data)
    probs=dict()
    for i in seperated_data.keys():
        probs[i]=len(seperated_data[i])/len(data);
    return probs

#########################################################################
def mu_sigma(data):
    seperated_data=seperate_by_classval(data)
    mean=dict()
    sigma={}
    for i in list(seperated_data.keys()):
        features=seperated_data[i]
        mean[i]=[]
        sigma[i]=[]
        for j in range(seperated_data[i].shape[1]):
            mean[i].append(np.mean(features.iloc[:,j]))
            sigma[i].append(np.std(features.iloc[:,j]))
    return(mean,sigma)

########################################################################
def gauss_val(x,cov_matrix,mean):
    x=np.array(x)
    A=(x-mean)
    B=np.linalg.inv(cov_matrix)
    C=np.transpose(A)
    det=np.linalg.det(cov_matrix)
    AB=A.dot(B)
    m=AB.dot(C)

    exp_term=np.exp(-m/2)
    d=2
    return (exp_term/(2*np.pi*det**0.5))


# ## Seperating the data according to class label:
########################################################################
seperated_data=seperate_by_classval(data1a)

########################################################################
### Labels:
labels=list(data1a["y"].unique())

########################################################################
########################################################################
accuracy_table_bayes=pd.DataFrame()ccuracy_table_bayes["method"]=["Ci=Cj=sigma**2*I","Ci=Cj=C","Ci!=Cj"]

########################################################################
accuracy_table_bayes["Train Accuracy"]=[0,0,0]
accuracy_table_bayes["CV accuracy"]=[0,0,0]
accuracy_table_bayes["Test Accuracy"]=[0,0,0]


# ### Case 1: Ci=Cj=sigma**2 * I
########################################################################
sigma=mu_sigma(data1a)[1]

########################################################################
sigma

########################################################################
var=0
for i in labels:
    var+=sigma[i][0]**2+sigma[i][1]**2

var=var/(4*2)

########################################################################
def predictor1(x):
    pyi_x={}
    pyi=priori(data1a)
    means=mu_sigma(data1a)[0]
    for i in labels:
        pyi_x[i]=pyi[i]*gauss_val(x,var*np.eye(2),means[i])
    val=sum(pyi_x.values())
    p=0
    for i in labels:
        pyi_x[i]/=val
        if pyi_x[i]>p:
            prediction=i
            p=pyi_x[i]


    return(pyi_x,prediction)
```

```

##########################################################################
predictor1([-10,5])

##########################################################################
Y_nb1_cv=[]
Y_nb1_test=[]
Y_nb1_train=[]
for i in range(len(X_cv)):
    Y_nb1_cv.append(predictor1(X_cv[i])[1])
for i in range(len(X_test)):
    Y_nb1_test.append(predictor1(X_test[i])[1])
for i in range(len(Xtrain)):
    Y_nb1_train.append(predictor1(Xtrain[i])[1])


##########################################################################
accuracy_table_bayes.iloc[0,1:]=[accuracy(Y_nb1_train,Ytrain),accuracy(Y_nb1_cv,...
    Y_cv),accuracy(Y_nb1_test,Y_test)]


# ### Confusion Matrix
##########################################################################
cm_nb_train=confusion_matrix(Y_nb1_train,Ytrain)
cm_nb_test=confusion_matrix(Y_nb1_test,Y_test)

##########################################################################
len(Y_nb1_train)

##########################################################################
cmd_nb_train=ConfusionMatrixDisplay(cm_nb_train,display_labels=[0.0,1.0,2.0,3.0])
plt.figure()
cmd_nb_train.plot()
plt.savefig("1a_cm_nb_train.jpg")

##########################################################################
cmd_nb_test=ConfusionMatrixDisplay(cm_nb_test,display_labels=[0.0,1.0,2.0,3.0])
plt.figure()
cmd_nb_test.plot()
plt.savefig("1a_cm_nb_test.jpg")


# ### Case 2: Covariance matrix is same for all the classes:
##########################################################################
cov_matrix={}
for i in labels:
    cov_matrix[i]=np.cov(seperated_data[i],rowvar=False)

##########################################################################
cov_matrix

##########################################################################
C=np.zeros((2,2))
for i in labels:
    C+=cov_matrix[i]
C/=4

##########################################################################
C

##########################################################################
def predictor2(x):
    pyi_x={}
    pyi=priori(data1a)
    means=mu_sigma(data1a)[0]
    for i in labels:
        pyi_x[i]=pyi[i]*gauss_val(x,C,means[i])
    val=sum(pyi_x.values())
    p=0
    for i in labels:
        pyi_x[i]/=val
        if pyi_x[i]>p:
            prediction=i
            p=pyi_x[i]


    return(pyi_x,prediction)

#########################################################################
Y_nb2_cv=[]
Y_nb2_test=[]
Y_nb2_train=[]
for i in range(len(X_cv)):
    Y_nb2_cv.append(predictor2(X_cv[i])[1])
for i in range(len(X_test)):
    Y_nb2_test.append(predictor2(X_test[i])[1])
for i in range(len(Xtrain)):
    Y_nb2_train.append(predictor2(Xtrain[i])[1])


#########################################################################
accuracy_table_bayes.iloc[1,1:]=[accuracy(Y_nb2_train,Ytrain),accuracy(Y_nb2_cv,...
    Y_cv),accuracy(Y_nb2_test,Y_test)]


# ###  Case 3: Covariance matrix is different for all the classes:
#########################################################################
def predictor3(x):
    pyi_x={}
    pyi=priori(data1a)
    means=mu_sigma(data1a)[0]
    for i in labels:
        pyi_x[i]=pyi[i]*gauss_val(x,cov_matrix[i],means[i])
    val=sum(pyi_x.values())
    p=0
    for i in labels:
        pyi_x[i]/=val
        if pyi_x[i]>p:
            prediction=i
            p=pyi_x[i]


    return(pyi_x,prediction)


#########################################################################
predictor3([5,5])

#########################################################################
Y_nb3_cv=[]
Y_nb3_test=[]
Y_nb3_train=[]
for i in range(len(X_cv)):
    Y_nb3_cv.append(predictor3(X_cv[i])[1])
for i in range(len(X_test)):
    Y_nb3_test.append(predictor3(X_test[i])[1])
for i in range(len(Xtrain)):
    Y_nb3_train.append(predictor3(Xtrain[i])[1])


#########################################################################
accuracy_table_bayes.iloc[2,1:]=[accuracy(Y_nb3_train,Ytrain),accuracy(Y_nb3_cv,...
    Y_cv),accuracy(Y_nb3_test,Y_test)]

#########################################################################
accuracy_table_bayes


```

```
# ### Confusion matrix for naive bayes classifier:
######################################################################



# ### Decision boundary plot for knn:
######################################################################
min1,max1=data1a["x1"].min()-1,data1a["x1"].max()+1
min2,max2=data1a["x2"].min()-1,data1a["x2"].max()+1

######################################################################
resolution=0.5
x1grid=np.arange(min1,max1,resolution)
x2grid=np.arange(min2,max2,resolution)

######################################################################
xx,yy=np.meshgrid(x1grid,x2grid)

######################################################################
r1,r2=xx.flatten(),yy.flatten()
r1,r2=r1.reshape((len(r1),1)),r2.reshape((len(r2),1))

######################################################################
grid=np.hstack((r1,r2))

######################################################################
yhat_knn_1=[]
for i in range(len(grid)):
    yhat_knn_1.append(knn(Xtrain,Ytrain,grid[i,:],1)[1])

######################################################################
len(grid)

######################################################################
yhat_knn_1=np.array(yhat_knn_1)

######################################################################
zz=yhat_knn_1.reshape(xx.shape)

######################################################################
data1a["y"].unique()

######################################################################
plt.figure()
plt.contourf(xx,yy,zz,alpha=0.5,cmap="Paired")
plt.scatter(X1train[Ytrain==0],X2train[Ytrain==0],label="y=0",c="Blue")
plt.scatter(X1train[Ytrain==1],X2train[Ytrain==1],label="y=1",c="Green")
plt.scatter(X1train[Ytrain==2],X2train[Ytrain==2],label="y=2",c="Orange")
plt.scatter(X1train[Ytrain==3],X2train[Ytrain==3],label="y=3",c='red')
plt.legend()
plt.xlabel("X1")
plt.ylabel("X2")
plt.title("Decision region plot of data 1a, knn classifier")
plt.savefig("1a_knn_decision_region.jpg")
plt.show()

######################################################################
grid

######################################################################
yhat_nb=[]
for i in range(len(grid)):
    yhat_nb.append(predictor1(grid[i,:])[1])

######################################################################
yhat_nb=np.array(yhat_nb)

######################################################################
zz_nb=yhat_nb.reshape(xx.shape)
```

```

#######################################################################
plt.figure()
plt.contourf(xx,yy,zz_nb,alpha=0.5,cmap="Paired")
plt.scatter(X1train[Ytrain==0],X2train[Ytrain==0],label="y=0",c="Blue")
plt.scatter(X1train[Ytrain==1],X2train[Ytrain==1],label="y=1",c="Green")
plt.scatter(X1train[Ytrain==2],X2train[Ytrain==2],label="y=2",c="Orange")
plt.scatter(X1train[Ytrain==3],X2train[Ytrain==3],label="y=3",c='red')
plt.legend()
plt.xlabel("X1")
plt.ylabel("X2")
plt.title("Decision region plot of data 1a,naive-bayes classifier")
plt.savefig("1a_nb_case1_decisionregion.jpg")
plt.show()
```

2 Dataset 1B

2.1 Bayes Classification, GMM, Full Covariance

The GMM full covariance model code is as follows:

```
#!/usr/bin/env python
# coding: utf-8
#######################################################################
import time
import pickle
import numpy as np
import pandas as pd
from gmm import GMM
import matplotlib.pyplot as plt
from multiprocessing import Pool
from collections import defaultdict
from scipy.stats import multivariate_normal as mvn
from sklearn.model_selection import train_test_split

plt.rcParams["font.size"] = 18
plt.rcParams["axes.grid"] = True
plt.rcParams["figure.figsize"] = 8,6
plt.rcParams['font.serif'] = "Cambria"
plt.rcParams['font.family'] = "serif"

get_ipython().run_line_magic('load_ext', 'autoreload')
get_ipython().run_line_magic('autoreload', '2')

#######################################################################
df = pd.read_csv("../datasets/1B/train.csv", header=None)
X = df.drop(2, axis=1).to_numpy()
df.head()

#######################################################################
classes = np.unique(df[2])
gmm_list = defaultdict(list)
q_list = list(range(2,10))

for i in classes:
    df_select = df[df[2]==i]
    X_select = df_select.drop(2, axis=1).to_numpy()
    for q in q_list:
        gmm = GMM(q=q)
        gmm.fit(X_select)
        gmm_list[i].append(gmm)

#######################################################################
import pickle
fin = open("1b_gmm_results", "wb")
pickle.dump(gmm_list, fin)
```

```python
fin.close()

########################################################################
df_test = pd.read_csv("../datasets/1B/dev.csv", header=None)
df_cv = df_test.sample(frac=0.7)
X_cv = df_cv.drop(2, axis=1).to_numpy()
display(df_cv.head())
df_test = df_test.drop(df_cv.index)
X_test = df_test.drop(2, axis=1).to_numpy()
df_test.head()

########################################################################
classes = np.unique(df[2])
q_list = list(range(2,10))

accuracy_list = []
cv_accuracy_list = []
test_accuracy_list = []
for i in range(len(q_list)):
    gmm0 = gmm_list[0.0][i]
    gmm1 = gmm_list[1.0][i]
    gmm2 = gmm_list[2.0][i]

    # Training
    a = gmm0.indv_log_likelihood(X)
    b = gmm1.indv_log_likelihood(X)
    c = gmm2.indv_log_likelihood(X)

    d = np.hstack((a, b, c))
    pred = np.argmax(d, axis=1)
    accuracy_list.append(np.sum(pred == df[2])/df[2].size)

    # CV
    a = gmm0.indv_log_likelihood(X_cv)
    b = gmm1.indv_log_likelihood(X_cv)
    c = gmm2.indv_log_likelihood(X_cv)

    d = np.hstack((a, b, c))
    pred = np.argmax(d, axis=1)
    cv_accuracy_list.append(np.sum(pred == df_cv[2])/df_cv[2].size)

    # Testing
    a = gmm0.indv_log_likelihood(X_test)
    b = gmm1.indv_log_likelihood(X_test)
    c = gmm2.indv_log_likelihood(X_test)

    d = np.hstack((a, b, c))
    pred = np.argmax(d, axis=1)
    test_accuracy_list.append(np.sum(pred == df_test[2])/df_test[2].size)

########################################################################
plt.plot(q_list, accuracy_list, '.-')
plt.title("Accuracy across varying Q")
plt.xlabel("Q for each class")
plt.ylabel("Accuracy")
plt.show()

plt.plot(q_list, cv_accuracy_list, '.-')
plt.title("CV Accuracy across varying Q")
plt.xlabel("Q for each class")
plt.ylabel("Accuracy")
plt.show()

plt.plot(q_list, test_accuracy_list, '.-')
plt.title("Test Accuracy across varying Q")
plt.xlabel("Q for each class")
plt.ylabel("Accuracy")
plt.show()

```

```
###########################################################################
fout = open("1b_gmm_results", "rb")
gmm_list = pickle.load(fout)
fout.close()

###########################################################################
x, y = np.mgrid[-3:3:30j, -3:3:30j]
xy = np.column_stack([x.flat, y.flat])

z0_val = gmm_list[0.0][3].indv_log_likelihood(xy)
z1_val = gmm_list[1.0][3].indv_log_likelihood(xy)
z2_val = gmm_list[2.0][3].indv_log_likelihood(xy)

d = np.hstack((z0_val, z1_val, z2_val))
classes = np.argmax(d, axis=1)
classes = classes.reshape(x.shape)

plt.figure()
df.plot.scatter(0, 1, c=[color_list[int(i)] for i in df[2]], alpha=1)
plt.contourf(x, y, classes, 2, colors=color_list, alpha=0.1)
plt.contour(x, y, classes, 2, colors=color_list, alpha=1)
plt.title("Decision Boundaries - Full Covariance")
plt.show()

###########################################################################
classes = np.unique(df[2])
q_list = list(range(2,10))

# color_list = np.random.rand(len(classes), 3)
color_list = ["springgreen", "mediumturquoise", "palevioletred"]
x, y = np.mgrid[-3:3:30j, -3:3:30j]
xy = np.column_stack([x.flat, y.flat])

z0 = gmm_list[0.0][3].gaussian_val(xy)
z0 = z0.reshape(x.shape)

z1 = gmm_list[1.0][3].gaussian_val(xy)
z1 = z1.reshape(x.shape)

z2 = gmm_list[2.0][3].gaussian_val(xy)
z2 = z2.reshape(x.shape)

plt.figure()
df.plot.scatter(0, 1, c=[color_list[int(i)] for i in df[2]], alpha=1)
plt.contour(x, y, z0, levels=np.logspace(-2,2,20), colors=color_list[0])
plt.contour(x, y, z1, levels=np.logspace(-2,2,20), colors=color_list[1])
plt.contour(x, y, z2, levels=np.logspace(-2,2,20), colors=color_list[2])
plt.title("Contour Plot - Full Covariance")

###########################################################################
x, y = np.mgrid[-3:3:30j, -3:3:30j]
xy = np.column_stack([x.flat, y.flat])

z0_val = gmm_list[0.0][3].indv_log_likelihood(xy)
z1_val = gmm_list[1.0][3].indv_log_likelihood(xy)
z2_val = gmm_list[2.0][3].indv_log_likelihood(xy)

d = np.hstack((z0_val, z1_val, z2_val))
classes = np.argmax(d, axis=1)
classes = classes.reshape(x.shape)

plt.figure()
df.plot.scatter(0, 1, c=[color_list[int(i)] for i in df[2]], alpha=1)
plt.contourf(x, y, classes, 2, colors=color_list, alpha=0.1)
plt.contour(x, y, classes, 2, colors=color_list, alpha=1)
plt.title("Decision Boundaries - Full Covariance")
plt.show()

###########################################################################
```

```
x, y = np.mgrid[-3:3:30j, -3:3:30j]
xy = np.column_stack([x.flat, y.flat])

z0_val = gmm_list[0.0][3].indv_log_likelihood(xy)
z1_val = gmm_list[1.0][3].indv_log_likelihood(xy)
z2_val = gmm_list[2.0][3].indv_log_likelihood(xy)

d = np.hstack((z0_val, z1_val, z2_val))
classes = np.argmax(d, axis=1)
classes = classes.reshape(x.shape)

plt.figure()
df.plot.scatter(0, 1, c=[color_list[int(i)] for i in df[2]], alpha=1)
plt.contourf(x, y, classes, 2, colors=color_list, alpha=0.1)
plt.contour(x, y, classes, 2, colors=color_list, alpha=1)
plt.contour(x, y, z0, levels=np.logspace(-2,2,20), colors=color_list[0])
plt.contour(x, y, z1, levels=np.logspace(-2,2,20), colors=color_list[1])
plt.contour(x, y, z2, levels=np.logspace(-2,2,20), colors=color_list[2])
plt.title("Decision Boundaries + Contours - Full Covariance")
plt.show()

##########################################################################
import seaborn as sns
from sklearn.metrics import confusion_matrix

classes = np.unique(df[2])
q_list = list(range(2,10))

gmm0 = gmm_list[0.0][3]
gmm1 = gmm_list[1.0][3]
gmm2 = gmm_list[2.0][3]

# Training
a = gmm0.indv_log_likelihood(X)
b = gmm1.indv_log_likelihood(X)
c = gmm2.indv_log_likelihood(X)

d = np.hstack((a, b, c))
pred = np.argmax(d, axis=1)
conf_mat = confusion_matrix(pred, df[2])
plt.figure()
sns.heatmap(conf_mat, annot=True)
plt.title("Training Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("Actual Class")
plt.show()

# CV
a = gmm0.indv_log_likelihood(X_cv)
b = gmm1.indv_log_likelihood(X_cv)
c = gmm2.indv_log_likelihood(X_cv)

d = np.hstack((a, b, c))
pred = np.argmax(d, axis=1)
conf_mat = confusion_matrix(pred, df_cv[2])
plt.figure()
sns.heatmap(conf_mat, annot=True)
plt.title("CV Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("Actual Class")
plt.show()

# Testing
a = gmm0.indv_log_likelihood(X_test)
b = gmm1.indv_log_likelihood(X_test)
c = gmm2.indv_log_likelihood(X_test)

d = np.hstack((a, b, c))
pred = np.argmax(d, axis=1)
```

```
conf_mat = confusion_matrix(pred, df_test[2])
plt.figure()
sns.heatmap(conf_mat, annot=True)
plt.title("Testing Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("Actual Class")
plt.show()
```

The GMM class module is as follows:

```
import numpy as np
from tqdm import tqdm
from sklearn.cluster import KMeans
from scipy.stats import multivariate_normal as mvn
import pandas as pd

class GMM():
    def __init__(self, q):
        self.q = q

    def fit(self, X, covariance_type="diag", tol=1e-5):
        """
        X: n*d
        mu: q*d
        C: q*d*d
        gamma: n*q
        """
        self.n, self.d = X.shape
        self.X = X
        self.covariance_type = covariance_type
        self.initialization()
        self.lglk_list = []
        for i in tqdm(range(100)):
            self.lglk_list.append(self.log_likelihood(self.X))
            self.expectation()
            self.maximization()
            new_lk = self.log_likelihood(self.X)
            diff = new_lk - self.lglk_list[-1]
            if  diff < tol:
                if diff < 0: print("Difference is less than 0")
                break


    def initialization(self):
        # kmeans = KMeans(n_clusters=self.q, random_state=0).fit(self.X)
        kmeans = KMeans(n_clusters=self.q).fit(self.X)
        labels = kmeans.labels_
        unique, counts = np.unique(labels, return_counts=True)

        self.subcomponents = unique.size
        self.gamma = np.eye(self.subcomponents)[labels]
        self.Nq = np.sum(self.gamma, axis=0)
        self.weights = counts/self.n
        self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
        self.C = np.zeros((self.subcomponents, self.d, self.d))

        for i in range(self.q):
            self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self....
                mu[i,:])).T@(self.X-self.mu[i,:])

            if self.covariance_type == "diag":
                self.C[i] = np.diag(self.C[i])


    def expectation(self):
        self.gamma = np.zeros((self.n, self.q))

        for i in range(self.q):
            try:
```

```python
                self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self.C[i])
            except:
                self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self.C[i]+np.eye(self.C[i].shape[0])*1e-7)
        self.gamma = self.gamma/np.sum(self.gamma, axis=1).reshape(-1,1)

    def maximization(self):
        # print(np.sum(self.weights))
        self.Nq = np.sum(self.gamma, axis=0)
        self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)

        for i in range(self.q):
            self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self.mu[i,:])).T@(self.X-self.mu[i,:])

            if self.covariance_type == "diag":
                self.C[i] = np.diag(self.C[i])

        self.weights = self.Nq/self.n

    def log_likelihood(self, X_test):
        lk = 0
        n, d = X_test.shape
        for i in range(n):
            val = 0
            for j in range(self.q):
                try:
                    val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j])
                except:
                    val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j]+np.eye(self.C[j].shape[0])*1e-7)
            lk += np.log(val)

        return lk

    def indv_log_likelihood(self, X_test):
        n, d = X_test.shape
        lk = np.zeros((X_test.shape[0], 1))
        for i in range(n):
            val = 0
            for j in range(self.q):
                try:
                    val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j])
                except:
                    val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j]+np.eye(self.C[j].shape[0])*1e-7)
            lk[i] = np.log(val)

        return lk

    def gaussian_val(self, X_test):
        n, d = X_test.shape
        val = np.zeros((n, self.q))

        for i in range(self.q):
            val[:,i] = self.weights[i]*mvn.pdf(X_test, self.mu[i], self.C[i])

        return np.sum(val, axis=1)

class GMM_vl():
    def __init__(self, q):
        self.q = q

    def fit(self, X, epochs=100, covariance_type="diag", tol=1e-5):
        """
        X: n*d
        mu: q*d
        C: q*d*d
        gamma: n*q
        """
        self.n, self.d = X.shape
        self.X = X
        self.epochs = epochs
        self.covariance_type = covariance_type
        self.initialization()
        self.lglk_list = []
        for i in tqdm(range(self.epochs)):
            self.lglk_list.append(self.log_likelihood(self.X))
            self.expectation()
            self.maximization()
            new_lk = self.log_likelihood(self.X)
            diff = new_lk - self.lglk_list[-1]
            if  diff < tol:
                if diff < 0:
                    print("Difference is less than 0")
                    break

    def initialization(self):
    # kmeans = KMeans(n_clusters=self.q, random_state=0).fit(self.X)
        kmeans = KMeans(n_clusters=self.q).fit(self.X)
        labels = kmeans.labels_
        unique, counts = np.unique(labels, return_counts=True)

        self.subcomponents = unique.size
        self.gamma = np.eye(self.subcomponents)[labels]
        self.Nq = np.sum(self.gamma, axis=0)
        self.weights = counts/self.n
        self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
        self.C = np.zeros((self.subcomponents, self.d, self.d))

        for i in range(self.q):
            self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self.mu[i,:])).T@(self.X-self.mu[i,:])

            if self.covariance_type == "diag":
                self.C[i] = np.diag(np.diag(self.C[i]))


    def expectation(self):
        self.gamma = np.zeros((self.n, self.q))

        for i in range(self.q):
            try:
                self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self.C[i])
            except:
                self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self.C[i]+np.eye(self.C[i].shape[0])*1e-3)
                self.gamma = self.gamma/np.sum(self.gamma, axis=1).reshape(-1,1)

    def maximization(self):
        # print(np.sum(self.weights))
        self.Nq = np.sum(self.gamma, axis=0)
        self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)

        for i in range(self.q):
            self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self.mu[i,:])).T@(self.X-self.mu[i,:])

        if self.covariance_type == "diag":
            self.C[i] = np.diag(np.diag(self.C[i]))

        self.weights = self.Nq/self.n

    def log_likelihood(self, X_test):
        lk = 0
        n, d = X_test.shape
        for i in range(n):
            val = 0
            for j in range(self.q):
                try:
                        val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self....
                            C[j])
                except:
                    val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j...
                        ]+np.eye(self.C[j].shape[0])*1e-3)
            lk += np.log(val)

        return lk

    def indv_log_likelihood(self, X_test):
        n, d = X_test.shape
        lk = np.zeros((X_test.shape[0], 1))
        for i in range(n):
            val = 0
            for j in range(self.q):
                try:
                        val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self....
                            C[j])
                except:
                        val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self....
                            C[j]+np.eye(self.C[j].shape[0])*1e-3)
            lk[i] = np.log(val)

        return lk

    def gaussian_val(self, X_test):
        n, d = X_test.shape
        val = np.zeros((n, self.q))

        for i in range(self.q):
            val[:,i] = self.weights[i]*mvn.pdf(X_test, self.mu[i], self.C[i])

        return np.sum(val, axis=1)

    def probab(self, df):
        df = pd.DataFrame(df)
        grouped_df = df.groupby(by=["class", "image"])
        for key, item in grouped_df:
            selected_df = grouped_df.get_group(key)
            X_select = selected_df.drop(["index", "image", "class"], axis=1)....
                to_numpy()
            val = self.gaussian_val(X_select)
        print(val.shape)
```

2.2 Bayes Classification, GMM, Diagonal Covariance

The GMM diagonal covariance model code is as follows:

```
#!/usr/bin/env python
# coding: utf-8
#########################################################################
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#########################################################################
import pickle
from scipy.stats import multivariate_normal as mvn

#########################################################################
plt.style.use('science')
plt.rcParams['font.size'] = 18
plt.rcParams['axes.grid'] = True
plt.rcParams["grid.linestyle"] = (5,9)
plt.rcParams['figure.figsize'] = 8,6

#########################################################################
import statistics as sts
from sklearn.model_selection import train_test_split

#########################################################################
from sklearn.cluster import KMeans

#########################################################################
from separate_class import Separate

#########################################################################
ds2_train = pd.read_csv("train2.csv", header = None)

#########################################################################
ds2_test = pd.read_csv("dev2.csv", header = None)

#########################################################################
ds2_train.head()

#########################################################################
ds2_test.head()

#########################################################################
ds2_train.describe()

#########################################################################
ds2_test.describe()

#########################################################################
X_train = ds2_train.iloc[:,:2]
Y_train = ds2_train.iloc[:,2]

#########################################################################
def gaus(x,m,c,d):
    return((1/(((2*np.pi)**(d/2))*np.sqrt(np.linalg.det(c))))*np.exp(-(x-m).T@np....
        linalg.inv(c)@(x-m)/2))

#########################################################################
sep_train = Separate(ds2_train)

#########################################################################
classses_dat = sep_train.classes

#########################################################################
X_sep_train = sep_train.get_x()
Y_sep_train = sep_train.get_y()
dat_sep_train = sep_train.get_separated_data()

#########################################################################
pd.DataFrame(X_sep_train[0]).to_csv("X_sep_train.csv")

#########################################################################
def likelihood(x,m,W,c):
    s = 0
    m = np.array(m)
    l = len(W)
    for i in range(l):
        s += W[i]*gaus(x,m[i],c[i],d)
    return(s)

#########################################################################
plt.rcParams["font.size"] = 18
plt.rcParams["axes.grid"] = True
plt.rcParams["figure.figsize"] = 8,6
plt.rcParams['font.serif'] = "Cambria"
plt.rcParams['font.family'] = "serif"

#########################################################################
import time

#########################################################################
from multiprocessing import Pool

#########################################################################
class_ = 2
d = 2
threshold = 0.01

#########################################################################
# parameter estimation for Bayesian GMM - EM method
# training and obtaining parameters for different hperparameter values

#for Q in q:
def f(Q):
    L_old = 0
    L_new = 1
    L = []
    difference = L_new - L_old
    cond = True
    # inititalization
    while (cond==True):
        kmeans = KMeans(n_clusters = Q, random_state = 0).fit(X_sep_train[class_])
        labels = kmeans.labels_
        N = np.array([])
        for i in range(Q):
            N = np.append(N,np.count_nonzero(labels==i))
        cond = True in (ele ==1 for ele in N)

    Nt = np.sum(N)
    w = N/Nt
    gamma = []
    for i in range(Q):
        gamma.append(np.multiply(labels==i,1))
    mu = kmeans.cluster_centers_
    n = len(X_sep_train[0])
    C = np.zeros((Q,d,d))
    for i in range(Q):
        for j in range(n):
            C[i] += gamma[i][j]*np.outer(X_sep_train[class_].iloc[j] - mu[i],...
                X_sep_train[class_].iloc[j] - mu[i])
        C[i] = np.diag(np.diag(C[i]/N[i]))

    L_old = 0
    for i in range(n):
        L_old += np.log(likelihood(X_sep_train[class_].iloc[i],mu,w,C))

    while (difference > threshold):


        #Expectation
        den = np.zeros(n)
        for i in range(n):
            for j in range(Q):
                den[i] += w[j]*gaus(X_sep_train[class_].iloc[i],np.array(mu)[j],C[j...
                    ],d)

        gamma = np.zeros((Q,n))
        for i in range(n):
            for j in range(Q):
                gamma[j][i] = w[j]*gaus(X_sep_train[class_].iloc[i], np.array(mu)[j...
                    ], C[j],d)/den[i]

        # maximization step
        N = []
        for i in range(Q):
            N.append(np.sum(gamma[i]))
        Nt = np.sum(N)
        w = N/Nt
        mu = np.divide(gamma@X_sep_train[class_],np.array([N,N]).T)
        C = np.zeros((Q,d,d))
        for i in range(Q):
            for j in range(n):
                C[i] += gamma[i][j]*np.outer(X_sep_train[class_].iloc[j] - mu.iloc[...
                    i],X_sep_train[class_].iloc[j] - mu.iloc[i])
            C[i] = np.diag(np.diag(C[i]/N[i]))

        L_new = 0
        for i in range(n):
            L_new += np.log(likelihood(X_sep_train[class_].iloc[i],mu.to_numpy(),w,...
                C))
        #print(L_new,L_old)
        difference = L_new - L_old
        L_old = L_new
        L.append(L_new)
    return([mu,w,C,L])
    #L_q.append(L)
    #add accuracy and confusion matrix

##########################################################################
pool = Pool(processes=4)

##########################################################################
from multiprocessing import cpu_count

##########################################################################
cpu_count()

##########################################################################
q = list(range(2,10))

##########################################################################
t1 = time.time()
params = pool.map(f,q)
t2 = time.time()

##########################################################################
class_2_param = params
get_ipython().run_line_magic('store', 'class_2_param')

##########################################################################
dbfile = open("class2_1b",'ab')
pickle.dump(class_2_param,dbfile)
dbfile.close()

##########################################################################
dbfile = open("class0_1b",'rb')
class_0_param = pickle.load(dbfile)
dbfile.close()

##########################################################################
dbfile = open("class1_1b",'rb')
class_1_param = pickle.load(dbfile)
dbfile.close()

##########################################################################
dbfile = open("class2_1b",'rb')
class_2_param = pickle.load(dbfile)
dbfile.close()

##########################################################################
parameters = [class_0_param,class_1_param, class_2_param]

```

```
#########################################################################
import accuracy

#########################################################################
#predicting training data - selecting max likelihood value
d = 2
acc_train = []
for Q in range(len(q)):
    y_Pred = []
    for i in range(600):
        lst = []
        for j in range(featvec_length+1):
            lst.append(likelihood(X_train.iloc[i],parameters[j][Q][0],parameters[j...
                ][Q][1],parameters[j][Q][2]))
        y_Pred.append(lst.index(max(lst)))
        #print(y_Pred[i])
    acc_calc = accuracy.Confusion_matrix(y_Pred,Y_train)
    acc_train.append(acc_calc.accuracy)

#########################################################################
df = pd.DataFrame(list(zip(q,acc_train)),columns=["Hyperparameter Value", "Accuracy...
    "])

#########################################################################
acc_train = pd.read_csv("acc1b_train.csv",index_col = 0)

#########################################################################
acc_train

#########################################################################
plt.plot(acc_train)

#########################################################################
pd.crosstab(ds2_train.iloc[:,featvec_length],y_Pred)

#########################################################################
ds2_test = pd.read_csv("dev2.csv", header = None)

#########################################################################
X_cv,X_test,y_cv,y_test = train_test_split(ds2_test.iloc[:,:2],ds2_test.iloc[:,2], ...
    test_size=0.3, random_state=0)

#########################################################################
acc_cv = []
for Q in range(len(q)):
    y_Pred = []
    for i in range(len(X_cv)):
        lst = []
        for j in range(featvec_length+1):
            lst.append(likelihood(X_cv.iloc[i],parameters[j][Q][0],parameters[j][Q...
                ][1],parameters[j][Q][2]))
        y_Pred.append(lst.index(max(lst)))
        #print(y_Pred[i])
    acc_calc = accuracy.Confusion_matrix(y_Pred,y_cv)
    acc_cv.append(acc_calc.accuracy)

#########################################################################
df = pd.DataFrame(list(zip(q,acc_cv)),columns=["Hyperparameter Value", "Accuracy"])
df.to_csv("acc1b_cv.csv")

#########################################################################
acc_cv = pd.read_csv("acc1b_cv.csv",index_col=0)

#########################################################################
plt.plot(q,acc_train.iloc[:,1],label = "Training Data")
plt.plot(q,acc_cv.iloc[:,1],label = "Validation Data")
plt.xlabel("No. of Gaussian Components")
plt.ylabel("Accuracy")
plt.title("Accuracy with hyperparameter values on Training and Validation data")
plt.legend()
plt.savefig("acc_1b.png")
plt.show()

##########################################################################
acc_cv.index(max(acc_cv))

##########################################################################
q[3]

##########################################################################
acc_train.index(max(acc_train))

##########################################################################
# best model, q = 5
Q = 3
y_Pred = []
for i in range(len(X_test)):
    lst = []
    for j in range(featvec_length+1):
        lst.append(likelihood(X_test.iloc[i],parameters[j][Q][0],parameters[j][Q...
            ][1],parameters[j][Q][2]))
    y_Pred.append(lst.index(max(lst)))
    #print(y_Pred[i])
acc_calc = accuracy.Confusion_matrix(y_Pred,y_test)
acc_test = acc_calc.accuracy

##########################################################################
acc_test

##########################################################################
Q=3
d=2
YPredTrain = []
for i in range(len(X_train)):
    lst = []
    for j in range(3):
        lst.append(likelihood(X_train.iloc[i],parameters[j][Q][0],parameters[j][Q...
            ][1],parameters[j][Q][2]))
    YPredTrain.append(lst.index(max(lst)))

##########################################################################
pd.DataFrame(YPredTrain).to_csv("YPredTrain.csv")
pd.DataFrame(YPredCV).to_csv("YPredCV.csv")
pd.DataFrame(YPredTest).to_csv("YPredTest.csv")
pd.DataFrame(yGridPred).to_csv("YPredGrid.csv")
pd.DataFrame(acc_cv).to_csv("acc_cv.csv")
pd.DataFrame(acc_train).to_csv("acc_train.csv")

##########################################################################
YPredCV = []
for i in range(len(X_cv)):
    lst = []
    for j in range(3):
        lst.append(likelihood(X_cv.iloc[i],parameters[j][Q][0],parameters[j][Q][1],...
            parameters[j][Q][2]))
    YPredCV.append(lst.index(max(lst)))

YPredTest = []
for i in range(len(X_test)):
    lst = []
    for j in range(3):
        lst.append(likelihood(X_test.iloc[i],parameters[j][Q][0],parameters[j][Q...
            ][1],parameters[j][Q][2]))
    YPredTest.append(lst.index(max(lst)))

##########################################################################
import seaborn as sns

```

```
#########################################################################
from sklearn.metrics import confusion_matrix

#########################################################################
conf_mat = confusion_matrix(YPredTrain,Y_train)
plt.figure()
sns.heatmap(conf_mat, annot=True)
plt.title("Training Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("Actual Class")
plt.savefig("conf_train1b.png")
plt.show()

#########################################################################
conf_Train = ac_train.get_matrix()

#########################################################################
pd.DataFrame(conf_Train).to_csv("conf_train_1b.csv")

#########################################################################
ac_test = accuracy.Confusion_matrix(YPredTest,y_test)
conf_Test = ac_test.get_matrix()
pd.DataFrame(conf_Train).to_csv("conf_test_1b.csv")

#########################################################################
conf_mat = confusion_matrix(YPredTest,y_test)
plt.figure()
sns.heatmap(conf_mat, annot=True)
plt.title("Test Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("Actual Class")
plt.savefig("conf_test1b.png")
plt.show()

#########################################################################
for class_val in range(3):
    row_idx = np.where(ds2_train.iloc[:,featvec_length]==class_val)
    plt.scatter(np.array(ds2_train)[row_idx,0],np.array(ds2_train)[row_idx,1])
plt.show()

#########################################################################
Q = 3
d = 2

#########################################################################
min_x1 = min(X_train[0])
max_x1 = max(X_train[0])
min_x2 = min(X_train[1])
max_x2 = max(X_train[1])

x1_range = np.linspace(min_x1,max_x1)
x2_range = np.linspace(min_x2,max_x2)

X1,X2 = np.meshgrid(x1_range,x2_range)

x1,x2 = X1.flatten(),X2.flatten()
x1,x2 = x1.reshape(len(x1),1),x2.reshape(len(x2),1)
grid = np.hstack((x1,x2))

#########################################################################
yGridPred = []
for i in range(len(grid)):
    lst = []
    for j in range(3):
        lst.append(likelihood(grid[i],parameters[j][Q][0],parameters[j][Q][1],...
            parameters[j][Q][2]))
    yGridPred.append(lst.index(max(lst)))
yGridPred = np.array(yGridPred).reshape(X1.shape)

```

```
########################################################################
plt.contourf(X1,X2,yGridPred)
for class_val in range(3):
    row_idx = np.where(ds2_train.iloc[:,featvec_length]==class_val)
    plt.scatter(np.array(ds2_train)[row_idx,0],np.array(ds2_train)[row_idx,1],label...
         = "Class "+ str(class_val))
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.xlabel("X1")
plt.ylabel("X2")
plt.title("Decision region plot with training data superposed")
plt.savefig("decisionReg_ds2.png")
plt.show()

########################################################################

Q=3
d=2
x, y = np.mgrid[-3:3:30j, -3:3:30j]
xy = np.column_stack([x.flat, y.flat])
z0_val = np.zeros(len(xy))
z1_val = np.zeros(len(xy))
z2_val = np.zeros(len(xy))

for i in range(len(xy)):
    lst = np.array((len(xy)))
    z0_val[i] = likelihood(xy[i],parameters[0][Q][0],parameters[0][Q][1],parameters...
        [0][Q][2])
    z1_val[i] = likelihood(xy[i],parameters[1][Q][0],parameters[1][Q][1],parameters...
        [1][Q][2])
    z2_val[i] = likelihood(xy[i],parameters[2][Q][0],parameters[2][Q][1],parameters...
        [2][Q][2])
d = np.hstack((z0_val.reshape(900,-2),z1_val.reshape(900,-2),z2_val.reshape(900,-2)...
    ))
classes = np.argmax(d,axis=1)
classes = classes.reshape(x.shape)


########################################################################
def gaussian_val(X_test,w,mu,C):
    n, d = X_test.shape
    val = np.zeros((n, 5))

    for i in range(5):
        val[:,i] = w[i]*mvn.pdf(X_test, mu.iloc[i], C[i])

    return np.sum(val, axis=1)

########################################################################
color_list = ["springgreen", "mediumturquoise", "palevioletred"]

########################################################################
z0 = np.zeros(len(xy))
z1 = np.zeros(len(xy))
z1 = np.zeros(len(xy))


z0 = gaussian_val(xy,parameters[0][Q][1],parameters[0][Q][0],parameters[0][Q][2])
z1 = gaussian_val(xy,parameters[1][Q][1],parameters[1][Q][0],parameters[1][Q][2])
z2 = gaussian_val(xy,parameters[2][Q][1],parameters[2][Q][0],parameters[2][Q][2])

z0 = z0.reshape(x.shape)


z1 = z1.reshape(x.shape)

z2 = z2.reshape(x.shape)

########################################################################
plt.figure()
```

```
ds2_train.plot.scatter(0, 1, c=[color_list[int(i)] for i in ds2_train[2]], alpha=1)
plt.contourf(x, y, classes, 2, colors=color_list, alpha=0.1)
plt.contour(x, y, classes, 2, colors=color_list, alpha=1)
plt.contour(x, y, z0, levels=np.logspace(-2,2,20), colors=color_list[0])
plt.contour(x, y, z1, levels=np.logspace(-2,2,20), colors=color_list[1])
plt.contour(x, y, z2, levels=np.logspace(-2,2,20), colors=color_list[2])
plt.title("Decision Boundaries + Contours - Diagonal Covariance")
plt.xlabel("X1")
plt.ylabel("X2")
plt.savefig("contour1b.png")
plt.show()
```

The accuracy module is as follows:

```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
class Confusion_matrix():
    def __init__(self,y_pred, y_orig):
        self.pred = y_pred
        self.original = y_orig
        self.length = len(y_pred)
        self.compare = y_pred == y_orig
        self.accuracy = np.sum(self.compare)/self.length
        self.classes = pd.Series(y_orig).unique()[0]

    def get_matrix(self):
        #mat = np.zeros((l,l))
        #conf_matrix = pd.crosstab(self.original,self.pred,rownames=["actual"],...
            colnames = ["predicted"])
        mat = confusion_matrix(self.original,self.pred)
        return(mat)
```

2.3 Bayes Classification, KNN

```
#!/usr/bin/env python
# coding: utf-8

########################################################################
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
get_ipython().run_line_magic('matplotlib', 'inline')


# # For dataset 1b:
########################################################################
col_names=["x1","x2","y"]
data1b=pd.read_csv("train1b.csv",names=col_names)
Xtrain_1=data1b["x1"]
Xtrain_2=data1b["x2"]
Ytrain=np.array(data1b["y"])
Xtrain=np.array(data1b.drop("y",axis=1))

########################################################################
data1b_dev=pd.read_csv("dev1b.csv",names=col_names)

########################################################################
plt.figure()
plt.scatter(Xtrain_1[Ytrain==0],Xtrain_2[Ytrain==0],label="y=0")
plt.scatter(Xtrain_1[Ytrain==1],Xtrain_2[Ytrain==1],label="y=1")
plt.scatter(Xtrain_1[Ytrain==2],Xtrain_2[Ytrain==2],label="y=2")
plt.legend()
plt.xlabel("X1")
plt.ylabel("X2")
plt.title("Scatter plot of data 1b")
plt.savefig("Scatter plot of data 1b.jpg")
plt.show()

#########################################################################
## Shuffles a provided data set and splits it into cross-validation and test dataset

def create_datasets(data,cv_size):
    data.sample(frac=1).reset_index(drop=True)
    test_size=len(data)-cv_size
    data_test=data[0:test_size]
    data_cv=data[test_size:]
    return(data_cv,data_test)

#########################################################################
## Calculates accuracy of the model

def accuracy(y_pred,y_actual):
    true_count=0
    for i in range(len(y_pred)):
        if y_pred[i]==y_actual[i]:
            true_count+=1;
    return(true_count/len(y_pred))

#########################################################################
## Calculates euclidean distance between two vector points

def euclidean(p1,p2):
    d=np.linalg.norm(np.array(p1)-np.array(p2))
    return d

#########################################################################
data1b_cv,data1b_test=create_datasets(data1b_dev,50)

#########################################################################
data1b_test=data1b_test.append(data1b.iloc[595:,:]);

#########################################################################
def knn(x,y,test,k):
    distances=[]
    for i in range(len(x)):
        d=euclidean(x[i],test)
        l=(d,x[i],y[i])
        distances.append(l)
    distances.sort(key = lambda x:x[0])
    count=Counter()
    for i in distances[:k]:
        count[i[2]]+=1
    pred=count.most_common(1)[0][0]
    return(distances[:k],pred)


#########################################################################
k_list=[1,7,15]
Accuracyknn_cv=[]
Accuracyknn_train=[]
Accuracyknn_test=[]

#########################################################################
X_cv=np.array(data1b_cv.drop("y",axis=1))
Y_cv=np.array(data1b_cv["y"])
X_test=np.array(data1b_test.drop("y",axis=1))
Y_test=np.array(data1b_test["y"])

## iterating over k-values
for i in k_list:
    ycv_pred=[]
    for j in X_cv:
        ycv_pred.append(knn(Xtrain,Ytrain,j,i)[1])
    ytest_pred=[]
    for j in X_test:
        ytest_pred.append(knn(Xtrain,Ytrain,j,i)[1])
    ytrain_pred=[]
    for j in Xtrain:
        ytrain_pred.append(knn(Xtrain,Ytrain,j,i)[1])
    Accuracyknn_cv.append(accuracy(Y_cv,ycv_pred))
    Accuracyknn_test.append(accuracy(Y_test,ytest_pred))
    Accuracyknn_train.append(accuracy(Ytrain,ytrain_pred))

#########################################################################
accuracy_table_KNN=pd.DataFrame(list(zip(k_list,Accuracyknn_train,Accuracyknn_cv,...
    Accuracyknn_test)),columns=["k-value", "Accuracy train","Accuracy CV","Accuracy ...
    test"])

#########################################################################
ytrainpred_1=[]
ytestpred_1=[]
for i in Xtrain:
    ytrainpred_1.append(knn(Xtrain,Ytrain,i,1)[1])
for i in X_test:
    ytestpred_1.append(knn(Xtrain,Ytrain,i,1)[1])

#########################################################################
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay

#########################################################################
cm_knn_train=confusion_matrix(ytrainpred_1,Ytrain)
cm_knn_test=confusion_matrix(ytestpred_1,Y_test)
cmd_knn_train=ConfusionMatrixDisplay(cm_knn_train,display_labels=[0.0,1.0,2.0])
plt.figure()
cmd_knn_train.plot()
plt.savefig("1b_cm_knn_train.jpg")

#########################################################################
cmd_knn_test=ConfusionMatrixDisplay(cm_knn_test,display_labels=[0.0,1.0,2.0])
plt.figure()
cmd_knn_test.plot()
plt.savefig("1b_cm_knn_test.jpg")

#########################################################################
accuracy_table_KNN


# ## Bayes classifier with KNN to calculate class conditional probabilities
#########################################################################
## Seperating the rows by class values

def seperate_by_classval(data):
    ## the target variable must be stored in a column named "y"
    class_vals=list(data["y"].unique())
    seperated=dict()
    features=data.drop('y',axis=1)
    Y=np.array(data["y"])
    ## creates a key value corresponding to each class label
    for i in class_vals:
        seperated[i]=features[Y==i];
    return(seperated)

#########################################################################
## Calculates the prior probability of classes and returns a dictionary such that
## probs[i] is the prior probability of class i

def priori(data):
    seperated_data=seperate_by_classval(data)
    probs=dict()
    for i in seperated_data.keys():
        probs[i]=len(seperated_data[i])/len(data);
    return probs

########################################################################
## Calculates the class-conditional probability p(x/yi) using knn method
## the input x is the data points for a particular class i
## Each row of knn_list consists of a nearest neighbour and its distance from the ...
    test point
## prob is the class conditional probability p(x/yi)

def knn_prob(x,test,k):
    distances=[]
    for i in range(len(x)):
        d=euclidean(x[i],test)
        l=(d,x[i])
        distances.append(l)
    distances.sort(key = lambda x:x[0])
    knn_list=distances[:k]
    r=knn_list[-1][0]
    prob=k/(np.pi*r**2*len(x))
    return(knn_list,prob)

########################################################################
## This uses the above code blocks to evaluate p(yi/x) for all the classes
## Returns a dictionary probabs, such that probabs[i] is the p(yi/x)
## also returns label which is the class label corresponding to the maximum p(yi/x)

def predictor(train_data,k,test_data):
    X_train=seperate_by_classval(train_data)
    p_y=priori(train_data)
    p=0
    probabs={}
    for i in list(priori(train_data).keys()):
        p_yi=p_y[i]
        X_traini=X_train[i]
        px_yi=knn_prob(np.array(X_traini),test_data,k)[1]
        pyi_x=px_yi*p_yi
        probabs[i]=pyi_x
        if probabs[i]>p:
            p=probabs[i]
            label=i
    sum_vals=sum(list(probabs.values()))
    for i in probabs.keys():
        probabs[i]=probabs[i]/sum_vals
    return(probabs,label)


# ### Predicting for k=10 and k=20
########################################################################
ypred10_cv=[]
ypred10_test=[]
ypred20_cv=[]
ypred20_test=[]
ypred10_train=[]
ypred20_train=[]
for i in range(len(data1b_cv)):
    ypred10_cv.append(predictor(data1b,10,data1b_cv.iloc[i,:-1])[1])
    ypred20_cv.append(predictor(data1b,20,data1b_cv.iloc[i,:-1])[1])
for i in range(len(data1b_test)):
    ypred10_test.append(predictor(data1b,10,data1b_test.iloc[i,:-1])[1])
    ypred20_test.append(predictor(data1b,20,data1b_test.iloc[i,:-1])[1])
for i in range(len(data1b)):
    ypred10_train.append(predictor(data1b,10,data1b.iloc[i,:-1])[1])
    ypred20_train.append(predictor(data1b,20,data1b.iloc[i,:-1])[1])

########################################################################
accuracy_table=pd.DataFrame()
accuracy_table["k-value"]=[10,20]
accuracy_table["Train data"]=[accuracy(ypred10_train,list(data1b.iloc[:,-1])),...
    accuracy(ypred20_train,list(data1b.iloc[:,-1]))]
```

```
accuracy_table["CV data"]=[accuracy(ypred10_cv,list(data1b_cv.iloc[:,-1])),accuracy...
    (ypred20_cv,list(data1b_cv.iloc[:,-1]))]
accuracy_table["Test data"]=[accuracy(ypred10_test,list(data1b_test.iloc[:,-1])),...
    accuracy(ypred20_test,list(data1b_test.iloc[:,-1]))]

##########################################################################
accuracy_table

##########################################################################
cm_nb_train=confusion_matrix(ypred10_train,Ytrain)
cm_nb_test=confusion_matrix(ypred10_test,Y_test)
cmd_nb_train=ConfusionMatrixDisplay(cm_nb_train,display_labels=[0.0,1.0,2.0])
plt.figure()
cmd_nb_train.plot()
plt.savefig("1b_cm_nb_train.jpg")

##########################################################################
cmd_nb_test=ConfusionMatrixDisplay(cm_nb_test,display_labels=[0.0,1.0,2.0])
plt.figure()
cmd_nb_test.plot()
plt.savefig("1b_cm_nb_test.jpg")


# ### Decision region plots:
##########################################################################
min1,max1=data1b["x1"].min()-1,data1b["x1"].max()+1
min2,max2=data1b["x2"].min()-1,data1b["x2"].max()+1

resolution=0.5
x1grid=np.arange(min1,max1,resolution)
x2grid=np.arange(min2,max2,resolution)

xx,yy=np.meshgrid(x1grid,x2grid)

r1,r2=xx.flatten(),yy.flatten()
r1,r2=r1.reshape((len(r1),1)),r2.reshape((len(r2),1))

grid=np.hstack((r1,r2))

##########################################################################
yhat_knn=[]
for i in range(len(grid)):
    yhat_knn.append(knn(Xtrain,Ytrain,grid[i,:],10)[1])

##########################################################################
yhat_knn=np.array(yhat_knn)
zz_knn=yhat_knn.reshape(xx.shape)

##########################################################################
plt.figure()
plt.contourf(xx,yy,zz_knn,alpha=0.6,cmap="Paired")
plt.scatter(Xtrain_1[Ytrain==0],Xtrain_2[Ytrain==0],label="y=0",c="Blue")
plt.scatter(Xtrain_1[Ytrain==1],Xtrain_2[Ytrain==1],label="y=1",c="red")
plt.scatter(Xtrain_1[Ytrain==2],Xtrain_2[Ytrain==2],label="y=2",c="Brown")
plt.legend()
plt.xlabel("X1")
plt.ylabel("X2")
plt.title("Decision region plot of data 1b, knn classifier")
plt.savefig("1b_knn_decision_region.jpg")
plt.show()

##########################################################################
yhat_nb=[]
for i in range(len(grid)):
    yhat_nb.append(predictor(data1b,10,grid[i,:])[1])
yhat_nb=np.array(yhat_nb)
zz_nb=yhat_nb.reshape(xx.shape)

##########################################################################
```

```
plt.figure()
plt.contourf(xx,yy,zz_nb,alpha=0.6,cmap="Paired")
plt.scatter(Xtrain_1[Ytrain==0],Xtrain_2[Ytrain==0],label="y=0",c="Blue")
plt.scatter(Xtrain_1[Ytrain==1],Xtrain_2[Ytrain==1],label="y=1",c="red")
plt.scatter(Xtrain_1[Ytrain==2],Xtrain_2[Ytrain==2],label="y=2",c="Brown")
plt.legend()
plt.xlabel("X1")
plt.ylabel("X2")
plt.title("Decision region plot of data 1b, bayes with knn classifier")
plt.savefig("1b_nb_decision_region.jpg")
plt.show()
```

3 Dataset 2A

3.1 Bayes Classification, GMM, Full Covariance

The GMM full covariance model code is as follows:

```
#!/usr/bin/env python
# coding: utf-8

#########################################################################
import time
import numpy as np
import pandas as pd
from gmm import GMM
from tqdm import tqdm
import matplotlib.pyplot as plt
from multiprocessing import Pool
from collections import defaultdict
from scipy.stats import multivariate_normal as mvn
from sklearn.model_selection import train_test_split

plt.rcParams["font.size"] = 18
plt.rcParams["axes.grid"] = True
plt.rcParams["figure.figsize"] = 8,6
plt.rcParams['font.serif'] = "Cambria"
plt.rcParams['font.family'] = "serif"

get_ipython().run_line_magic('load_ext', 'autoreload')
get_ipython().run_line_magic('autoreload', '2')


#########################################################################
df = pd.read_csv("../datasets/2A/consolidated_train.csv")
X = df.drop("class", axis=1).to_numpy()
df.head()


#########################################################################
classes = np.unique(df["class"])
gmm_list = defaultdict(list)
q_list = list(range(2,23))

for i in classes:
    print("="*50)
    df_select = df[df["class"]==i]
    X_select = df_select.drop("class", axis=1).to_numpy()
    for q in q_list:
        gmm = GMM(q=q)
        gmm.fit(X_select)
        gmm_list[i].append(gmm)


#########################################################################
import pickle
```

```python
fin = open("2a_gmm_results", "wb")
pickle.dump(gmm_list, fin)
fin.close()


#########################################################################
df_test = pd.read_csv("../datasets/2A/consolidated_dev.csv")
df_cv = df_test.sample(frac=0.7)
X_cv = df_cv.drop("class", axis=1).to_numpy()
display(df_cv.head())
df_test = df_test.drop(df_cv.index)
X_test = df_test.drop("class", axis=1).to_numpy()
display(df_test.head())


#########################################################################
accuracy_list = []
test_accuracy_list = []
for i in tqdm(range(len(q_list))):
    gmm0 = gmm_list[0][i]
    gmm1 = gmm_list[1][i]
    gmm2 = gmm_list[2][i]
    gmm3 = gmm_list[3][i]
    gmm4 = gmm_list[4][i]

    # Training
    a = gmm0.indv_log_likelihood(X)
    b = gmm1.indv_log_likelihood(X)
    c = gmm2.indv_log_likelihood(X)
    d = gmm3.indv_log_likelihood(X)
    e = gmm4.indv_log_likelihood(X)

    f = np.hstack((a, b, c, d, e))
    pred = np.argmax(f, axis=1)
    accuracy_list.append(np.sum(pred == df["class"])/df["class"].size)

    # Testing
    a = gmm0.indv_log_likelihood(X_test)
    b = gmm1.indv_log_likelihood(X_test)
    c = gmm2.indv_log_likelihood(X_test)
    d = gmm3.indv_log_likelihood(X_test)
    e = gmm4.indv_log_likelihood(X_test)

    f = np.hstack((a, b, c, d, e))
    pred = np.argmax(f, axis=1)
    test_accuracy_list.append(np.sum(pred == df_test["class"])/df_test["class"]....
        size)


#########################################################################
plt.plot(q_list, accuracy_list, '.-')
plt.title("Accuracy across varying Q")
plt.xlabel("Q for each class")
plt.ylabel("Accuracy")
plt.show()

plt.plot(q_list, cv_accuracy_list, '.-')
plt.title("CV Accuracy across varying Q")
plt.xlabel("Q for each class")
plt.ylabel("Accuracy")
plt.show()

plt.plot(q_list, test_accuracy_list, '.-')
plt.title("Test Accuracy across varying Q")
plt.xlabel("Q for each class")
plt.ylabel("Accuracy")
plt.show()


```

```python
##########################################################################
import seaborn as sns
from sklearn.metrics import confusion_matrix

best_model = np.argmax(acc["Sum"])

gmm0 = gmm_list[0][best_model]
gmm1 = gmm_list[1][best_model]
gmm2 = gmm_list[2][best_model]
gmm3 = gmm_list[3][best_model]
gmm4 = gmm_list[4][best_model]

# Training
a = gmm0.indv_log_likelihood(X)
b = gmm1.indv_log_likelihood(X)
c = gmm2.indv_log_likelihood(X)
d = gmm3.indv_log_likelihood(X)
e = gmm4.indv_log_likelihood(X)

f = np.hstack((a, b, c, d, e))
pred = np.argmax(f, axis=1)
conf_mat = confusion_matrix(pred, df["class"])
plt.figure()
sns.heatmap(conf_mat, annot=True)
plt.title("Training Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("Actual Class")
plt.show()

# CV
a = gmm0.indv_log_likelihood(X_cv)
b = gmm1.indv_log_likelihood(X_cv)
c = gmm2.indv_log_likelihood(X_cv)
d = gmm3.indv_log_likelihood(X_cv)
e = gmm4.indv_log_likelihood(X_cv)

f = np.hstack((a, b, c, d, e))
pred = np.argmax(f, axis=1)
conf_mat = confusion_matrix(pred, df_cv["class"])
plt.figure()
sns.heatmap(conf_mat, annot=True)
plt.title("Validation Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("Actual Class")
plt.show()

# Testing
a_test = gmm0.indv_log_likelihood(X_test)
b_test = gmm1.indv_log_likelihood(X_test)
c_test = gmm2.indv_log_likelihood(X_test)
d_test = gmm3.indv_log_likelihood(X_test)
e_test = gmm4.indv_log_likelihood(X_test)

f_test = np.hstack((a_test, b_test, c_test, d_test, e_test))
pred_test = np.argmax(f_test, axis=1)
conf_mat = confusion_matrix(pred_test, df_test["class"])
plt.figure()
sns.heatmap(conf_mat, annot=True)
plt.title("Testing Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("Actual Class")
plt.show()


##########################################################################
import seaborn as sns
from sklearn.metrics import confusion_matrix

gmm0 = gmm_list[0][0]
```

```
gmm1 = gmm_list[1][0]
gmm2 = gmm_list[2][0]
gmm3 = gmm_list[3][4]
gmm4 = gmm_list[4][3]

# Training
a = gmm0.indv_log_likelihood(X)
b = gmm1.indv_log_likelihood(X)
c = gmm2.indv_log_likelihood(X)
d = gmm3.indv_log_likelihood(X)
e = gmm4.indv_log_likelihood(X)

f = np.hstack((a, b, c, d, e))
pred = np.argmax(f, axis=1)
conf_mat = confusion_matrix(pred, df["class"])
plt.figure()
sns.heatmap(conf_mat, annot=True)
plt.title("Training Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("Actual Class")
plt.show()

# CV
a = gmm0.indv_log_likelihood(X_cv)
b = gmm1.indv_log_likelihood(X_cv)
c = gmm2.indv_log_likelihood(X_cv)
d = gmm3.indv_log_likelihood(X_cv)
e = gmm4.indv_log_likelihood(X_cv)

f = np.hstack((a, b, c, d, e))
pred = np.argmax(f, axis=1)
conf_mat = confusion_matrix(pred, df_cv["class"])
plt.figure()
sns.heatmap(conf_mat, annot=True)
plt.title("Validation Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("Actual Class")
plt.show()

# Testing
a_test = gmm0.indv_log_likelihood(X_test)
b_test = gmm1.indv_log_likelihood(X_test)
c_test = gmm2.indv_log_likelihood(X_test)
d_test = gmm3.indv_log_likelihood(X_test)
e_test = gmm4.indv_log_likelihood(X_test)

f_test = np.hstack((a_test, b_test, c_test, d_test, e_test))
pred_test = np.argmax(f_test, axis=1)
conf_mat = confusion_matrix(pred_test, df_test["class"])
plt.figure()
sns.heatmap(conf_mat, annot=True)
plt.title("Testing Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("Actual Class")
plt.show()
```

The GMM class module is as follows:

```
import numpy as np
from tqdm import tqdm
from sklearn.cluster import KMeans
from scipy.stats import multivariate_normal as mvn
import pandas as pd

class GMM():
    def __init__(self, q):
        self.q = q

    def fit(self, X, covariance_type="diag", tol=1e-5):
        """
        X: n*d
        mu: q*d
        C: q*d*d
        gamma: n*q
        """
        self.n, self.d = X.shape
        self.X = X
        self.covariance_type = covariance_type
        self.initialization()
        self.lglk_list = []
        for i in tqdm(range(100)):
            self.lglk_list.append(self.log_likelihood(self.X))
            self.expectation()
            self.maximization()
            new_lk = self.log_likelihood(self.X)
            diff = new_lk - self.lglk_list[-1]
            if  diff < tol:
                if diff < 0: print("Difference is less than 0")
                break


    def initialization(self):
        # kmeans = KMeans(n_clusters=self.q, random_state=0).fit(self.X)
        kmeans = KMeans(n_clusters=self.q).fit(self.X)
        labels = kmeans.labels_
        unique, counts = np.unique(labels, return_counts=True)

        self.subcomponents = unique.size
        self.gamma = np.eye(self.subcomponents)[labels]
        self.Nq = np.sum(self.gamma, axis=0)
        self.weights = counts/self.n
        self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
        self.C = np.zeros((self.subcomponents, self.d, self.d))

        for i in range(self.q):
            self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self.mu[i,:])).T@(self.X-self.mu[i,:])

            if self.covariance_type == "diag":
                self.C[i] = np.diag(self.C[i])


    def expectation(self):
        self.gamma = np.zeros((self.n, self.q))

        for i in range(self.q):
            try:
                self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self.C[i])
            except:
                self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self.C[i]+np.eye(self.C[i].shape[0])*1e-7)
        self.gamma = self.gamma/np.sum(self.gamma, axis=1).reshape(-1,1)

    def maximization(self):
        # print(np.sum(self.weights))
        self.Nq = np.sum(self.gamma, axis=0)
        self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)

        for i in range(self.q):
            self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self.mu[i,:])).T@(self.X-self.mu[i,:])

            if self.covariance_type == "diag":
                self.C[i] = np.diag(self.C[i])

        self.weights = self.Nq/self.n

```

```python
    def log_likelihood(self, X_test):
        lk = 0
        n, d = X_test.shape
        for i in range(n):
            val = 0
            for j in range(self.q):
                try:
                    val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j])
                except:
                    val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j]+np.eye(self.C[j].shape[0])*1e-7)
            lk += np.log(val)

        return lk

    def indv_log_likelihood(self, X_test):
        n, d = X_test.shape
        lk = np.zeros((X_test.shape[0], 1))
        for i in range(n):
            val = 0
            for j in range(self.q):
                try:
                    val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j])
                except:
                    val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j]+np.eye(self.C[j].shape[0])*1e-7)
            lk[i] = np.log(val)

        return lk

    def gaussian_val(self, X_test):
        n, d = X_test.shape
        val = np.zeros((n, self.q))

        for i in range(self.q):
            val[:,i] = self.weights[i]*mvn.pdf(X_test, self.mu[i], self.C[i])

        return np.sum(val, axis=1)

class GMM_vl():
    def __init__(self, q):
        self.q = q

    def fit(self, X, epochs=100, covariance_type="diag", tol=1e-5):
        """
        X: n*d
        mu: q*d
        C: q*d*d
        gamma: n*q
        """
        self.n, self.d = X.shape
        self.X = X
        self.epochs = epochs
        self.covariance_type = covariance_type
        self.initialization()
        self.lglk_list = []
        for i in tqdm(range(self.epochs)):
            self.lglk_list.append(self.log_likelihood(self.X))
            self.expectation()
            self.maximization()
            new_lk = self.log_likelihood(self.X)
            diff = new_lk - self.lglk_list[-1]
            if  diff < tol:
                if diff < 0:
                    print("Difference is less than 0")
                    break

    def initialization(self):
    # kmeans = KMeans(n_clusters=self.q, random_state=0).fit(self.X)
        kmeans = KMeans(n_clusters=self.q).fit(self.X)
        labels = kmeans.labels_
        unique, counts = np.unique(labels, return_counts=True)

        self.subcomponents = unique.size
        self.gamma = np.eye(self.subcomponents)[labels]
        self.Nq = np.sum(self.gamma, axis=0)
        self.weights = counts/self.n
        self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
        self.C = np.zeros((self.subcomponents, self.d, self.d))

        for i in range(self.q):
            self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self....
                mu[i,:])).T@(self.X-self.mu[i,:])

            if self.covariance_type == "diag":
                self.C[i] = np.diag(np.diag(self.C[i]))


    def expectation(self):
        self.gamma = np.zeros((self.n, self.q))

        for i in range(self.q):
            try:
                self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self....
                    C[i])
            except:
                self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self....
                    C[i]+np.eye(self.C[i].shape[0])*1e-3)
                self.gamma = self.gamma/np.sum(self.gamma, axis=1).reshape(-1,1)

    def maximization(self):
        # print(np.sum(self.weights))
        self.Nq = np.sum(self.gamma, axis=0)
        self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)

        for i in range(self.q):
            self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self....
                mu[i,:])).T@(self.X-self.mu[i,:])

        if self.covariance_type == "diag":
            self.C[i] = np.diag(np.diag(self.C[i]))

        self.weights = self.Nq/self.n

    def log_likelihood(self, X_test):
        lk = 0
        n, d = X_test.shape
        for i in range(n):
            val = 0
            for j in range(self.q):
                try:
                        val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self....
                            C[j])
                except:
                    val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j...
                        ]+np.eye(self.C[j].shape[0])*1e-3)
            lk += np.log(val)

        return lk

    def indv_log_likelihood(self, X_test):
        n, d = X_test.shape
        lk = np.zeros((X_test.shape[0], 1))
        for i in range(n):
            val = 0
            for j in range(self.q):
                try:
                        val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self....
                           C[j])
                except:
                        val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self....
                           C[j]+np.eye(self.C[j].shape[0])*1e-3)
            lk[i] = np.log(val)

        return lk

    def gaussian_val(self, X_test):
        n, d = X_test.shape
        val = np.zeros((n, self.q))

        for i in range(self.q):
            val[:,i] = self.weights[i]*mvn.pdf(X_test, self.mu[i], self.C[i])

        return np.sum(val, axis=1)

    def probab(self, df):
        df = pd.DataFrame(df)
        grouped_df = df.groupby(by=["class", "image"])
        for key, item in grouped_df:
            selected_df = grouped_df.get_group(key)
            X_select = selected_df.drop(["index", "image", "class"], axis=1)....
                to_numpy()
            val = self.gaussian_val(X_select)
        print(val.shape)
```

The utils script is as follows:

```
import os
import numpy as np
import pandas as pd
from tqdm import tqdm

def get_consolidated_data2A(classes_present):
    df = pd.DataFrame()
    df_test = pd.DataFrame()
    for i in classes_present:
        df_new = pd.read_csv("../datasets/2A/"+i+"/train.csv")
        # df_new = pd.read_csv("../datasets/2A/"+i+"/train.csv", nrows=182)
        df_new["image_names"] = classes_present[i]
        df_new = df_new.rename(columns={"image_names":"class"})
        df = df.append(df_new)

        df_new_test = pd.read_csv("../datasets/2A/"+i+"/dev.csv")
        # df_new_test = pd.read_csv("../datasets/2A/"+i+"/dev.csv", nrows=52)
        df_new_test["image_names"] = classes_present[i]
        df_new_test = df_new_test.rename(columns={"image_names":"class"})
        df_test = df_test.append(df_new_test)

    df.to_csv("../datasets/2A/consolidated_train.csv", index=False)
    df_test.to_csv("../datasets/2A/consolidated_dev.csv", index=False)
    # df.to_csv("../datasets/2A/consolidated_train_small.csv", index=False)
    # df_test.to_csv("../datasets/2A/consolidated_dev_small.csv", index=False)

def get_consolidated_data2B(classes_present):
    df = pd.DataFrame()
    df_test = pd.DataFrame()

    for i in classes_present:
        files = os.listdir("../datasets/2B/"+i+"/train/")
        for k,j in tqdm(enumerate(files)):
            df_new = pd.read_csv("../datasets/2B/"+i+"/train/"+j, header=None, sep=...
                " ")
            df_new["class"] = classes_present[i]
            df_new = df_new.reset_index()
            df_new["image"] = str(k)
            df = df.append(df_new)

        files = os.listdir("../datasets/2B/"+i+"/dev/")
        for k,j in tqdm(enumerate(files)):
            df_new_test = pd.read_csv("../datasets/2B/"+i+"/dev/"+j, header=None, ...
                sep=" ")
            df_new_test["class"] = classes_present[i]
            df_new_test = df_new_test.reset_index()
            df_new_test["image"] = str(k)
            df_test = df.append(df_new_test)

    df.to_csv("../datasets/2B/consolidated_train.csv", index=False)
    df_test.to_csv("../datasets/2B/consolidated_dev.csv", index=False)


if __name__ == "__main__":
    classes_present = {"coast":0, "highway":1, "mountain":2, "opencountry":3, "...
        tallbuilding":4}
    get_consolidated_data2B(classes_present)
```

3.2 Bayes Classification, GMM, Diagonal Covariance

The GMM diagonal covariance model code is as follows:

```
#!/usr/bin/env python
# coding: utf-8

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


#########################################################################
import seaborn as sns


#########################################################################
plt.rcParams["font.size"] = 18
plt.rcParams["axes.grid"] = True
plt.rcParams["figure.figsize"] = 8,6
plt.rcParams['font.serif'] = "Cambria"
plt.rcParams['font.family'] = "serif"


#########################################################################
import statistics as sts
from sklearn.model_selection import train_test_split


#########################################################################
from sklearn.cluster import KMeans


#########################################################################
coast_train = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/coast/train.csv"...
    )
mountain_train = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/mountain/...
    train.csv")
tallbuilding_train = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/...
    tallbuilding/train.csv")
highway_train = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/highway/train....
    csv")
opencountry_train = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/...
    opencountry/train.csv")

coast_train.drop(["image_names"],axis = 1,inplace=True)
```

```python
mountain_train.drop(["image_names"],axis = 1,inplace=True)
tallbuilding_train.drop(["image_names"],axis = 1,inplace=True)
highway_train.drop(["image_names"],axis = 1,inplace=True)
opencountry_train.drop(["image_names"],axis = 1,inplace=True)


#########################################################################
coast_test = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/coast/dev.csv")
mountain_test = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/mountain/dev.csv")
tallbuilding_test = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/tallbuilding/dev.csv")
highway_test = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/highway/dev.csv")
opencountry_test = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/opencountry/dev.csv")

coast_test.drop(["image_names"],axis = 1,inplace=True)
mountain_test.drop(["image_names"],axis = 1,inplace=True)
tallbuilding_test.drop(["image_names"],axis = 1,inplace=True)
highway_test.drop(["image_names"],axis = 1,inplace=True)
opencountry_test.drop(["image_names"],axis = 1,inplace=True)


#########################################################################
coast_train.head()


#########################################################################
class GMM():
    def __init__(self, q):
        self.q = q

    def fit(self, X, tol=1e-3):
        """
        X: n*d
        mu: q*d
        C: q*d*d
        gamma: n*q
        """
        self.n, self.d = X.shape
        self.X = X
        #self.covariance_type = covariance_type
        self.initialization()
        self.lglk_list = []
        for i in tqdm(range(100)):
            self.lglk_list.append(self.log_likelihood(self.X))
            self.expectation()
            self.maximization()
            new_lk = self.log_likelihood(self.X)
            if new_lk - self.lglk_list[-1] < tol:
                break


    def initialization(self):
        kmeans = KMeans(n_clusters=self.q, random_state=0).fit(self.X)
        labels = kmeans.labels_
        unique, counts = np.unique(labels, return_counts=True)

        self.subcomponents = unique.size
        self.gamma = np.eye(self.subcomponents)[labels]
        self.Nq = np.sum(self.gamma, axis=0)
        self.weights = counts/self.n
        self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
        self.C = np.zeros((self.subcomponents, self.d, self.d))

        for i in range(self.q):
            self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self.mu.iloc[i,:])).T@(self.X-self.mu.iloc[i,:])

            self.C[i] = np.diag(np.diag(self.C[i]))


    def expectation(self):
        self.gamma = np.zeros((self.n, self.q))
        for i in range(self.q):
            try:
                self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu.iloc[i], ...
                    self.C[i])
            except:
                self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu.iloc[i], ...
                    self.C[i]+np.eye(self.C[i].shape[0])*1e-5)
        self.gamma = self.gamma/np.sum(self.gamma, axis=1).reshape(-1,1)

    def maximization(self):
        # print(np.sum(self.weights))
        self.Nq = np.sum(self.gamma, axis=0)
        self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)

        for i in range(self.q):
            self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self....
                mu.iloc[i,:])).T@(self.X-self.mu.iloc[i,:])

            self.C[i] = np.diag(np.diag(self.C[i]))

        self.weights = self.Nq/self.n

    def log_likelihood(self, X_test):
        lk = 0
        n, d = X_test.shape
        for i in range(n):
            val = 0
            for j in range(self.q):
                #self.C[j] += np.eye(self.d)*1e-7
                val += self.weights[j]*mvn.pdf(X_test.iloc[i], self.mu.iloc[j], ...
                    self.C[j])
            lk += np.log(val)

        return lk

    def indv_log_likelihood(self, X_test):
        n, d = X_test.shape
        lk = np.zeros((X_test.shape[0], 1))
        for i in range(n):
            val = 0
            for j in range(self.q):
                val += self.weights[j]*mvn.pdf(X_test.iloc[i], self.mu.iloc[j], ...
                    self.C[j])
            lk[i] = np.log(val)

        return lk


##########################################################################
gmm_list = defaultdict(list)


##########################################################################
Q = list(range(2,15))
for q in Q:
    gmm = GMM(q)
    gmm.fit(mountain_train)
    gmm_list[4].append(gmm)



##########################################################################
import accuracy
```

```


##########################################################################
#predicting training data - selecting max likelihood value
X = mountain_train
ln = len(X)
Y_train = np.array([4]*ln)
acc_train = []
for i in tqdm(range(len(Q))):
    gmm0 = gmm_list[0][i]
    gmm1 = gmm_list[1][i]
    gmm2 = gmm_list[2][i]
    gmm3 = gmm_list[3][i]
    gmm4 = gmm_list[4][i]

    # Training
    a = gmm0.indv_log_likelihood(X)
    b = gmm1.indv_log_likelihood(X)
    c = gmm2.indv_log_likelihood(X)
    d = gmm3.indv_log_likelihood(X)
    e = gmm4.indv_log_likelihood(X)

    f = np.hstack((a, b, c, d, e))
    pred = np.argmax(f, axis=1)

    acc_calc = accuracy.Confusion_matrix(pred,Y_train)
    acc_train.append(acc_calc.accuracy)
m_acc_train = acc_train


##########################################################################
train_acc = pd.DataFrame([c_acc_train,h_acc_train,t_acc_train,o_acc_train,...
    m_acc_train])


##########################################################################
get_ipython().run_line_magic('store', 'train_acc')


##########################################################################
from sklearn.metrics import confusion_matrix


##########################################################################
X = mountain_train
ln = len(X)
Y_train = np.array([4]*ln)
acc_train = []
acc_cv = []
for i in tqdm(range(len(Q))):
    gmm0 = gmm_list[0][i]
    gmm1 = gmm_list[1][i]
    gmm2 = gmm_list[2][i]
    gmm3 = gmm_list[3][i]
    gmm4 = gmm_list[4][i]

    # Training
    a = gmm0.indv_log_likelihood(X)
    b = gmm1.indv_log_likelihood(X)
    c = gmm2.indv_log_likelihood(X)
    d = gmm3.indv_log_likelihood(X)
    e = gmm4.indv_log_likelihood(X)

    f = np.hstack((a, b, c, d, e))
    pred = np.argmax(f, axis=1)

    acc_calc = accuracy.Confusion_matrix(pred,y_cv)
    acc_cv.append(acc_calc.accuracy)
o_acc_cv = acc_cv
```

```


#########################################################################
cv_acc = pd.DataFrame([c_acc_cv,h_acc_cv,t_acc_cv,o_acc_cv,m_acc_cv])


#########################################################################
df = pd.DataFrame(list(zip(Q,train_acc.mean(axis=0),cv_acc.mean(axis=0))),columns=[...
    "Hyperparameter Value", "Accuracy for training data", "Accuracy for validation ...
    data"])
df.to_csv("acc2a.csv")


#########################################################################
plt.plot(Q,df.iloc[:,2],label="train")
plt.plot(Q,df.iloc[:,1],label = "test")
plt.title("Accuracy for training and test data 2A")
plt.xlabel("no. of components")
plt.ylabel("accuracy")
plt.legend()
plt.savefig("acc_2a.png")
plt.show()


#########################################################################
Q[5]


#########################################################################
X_test = mountain_test
ln = len(X_test)

Y_test = np.array([4]*ln)
X_cv,X_test,y_cv,y_test = train_test_split(X_test,Y_test, test_size=0.3, ...
    random_state=2)
ln = len(X_test)
X = X_test
i = 5
gmm0 = gmm_list[0][i]
gmm1 = gmm_list[1][i]
gmm2 = gmm_list[2][i]
gmm3 = gmm_list[3][i]
gmm4 = gmm_list[4][i]

# Training
a = gmm0.indv_log_likelihood(X)
b = gmm1.indv_log_likelihood(X)
c = gmm2.indv_log_likelihood(X)
d = gmm3.indv_log_likelihood(X)
e = gmm4.indv_log_likelihood(X)

f = np.hstack((a, b, c, d, e))
pred = np.argmax(f, axis=1)

acc_calc = accuracy.Confusion_matrix(pred,y_test)
acc_test.append(acc_calc.accuracy)
#o_acc_cv = acc_cv


#########################################################################
np.mean(np.array(acc_test))


#########################################################################
X_train = coast_train
X_train = X_train.append([highway_train,tallbuilding_train,opencountry_train,...
    mountain_train])


```

```
#########################################################################
#predicting training data - selecting max likelihood value
Y_train = [[0]*len(coast_train),[1]*len(highway_train),[2]*len(tallbuilding_train)...
    ,[3]*len(opencountry_train),[4]*len(mountain_train)]
X = X_train
#ln = len(X)
#Y_train = np.array([4]*ln)
acc_train = []
i = 5
gmm0 = gmm_list[0][i]
gmm1 = gmm_list[1][i]
gmm2 = gmm_list[2][i]
gmm3 = gmm_list[3][i]
gmm4 = gmm_list[4][i]

# Training
a = gmm0.indv_log_likelihood(X)
b = gmm1.indv_log_likelihood(X)
c = gmm2.indv_log_likelihood(X)
d = gmm3.indv_log_likelihood(X)
e = gmm4.indv_log_likelihood(X)

f = np.hstack((a, b, c, d, e))
pred = np.argmax(f, axis=1)


#########################################################################
flat_list = [item for sublist in Y_train for item in sublist]
pd.DataFrame(confusion_matrix(pred,flat_list)).to_csv("conf_train_2a.csv")


#########################################################################
X_test = coast_test
X_test  = X_test.append([highway_test,tallbuilding_test,opencountry_test,...
    mountain_test])
Y_test = [[0]*len(coast_test),[1]*len(highway_test),[2]*len(tallbuilding_test),[3]*...
    len(opencountry_test),[4]*len(mountain_test)]


#########################################################################
X = X_test
i = 5
gmm0 = gmm_list[0][i]
gmm1 = gmm_list[1][i]
gmm2 = gmm_list[2][i]
gmm3 = gmm_list[3][i]
gmm4 = gmm_list[4][i]

# Training
a = gmm0.indv_log_likelihood(X)
b = gmm1.indv_log_likelihood(X)
c = gmm2.indv_log_likelihood(X)
d = gmm3.indv_log_likelihood(X)
e = gmm4.indv_log_likelihood(X)

f = np.hstack((a, b, c, d, e))
pred = np.argmax(f, axis=1)


#########################################################################
flat_list = [item for sublist in Y_test for item in sublist]
pd.DataFrame(confusion_matrix(pred,flat_list)).to_csv("conf_test_2a.csv")


#########################################################################
conf_train = pd.read_csv("conf_train_2a.csv",index_col = 0)
conf_test = pd.read_csv("conf_test_2a.csv",index_col = 0)


```

```
########################################################################
plt.figure()
sns.heatmap(conf_train, annot=True)
plt.title("Training Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("Actual Class")
plt.savefig("conf_train2a.png")
plt.show()


########################################################################
plt.figure()
sns.heatmap(conf_test, annot=True)
plt.title("Test Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("Actual Class")
plt.savefig("conf_test2a.png")
plt.show()


########################################################################
```

4 Dataset 2B

The GMM class module is as follows:

```
import numpy as np
from tqdm import tqdm
from sklearn.cluster import KMeans
from scipy.stats import multivariate_normal as mvn
import pandas as pd

class GMM():
    def __init__(self, q):
        self.q = q

    def fit(self, X, covariance_type="diag", tol=1e-5):
        """
        X: n*d
        mu: q*d
        C: q*d*d
        gamma: n*q
        """
        self.n, self.d = X.shape
        self.X = X
        self.covariance_type = covariance_type
        self.initialization()
        self.lglk_list = []
        for i in tqdm(range(100)):
            self.lglk_list.append(self.log_likelihood(self.X))
            self.expectation()
            self.maximization()
            new_lk = self.log_likelihood(self.X)
            diff = new_lk - self.lglk_list[-1]
            if  diff < tol:
                if diff < 0: print("Difference is less than 0")
                break


    def initialization(self):
        # kmeans = KMeans(n_clusters=self.q, random_state=0).fit(self.X)
        kmeans = KMeans(n_clusters=self.q).fit(self.X)
        labels = kmeans.labels_
        unique, counts = np.unique(labels, return_counts=True)

        self.subcomponents = unique.size
        self.gamma = np.eye(self.subcomponents)[labels]
        self.Nq = np.sum(self.gamma, axis=0)
        self.weights = counts/self.n
        self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
        self.C = np.zeros((self.subcomponents, self.d, self.d))

        for i in range(self.q):
            self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self....
                mu[i,:])).T@(self.X-self.mu[i,:])

            if self.covariance_type == "diag":
                self.C[i] = np.diag(self.C[i])


    def expectation(self):
        self.gamma = np.zeros((self.n, self.q))

        for i in range(self.q):
            try:
                self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self....
                    C[i])
            except:
                self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self....
                    C[i]+np.eye(self.C[i].shape[0])*1e-7)
        self.gamma = self.gamma/np.sum(self.gamma, axis=1).reshape(-1,1)

    def maximization(self):
        # print(np.sum(self.weights))
        self.Nq = np.sum(self.gamma, axis=0)
        self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)

        for i in range(self.q):
            self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self....
                mu[i,:])).T@(self.X-self.mu[i,:])

            if self.covariance_type == "diag":
                self.C[i] = np.diag(self.C[i])

        self.weights = self.Nq/self.n

    def log_likelihood(self, X_test):
        lk = 0
        n, d = X_test.shape
        for i in range(n):
            val = 0
            for j in range(self.q):
                try:
                    val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j...
                        ])
                except:
                    val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j...
                        ]+np.eye(self.C[j].shape[0])*1e-7)
            lk += np.log(val)

        return lk

    def indv_log_likelihood(self, X_test):
        n, d = X_test.shape
        lk = np.zeros((X_test.shape[0], 1))
        for i in range(n):
            val = 0
            for j in range(self.q):
                try:
                    val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j...
                        ])
                except:
                    val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j...
                        ]+np.eye(self.C[j].shape[0])*1e-7)
            lk[i] = np.log(val)

        return lk

    def gaussian_val(self, X_test):
        n, d = X_test.shape
        val = np.zeros((n, self.q))

        for i in range(self.q):
            val[:,i] = self.weights[i]*mvn.pdf(X_test, self.mu[i], self.C[i])

        return np.sum(val, axis=1)

class GMM_vl():
    def __init__(self, q):
        self.q = q

    def fit(self, X, epochs=100, covariance_type="diag", tol=1e-5):
        """
        X: n*d
        mu: q*d
        C: q*d*d
        gamma: n*q
        """
        self.n, self.d = X.shape
        self.X = X
        self.epochs = epochs
        self.covariance_type = covariance_type
        self.initialization()
        self.lglk_list = []
        for i in tqdm(range(self.epochs)):
            self.lglk_list.append(self.log_likelihood(self.X))
            self.expectation()
            self.maximization()
            new_lk = self.log_likelihood(self.X)
            diff = new_lk - self.lglk_list[-1]
            if  diff < tol:
                if diff < 0:
                    print("Difference is less than 0")
                    break

    def initialization(self):
    # kmeans = KMeans(n_clusters=self.q, random_state=0).fit(self.X)
        kmeans = KMeans(n_clusters=self.q).fit(self.X)
        labels = kmeans.labels_
        unique, counts = np.unique(labels, return_counts=True)

        self.subcomponents = unique.size
        self.gamma = np.eye(self.subcomponents)[labels]
        self.Nq = np.sum(self.gamma, axis=0)
        self.weights = counts/self.n
        self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
        self.C = np.zeros((self.subcomponents, self.d, self.d))

        for i in range(self.q):
            self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self....
                mu[i,:])).T@(self.X-self.mu[i,:])

            if self.covariance_type == "diag":
                self.C[i] = np.diag(np.diag(self.C[i]))


    def expectation(self):
        self.gamma = np.zeros((self.n, self.q))

        for i in range(self.q):
            try:
                self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self....
                    C[i])
            except:
                self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self.C[i]+np.eye(self.C[i].shape[0])*1e-3)
                self.gamma = self.gamma/np.sum(self.gamma, axis=1).reshape(-1,1)

    def maximization(self):
        # print(np.sum(self.weights))
        self.Nq = np.sum(self.gamma, axis=0)
        self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)

        for i in range(self.q):
            self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self.mu[i,:])).T@(self.X-self.mu[i,:])

        if self.covariance_type == "diag":
            self.C[i] = np.diag(np.diag(self.C[i]))

        self.weights = self.Nq/self.n

    def log_likelihood(self, X_test):
        lk = 0
        n, d = X_test.shape
        for i in range(n):
            val = 0
            for j in range(self.q):
                try:
                        val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j])
                except:
                    val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j]+np.eye(self.C[j].shape[0])*1e-3)
            lk += np.log(val)

        return lk

    def indv_log_likelihood(self, X_test):
        n, d = X_test.shape
        lk = np.zeros((X_test.shape[0], 1))
        for i in range(n):
            val = 0
            for j in range(self.q):
                try:
                        val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j])
                except:
                        val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j]+np.eye(self.C[j].shape[0])*1e-3)
            lk[i] = np.log(val)

        return lk

    def gaussian_val(self, X_test):
        n, d = X_test.shape
        val = np.zeros((n, self.q))

        for i in range(self.q):
            val[:,i] = self.weights[i]*mvn.pdf(X_test, self.mu[i], self.C[i])

        return np.sum(val, axis=1)

    def probab(self, df):
        df = pd.DataFrame(df)
        grouped_df = df.groupby(by=["class", "image"])
        for key, item in grouped_df:
            selected_df = grouped_df.get_group(key)
            X_select = selected_df.drop(["index", "image", "class"], axis=1).to_numpy()
            val = self.gaussian_val(X_select)
        print(val.shape)
```

The code used is as follows:

```
#!/usr/bin/env python
# coding: utf-8
#########################################################################
import os
import numpy as np
import pandas as pd
from tqdm import tqdm
from gmm import GMM_vl

get_ipython().run_line_magic('load_ext', 'autoreload')
get_ipython().run_line_magic('autoreload', '2')

#########################################################################
df = pd.read_csv("../datasets/2B/consolidated_train.csv")
X = df.drop(["class","image", "index"], axis=1).to_numpy()
print(X.shape)
df.head()

#########################################################################
classes = np.unique(df["class"])
gmm_list = []

for i in classes:
    gmm = GMM_vl(q=14)
    df_selected = df[df["class"]==i]

    X_selected = df_selected.drop(["class", "image", "index"], axis=1).to_numpy()
    gmm.fit(X_selected, epochs=20)
    gmm_list.append(gmm)

#########################################################################
gmm.probab(df_selected)

#########################################################################
gmm.gamma

#########################################################################
```