

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

1	Dataset 1A	2
2	Dataset 1B	9
2.1	Bayes Classification, GMM, Full Covariance	9
2.2	Bayes Classification, GMM, Diagonal Covariance	16
2.3	Bayes Classification, KNN	24
3	Dataset 2A	29
3.1	Bayes Classification, GMM, Full Covariance	29
3.2	Bayes Classification, GMM, Diagonal Covariance	37
4	Dataset 2B	43

1 Dataset 1A

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

```
1  #!/usr/bin/env python
2  # coding: utf-8
3  #####
4  import pandas as pd
5  import numpy as np
6  import matplotlib.pyplot as plt
7  from collections import Counter
8  get_ipython().run_line_magic('matplotlib', 'inline')
9
10 #####
11 from sklearn.metrics import confusion_matrix
12
13 #####
14 from sklearn.metrics import classification_report
15
16
17 # # Dataset 1a:
18 # ## Importing the train, test and cross validation data sets
19 #####
20 col_names=["x1","x2","y"]
21
22 #####
23 ## Train data
24 data1a=pd.read_csv("train.csv",names=col_names)
25
26 #####
27 data1a.head()
28
29 #####
30 data1a.isnull().sum()
31
32 #####
33 data1a.describe()
34
35 #####
36 ## Splitting the columns of train data
37
38 X1train=data1a["x1"]
39 X2train=data1a["x2"]
40 Ytrain=np.array(data1a["y"])
41 Xtrain=np.array(data1a.drop("y",axis=1))
42
43 #####
44 ## group labels
45 data1a["y"].unique()
46
47 #####
48 ## Importing the test and cross-validation data
49 data1a_dev=pd.read_csv("dev.csv",names=col_names)
50
51 #####
52 ## Function to split a given dataset into test and cross-validation
53
54 def create_datasets(data,cv_size):
55     data.sample(frac=1).reset_index(drop=True)
56     data_cv=data[0:cv_size]
57     data_test=data[cv_size:]
58     return(data_cv,data_test)
59
60 #####
61 def euclidean(p1,p2):
```

```

62     d=np.linalg.norm(np.array(p1)-np.array(p2))
63     return d
64
65 #####
66 def accuracy(y_pred,y_actual):
67     true_count=0
68     for i in range(len(y_pred)):
69         if y_pred[i]==y_actual[i]:
70             true_count+=1;
71     return(true_count/len(y_pred))
72
73 #####
74 data1a_dev.shape
75
76 #####
77 ## Splitting in the ratio 70:30 (cv:test)
78 data1a_cv,data1a_test=create_datasets(data1a_dev,84)
79
80
81 # ### Plotting the train data set
82 #####
83 X_cv=np.array(data1a_cv.drop("y",axis=1))
84 Y_cv=np.array(data1a_cv["y"])
85 X_test=np.array(data1a_test.drop("y",axis=1))
86 Y_test=np.array(data1a_test["y"])
87
88 plt.figure()
89 plt.scatter(X1train[Ytrain==0],X2train[Ytrain==0],label="y=0")
90 plt.scatter(X1train[Ytrain==1],X2train[Ytrain==1],label="y=1")
91 plt.scatter(X1train[Ytrain==2],X2train[Ytrain==2],label="y=2")
92 plt.scatter(X1train[Ytrain==3],X2train[Ytrain==3],label="y=3")
93 plt.legend()
94 plt.xlabel("X1")
95 plt.ylabel("X2")
96 plt.title("Scatter plot of data 1a")
97 plt.savefig("Scatter plot of data_1a.jpg")
98 plt.show()
99
100
101 # # K Nearest Neighbour Classifier for dataset 1a:
102 #####
103 def knn(x,y,test,k):
104     distances=[]
105     for i in range(len(x)):
106         d=euclidean(x[i],test)
107         l=(d,x[i],y[i])
108         distances.append(l)
109     distances.sort(key = lambda x:x[0])
110     count=Counter()
111     for i in distances[:k]:
112         count[i[2]]+=1
113     pred=count.most_common(1)[0][0]
114     return(distances[:k],pred)
115
116
117
118 # ### KNN on given cross-validation and test datasets:
119 #####
120 k_list=[1,7,15]
121 Accuracy_cv=[]
122 Accuracy_train=[]
123 Accuracy_test=[]
124
125 #####
126 ## iterating over k-values
127 for i in k_list:
128     ycv_pred=[]
129     for j in X_cv:
130         ycv_pred.append(knn(Xtrain,Ytrain,j,i)[1])

```

```

131     ytest_pred=[]
132     for j in X_test:
133         ytest_pred.append(knn(Xtrain,Ytrain,j,i)[1])
134     ytrain_pred=[]
135     for j in Xtrain:
136         ytrain_pred.append(knn(Xtrain,Ytrain,j,i)[1])
137     Accuracy_cv.append(accuracy(Y_cv,ycv_pred))
138     Accuracy_test.append(accuracy(Y_test,ytest_pred))
139     Accuracy_train.append(accuracy(Ytrain,ytrain_pred))
140
141     #####
142     accuracy_table_knn=pd.DataFrame(list(zip(k_list,Accuracy_train,Accuracy_cv,...
143         Accuracy_test)),columns=["k-value", "Accuracy train","Accuracy CV","Accuracy ...
144         test"])
145
146     #####
147     accuracy_table_knn
148
149     #####
150     cm=confusion_matrix(Ytrain,ytrain_pred,labels=[1.0,3.0,0.0,2.0])
151     cm2=confusion_matrix(Y_test,ytest_pred)
152
153     #####
154     from sklearn.metrics import ConfusionMatrixDisplay
155
156     #####
157     cmd=ConfusionMatrixDisplay(cm,display_labels=[0.0,1.0,2.0,3.0])
158     plt.figure()
159     cmd.plot()
160     plt.savefig("1a_cm_knn_train.jpg")
161
162     #####
163     cmd2=ConfusionMatrixDisplay(cm2,display_labels=[0.0,1.0,2.0,3.0])
164     plt.figure()
165     cmd2.plot()
166     plt.savefig("1a_cm_knn_test.jpg")
167
168     #####
169     # # Naive Bayes Classifier:
170     #####
171     def seperate_by_classval(data):
172         ## the target variable must be stored in a column named "y"
173         class_vals=list(data["y"].unique())
174         seperated=dict()
175         features=data.drop('y',axis=1)
176         Y=np.array(data["y"])
177         ## creates a key value corresponding to each class label
178         for i in class_vals:
179             seperated[i]=features[Y==i];
180         return(seperated)
181
182     #####
183     def priori(data):
184         seperated_data=seperate_by_classval(data)
185         probs=dict()
186         for i in seperated_data.keys():
187             probs[i]=len(seperated_data[i])/len(data);
188         return probs
189
190     #####
191     def mu_sigma(data):
192         seperated_data=seperate_by_classval(data)
193         mean=dict()
194         sigma={}
195         for i in list(seperated_data.keys()):
196             features=seperated_data[i]
197             mean[i]=[]
198             sigma[i]=[]
199             for j in range(seperated_data[i].shape[1]):

```

```

198         mean[i].append(np.mean(features.iloc[:,j]))
199         sigma[i].append(np.std(features.iloc[:,j]))
200     return(mean,sigma)
201
202 #####
203 def gauss_val(x,cov_matrix,mean):
204     x=np.array(x)
205     A=(x-mean)
206     B=np.linalg.inv(cov_matrix)
207     C=np.transpose(A)
208     det=np.linalg.det(cov_matrix)
209     AB=A.dot(B)
210     m=AB.dot(C)
211
212     exp_term=np.exp(-m/2)
213     d=2
214     return (exp_term/(2*np.pi*det**0.5))
215
216
217 # ## Seperating the data according to class label:
218 #####
219 seperated_data=seperate_by_classval(data1a)
220
221 #####
222 ### Labels:
223 labels=list(data1a["y"].unique())
224
225 #####
226 #####
227 accuracy_table_bayes=pd.DataFrame()
228 accuracy_table_bayes["method"]=["Ci=Cj=sigma**2*I...
229     ", "Ci=Cj=C", "Ci!=Cj"]
230
231 #####
232 accuracy_table_bayes["Train Accuracy"]=[0,0,0]
233 accuracy_table_bayes["CV accuracy"]=[0,0,0]
234 accuracy_table_bayes["Test Accuracy"]=[0,0,0]
235
236
237 # ### Case 1: Ci=Cj=sigma**2 * I
238 #####
239 sigma=mu_sigma(data1a)[1]
240
241 #####
242 sigma
243
244 var=0
245 for i in labels:
246     var+=sigma[i][0]**2+sigma[i][1]**2
247
248 var=var/(4*2)
249 #####
250 def predictor1(x):
251     pyi_x={}
252     pyi=priori(data1a)
253     means=mu_sigma(data1a)[0]
254     for i in labels:
255         pyi_x[i]=pyi[i]*gauss_val(x,var*np.eye(2),means[i])
256     val=sum(pyi_x.values())
257     p=0
258     for i in labels:
259         pyi_x[i]/=val
260         if pyi_x[i]>p:
261             prediction=i
262             p=pyi_x[i]
263
264
265     return(pyi_x,prediction)

```

```

266
267 #####
268 predictor1([-10,5])
269
270 #####
271 Y_nb1_cv=[]
272 Y_nb1_test=[]
273 Y_nb1_train=[]
274 for i in range(len(X_cv)):
275     Y_nb1_cv.append(predictor1(X_cv[i])[1])
276 for i in range(len(X_test)):
277     Y_nb1_test.append(predictor1(X_test[i])[1])
278 for i in range(len(Xtrain)):
279     Y_nb1_train.append(predictor1(Xtrain[i])[1])
280
281
282 #####
283 accuracy_table_bayes.iloc[0,1:]=[accuracy(Y_nb1_train,Ytrain),accuracy(Y_nb1_cv,...
    Y_cv),accuracy(Y_nb1_test,Y_test)]
284
285
286 # ### Confusion Matrix
287 #####
288 cm_nb_train=confusion_matrix(Y_nb1_train,Ytrain)
289 cm_nb_test=confusion_matrix(Y_nb1_test,Y_test)
290
291 #####
292 len(Y_nb1_train)
293
294 #####
295 cmd_nb_train=ConfusionMatrixDisplay(cm_nb_train,display_labels=[0.0,1.0,2.0,3.0])
296 plt.figure()
297 cmd_nb_train.plot()
298 plt.savefig("1a_cm_nb_train.jpg")
299
300 #####
301 cmd_nb_test=ConfusionMatrixDisplay(cm_nb_test,display_labels=[0.0,1.0,2.0,3.0])
302 plt.figure()
303 cmd_nb_test.plot()
304 plt.savefig("1a_cm_nb_test.jpg")
305
306
307 # ### Case 2: Covariance matrix is same for all the classes:
308 #####
309 cov_matrix={}
310 for i in labels:
311     cov_matrix[i]=np.cov(seperated_data[i],rowvar=False)
312
313 #####
314 cov_matrix
315
316 #####
317 C=np.zeros((2,2))
318 for i in labels:
319     C+=cov_matrix[i]
320 C/=4
321
322 #####
323 C
324
325 #####
326 def predictor2(x):
327     pyi_x={}
328     pyi=priori(data1a)
329     means=mu_sigma(data1a)[0]
330     for i in labels:
331         pyi_x[i]=pyi[i]*gauss_val(x,C,means[i])
332     val=sum(pyi_x.values())
333     p=0

```

```

334     for i in labels:
335         pyi_x[i]/=val
336         if pyi_x[i]>p:
337             prediction=i
338             p=pyi_x[i]
339
340
341     return(pyi_x,prediction)
342
343 #####
344 Y_nb2_cv=[]
345 Y_nb2_test=[]
346 Y_nb2_train=[]
347 for i in range(len(X_cv)):
348     Y_nb2_cv.append(predictor2(X_cv[i])[1])
349 for i in range(len(X_test)):
350     Y_nb2_test.append(predictor2(X_test[i])[1])
351 for i in range(len(Xtrain)):
352     Y_nb2_train.append(predictor2(Xtrain[i])[1])
353
354
355 #####
356 accuracy_table_bayes.iloc[1,1:]=[accuracy(Y_nb2_train,Ytrain),accuracy(Y_nb2_cv,...
    Y_cv),accuracy(Y_nb2_test,Y_test)]
357
358
359 # ### Case 3: Covariance matrix is different for all the classes:
360 #####
361 def predictor3(x):
362     pyi_x={}
363     pyi=priori(data1a)
364     means=mu_sigma(data1a)[0]
365     for i in labels:
366         pyi_x[i]=pyi[i]*gauss_val(x,cov_matrix[i],means[i])
367     val=sum(pyi_x.values())
368     p=0
369     for i in labels:
370         pyi_x[i]/=val
371         if pyi_x[i]>p:
372             prediction=i
373             p=pyi_x[i]
374
375
376     return(pyi_x,prediction)
377
378
379 #####
380 predictor3([5,5])
381
382 #####
383 Y_nb3_cv=[]
384 Y_nb3_test=[]
385 Y_nb3_train=[]
386 for i in range(len(X_cv)):
387     Y_nb3_cv.append(predictor3(X_cv[i])[1])
388 for i in range(len(X_test)):
389     Y_nb3_test.append(predictor3(X_test[i])[1])
390 for i in range(len(Xtrain)):
391     Y_nb3_train.append(predictor3(Xtrain[i])[1])
392
393
394 #####
395 accuracy_table_bayes.iloc[2,1:]=[accuracy(Y_nb3_train,Ytrain),accuracy(Y_nb3_cv,...
    Y_cv),accuracy(Y_nb3_test,Y_test)]
396
397 #####
398 accuracy_table_bayes
399
400

```



```

401 # ### Confusion matrix for naive bayes classifier:
402 #####
403
404
405
406 # ### Decision boundary plot for knn:
407 #####
408 min1,max1=data1a["x1"].min()-1,data1a["x1"].max()+1
409 min2,max2=data1a["x2"].min()-1,data1a["x2"].max()+1
410
411 #####
412 resolution=0.5
413 x1grid=np.arange(min1,max1,resolution)
414 x2grid=np.arange(min2,max2,resolution)
415
416 #####
417 xx,yy=np.meshgrid(x1grid,x2grid)
418
419 #####
420 r1,r2=xx.flatten(),yy.flatten()
421 r1,r2=r1.reshape((len(r1),1)),r2.reshape((len(r2),1))
422
423 #####
424 grid=np.hstack((r1,r2))
425
426 #####
427 yhat_knn_1=[]
428 for i in range(len(grid)):
429     yhat_knn_1.append(knn(Xtrain,Ytrain,grid[i,:],1)[1])
430
431 #####
432 len(grid)
433
434 #####
435 yhat_knn_1=np.array(yhat_knn_1)
436
437 #####
438 zz=yhat_knn_1.reshape(xx.shape)
439
440 #####
441 data1a["y"].unique()
442
443 #####
444 plt.figure()
445 plt.contourf(xx,yy,zz,alpha=0.5,cmap="Paired")
446 plt.scatter(X1train[Ytrain==0],X2train[Ytrain==0],label="y=0",c="Blue")
447 plt.scatter(X1train[Ytrain==1],X2train[Ytrain==1],label="y=1",c="Green")
448 plt.scatter(X1train[Ytrain==2],X2train[Ytrain==2],label="y=2",c="Orange")
449 plt.scatter(X1train[Ytrain==3],X2train[Ytrain==3],label="y=3",c='red')
450 plt.legend()
451 plt.xlabel("X1")
452 plt.ylabel("X2")
453 plt.title("Decision region plot of data 1a, knn classifier")
454 plt.savefig("1a_knn_decision_region.jpg")
455 plt.show()
456
457 #####
458 grid
459
460 #####
461 yhat_nb=[]
462 for i in range(len(grid)):
463     yhat_nb.append(predictor1(grid[i,:])[1])
464
465 #####
466 yhat_nb=np.array(yhat_nb)
467
468 #####
469 zz_nb=yhat_nb.reshape(xx.shape)

```

```

470
471 #####
472 plt.figure()
473 plt.contourf(xx,yy,zz_nb,alpha=0.5,cmap="Paired")
474 plt.scatter(X1train[Ytrain==0],X2train[Ytrain==0],label="y=0",c="Blue")
475 plt.scatter(X1train[Ytrain==1],X2train[Ytrain==1],label="y=1",c="Green")
476 plt.scatter(X1train[Ytrain==2],X2train[Ytrain==2],label="y=2",c="Orange")
477 plt.scatter(X1train[Ytrain==3],X2train[Ytrain==3],label="y=3",c='red')
478 plt.legend()
479 plt.xlabel("X1")
480 plt.ylabel("X2")
481 plt.title("Decision region plot of data 1a,naive-bayes classifier")
482 plt.savefig("1a_nb_case1_decisionregion.jpg")
483 plt.show()

```

2 Dataset 1B

2.1 Bayes Classification, GMM, Full Covariance

The GMM full covariance model code is as follows:

```

1  #!/usr/bin/env python
2  # coding: utf-8
3  #####
4  import time
5  import pickle
6  import numpy as np
7  import pandas as pd
8  from gmm import GMM
9  import matplotlib.pyplot as plt
10 from multiprocessing import Pool
11 from collections import defaultdict
12 from scipy.stats import multivariate_normal as mvn
13 from sklearn.model_selection import train_test_split
14
15 plt.rcParams["font.size"] = 18
16 plt.rcParams["axes.grid"] = True
17 plt.rcParams["figure.figsize"] = 8,6
18 plt.rcParams['font.serif'] = "Cambria"
19 plt.rcParams['font.family'] = "serif"
20
21 get_ipython().run_line_magic('load_ext', 'autoreload')
22 get_ipython().run_line_magic('autoreload', '2')
23
24 #####
25 df = pd.read_csv("../datasets/1B/train.csv", header=None)
26 X = df.drop(2, axis=1).to_numpy()
27 df.head()
28
29 #####
30 classes = np.unique(df[2])
31 gmm_list = defaultdict(list)
32 q_list = list(range(2,10))
33
34 for i in classes:
35     df_select = df[df[2]==i]
36     X_select = df_select.drop(2, axis=1).to_numpy()
37     for q in q_list:
38         gmm = GMM(q=q)
39         gmm.fit(X_select)
40         gmm_list[i].append(gmm)
41
42 #####
43 import pickle
44 fin = open("1b_gmm_results", "wb")
45 pickle.dump(gmm_list, fin)

```

```

46 fin.close()
47
48 #####
49 df_test = pd.read_csv("../datasets/1B/dev.csv", header=None)
50 df_cv = df_test.sample(frac=0.7)
51 X_cv = df_cv.drop(2, axis=1).to_numpy()
52 display(df_cv.head())
53 df_test = df_test.drop(df_cv.index)
54 X_test = df_test.drop(2, axis=1).to_numpy()
55 df_test.head()
56
57 #####
58 classes = np.unique(df[2])
59 q_list = list(range(2,10))
60
61 accuracy_list = []
62 cv_accuracy_list = []
63 test_accuracy_list = []
64 for i in range(len(q_list)):
65     gmm0 = gmm_list[0.0][i]
66     gmm1 = gmm_list[1.0][i]
67     gmm2 = gmm_list[2.0][i]
68
69     # Training
70     a = gmm0.indv_log_likelihood(X)
71     b = gmm1.indv_log_likelihood(X)
72     c = gmm2.indv_log_likelihood(X)
73
74     d = np.hstack((a, b, c))
75     pred = np.argmax(d, axis=1)
76     accuracy_list.append(np.sum(pred == df[2])/df[2].size)
77
78     # CV
79     a = gmm0.indv_log_likelihood(X_cv)
80     b = gmm1.indv_log_likelihood(X_cv)
81     c = gmm2.indv_log_likelihood(X_cv)
82
83     d = np.hstack((a, b, c))
84     pred = np.argmax(d, axis=1)
85     cv_accuracy_list.append(np.sum(pred == df_cv[2])/df_cv[2].size)
86
87     # Testing
88     a = gmm0.indv_log_likelihood(X_test)
89     b = gmm1.indv_log_likelihood(X_test)
90     c = gmm2.indv_log_likelihood(X_test)
91
92     d = np.hstack((a, b, c))
93     pred = np.argmax(d, axis=1)
94     test_accuracy_list.append(np.sum(pred == df_test[2])/df_test[2].size)
95
96 #####
97 plt.plot(q_list, accuracy_list, '-.')
98 plt.title("Accuracy across varying Q")
99 plt.xlabel("Q for each class")
100 plt.ylabel("Accuracy")
101 plt.show()
102
103 plt.plot(q_list, cv_accuracy_list, '-.')
104 plt.title("CV Accuracy across varying Q")
105 plt.xlabel("Q for each class")
106 plt.ylabel("Accuracy")
107 plt.show()
108
109 plt.plot(q_list, test_accuracy_list, '-.')
110 plt.title("Test Accuracy across varying Q")
111 plt.xlabel("Q for each class")
112 plt.ylabel("Accuracy")
113 plt.show()
114

```

```

115 #####
116 fout = open("1b_gmm_results", "rb")
117 gmm_list = pickle.load(fout)
118 fout.close()
119
120 #####
121 x, y = np.mgrid[-3:3:30j, -3:3:30j]
122 xy = np.column_stack([x.flat, y.flat])
123
124 z0_val = gmm_list[0.0][3].indv_log_likelihood(xy)
125 z1_val = gmm_list[1.0][3].indv_log_likelihood(xy)
126 z2_val = gmm_list[2.0][3].indv_log_likelihood(xy)
127
128 d = np.hstack((z0_val, z1_val, z2_val))
129 classes = np.argmax(d, axis=1)
130 classes = classes.reshape(x.shape)
131
132 plt.figure()
133 df.plot.scatter(0, 1, c=[color_list[int(i)] for i in df[2]], alpha=1)
134 plt.contourf(x, y, classes, 2, colors=color_list, alpha=0.1)
135 plt.contour(x, y, classes, 2, colors=color_list, alpha=1)
136 plt.title("Decision Boundaries - Full Covariance")
137 plt.show()
138
139 #####
140 classes = np.unique(df[2])
141 q_list = list(range(2,10))
142
143 # color_list = np.random.rand(len(classes), 3)
144 color_list = ["springgreen", "mediumturquoise", "palevioletred"]
145 x, y = np.mgrid[-3:3:30j, -3:3:30j]
146 xy = np.column_stack([x.flat, y.flat])
147
148 z0 = gmm_list[0.0][3].gaussian_val(xy)
149 z0 = z0.reshape(x.shape)
150
151 z1 = gmm_list[1.0][3].gaussian_val(xy)
152 z1 = z1.reshape(x.shape)
153
154 z2 = gmm_list[2.0][3].gaussian_val(xy)
155 z2 = z2.reshape(x.shape)
156
157 plt.figure()
158 df.plot.scatter(0, 1, c=[color_list[int(i)] for i in df[2]], alpha=1)
159 plt.contour(x, y, z0, levels=np.logspace(-2,2,20), colors=color_list[0])
160 plt.contour(x, y, z1, levels=np.logspace(-2,2,20), colors=color_list[1])
161 plt.contour(x, y, z2, levels=np.logspace(-2,2,20), colors=color_list[2])
162 plt.title("Contour Plot - Full Covariance")
163
164 #####
165 x, y = np.mgrid[-3:3:30j, -3:3:30j]
166 xy = np.column_stack([x.flat, y.flat])
167
168 z0_val = gmm_list[0.0][3].indv_log_likelihood(xy)
169 z1_val = gmm_list[1.0][3].indv_log_likelihood(xy)
170 z2_val = gmm_list[2.0][3].indv_log_likelihood(xy)
171
172 d = np.hstack((z0_val, z1_val, z2_val))
173 classes = np.argmax(d, axis=1)
174 classes = classes.reshape(x.shape)
175
176 plt.figure()
177 df.plot.scatter(0, 1, c=[color_list[int(i)] for i in df[2]], alpha=1)
178 plt.contourf(x, y, classes, 2, colors=color_list, alpha=0.1)
179 plt.contour(x, y, classes, 2, colors=color_list, alpha=1)
180 plt.title("Decision Boundaries - Full Covariance")
181 plt.show()
182
183 #####

```

```

184 x, y = np.mgrid[-3:3:30j, -3:3:30j]
185 xy = np.column_stack([x.flat, y.flat])
186
187 z0_val = gmm_list[0.0][3].indv_log_likelihood(xy)
188 z1_val = gmm_list[1.0][3].indv_log_likelihood(xy)
189 z2_val = gmm_list[2.0][3].indv_log_likelihood(xy)
190
191 d = np.hstack((z0_val, z1_val, z2_val))
192 classes = np.argmax(d, axis=1)
193 classes = classes.reshape(x.shape)
194
195 plt.figure()
196 df.plot.scatter(0, 1, c=[color_list[int(i)] for i in df[2]], alpha=1)
197 plt.contourf(x, y, classes, 2, colors=color_list, alpha=0.1)
198 plt.contour(x, y, classes, 2, colors=color_list, alpha=1)
199 plt.contour(x, y, z0, levels=np.logspace(-2,2,20), colors=color_list[0])
200 plt.contour(x, y, z1, levels=np.logspace(-2,2,20), colors=color_list[1])
201 plt.contour(x, y, z2, levels=np.logspace(-2,2,20), colors=color_list[2])
202 plt.title("Decision Boundaries + Contours - Full Covariance")
203 plt.show()
204
205 #####
206 import seaborn as sns
207 from sklearn.metrics import confusion_matrix
208
209 classes = np.unique(df[2])
210 q_list = list(range(2,10))
211
212 gmm0 = gmm_list[0.0][3]
213 gmm1 = gmm_list[1.0][3]
214 gmm2 = gmm_list[2.0][3]
215
216 # Training
217 a = gmm0.indv_log_likelihood(X)
218 b = gmm1.indv_log_likelihood(X)
219 c = gmm2.indv_log_likelihood(X)
220
221 d = np.hstack((a, b, c))
222 pred = np.argmax(d, axis=1)
223 conf_mat = confusion_matrix(pred, df[2])
224 plt.figure()
225 sns.heatmap(conf_mat, annot=True)
226 plt.title("Training Confusion Matrix")
227 plt.xlabel("Predicted Class")
228 plt.ylabel("Actual Class")
229 plt.show()
230
231 # CV
232 a = gmm0.indv_log_likelihood(X_cv)
233 b = gmm1.indv_log_likelihood(X_cv)
234 c = gmm2.indv_log_likelihood(X_cv)
235
236 d = np.hstack((a, b, c))
237 pred = np.argmax(d, axis=1)
238 conf_mat = confusion_matrix(pred, df_cv[2])
239 plt.figure()
240 sns.heatmap(conf_mat, annot=True)
241 plt.title("CV Confusion Matrix")
242 plt.xlabel("Predicted Class")
243 plt.ylabel("Actual Class")
244 plt.show()
245
246 # Testing
247 a = gmm0.indv_log_likelihood(X_test)
248 b = gmm1.indv_log_likelihood(X_test)
249 c = gmm2.indv_log_likelihood(X_test)
250
251 d = np.hstack((a, b, c))
252 pred = np.argmax(d, axis=1)

```

```

253 conf_mat = confusion_matrix(pred, df_test[2])
254 plt.figure()
255 sns.heatmap(conf_mat, annot=True)
256 plt.title("Testing Confusion Matrix")
257 plt.xlabel("Predicted Class")
258 plt.ylabel("Actual Class")
259 plt.show()

```

The GMM class module is as follows:

```

1  import numpy as np
2  from tqdm import tqdm
3  from sklearn.cluster import KMeans
4  from scipy.stats import multivariate_normal as mvn
5  import pandas as pd
6
7  class GMM():
8      def __init__(self, q):
9          self.q = q
10
11     def fit(self, X, covariance_type="diag", tol=1e-5):
12         """
13         X: n*d
14         mu: q*d
15         C: q*d*d
16         gamma: n*q
17         """
18         self.n, self.d = X.shape
19         self.X = X
20         self.covariance_type = covariance_type
21         self.initialization()
22         self.lglk_list = []
23         for i in tqdm(range(100)):
24             self.lglk_list.append(self.log_likelihood(self.X))
25             self.expectation()
26             self.maximization()
27             new_lk = self.log_likelihood(self.X)
28             diff = new_lk - self.lglk_list[-1]
29             if diff < tol:
30                 if diff < 0: print("Difference is less than 0")
31                 break
32
33
34     def initialization(self):
35         # kmeans = KMeans(n_clusters=self.q, random_state=0).fit(self.X)
36         kmeans = KMeans(n_clusters=self.q).fit(self.X)
37         labels = kmeans.labels_
38         unique, counts = np.unique(labels, return_counts=True)
39
40         self.subcomponents = unique.size
41         self.gamma = np.eye(self.subcomponents)[labels]
42         self.Nq = np.sum(self.gamma, axis=0)
43         self.weights = counts/self.n
44         self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
45         self.C = np.zeros((self.subcomponents, self.d, self.d))
46
47         for i in range(self.q):
48             self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self.mu[i,:]).T@
49                                     (self.X-self.mu[i,:]))
50
51             if self.covariance_type == "diag":
52                 self.C[i] = np.diag(self.C[i])
53
54     def expectation(self):
55         self.gamma = np.zeros((self.n, self.q))
56
57         for i in range(self.q):
58             try:

```

```

59         self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self....
        C[i])
60     except:
61         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)
62     self.gamma = self.gamma/np.sum(self.gamma, axis=1).reshape(-1,1)
63
64     def maximization(self):
65         # print(np.sum(self.weights))
66         self.Nq = np.sum(self.gamma, axis=0)
67         self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
68
69         for i in range(self.q):
70             self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self....
            mu[i,:])).T@(self.X-self.mu[i,:])
71
72             if self.covariance_type == "diag":
73                 self.C[i] = np.diag(self.C[i])
74
75         self.weights = self.Nq/self.n
76
77     def log_likelihood(self, X_test):
78         lk = 0
79         n, d = X_test.shape
80         for i in range(n):
81             val = 0
82             for j in range(self.q):
83                 try:
84                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j]...
                    ])
85                 except:
86                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j]...
                    ]+np.eye(self.C[j].shape[0])*1e-7)
87             lk += np.log(val)
88
89         return lk
90
91     def indv_log_likelihood(self, X_test):
92         n, d = X_test.shape
93         lk = np.zeros((X_test.shape[0], 1))
94         for i in range(n):
95             val = 0
96             for j in range(self.q):
97                 try:
98                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j]...
                    ])
99                 except:
100                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j]...
                    ]+np.eye(self.C[j].shape[0])*1e-7)
101             lk[i] = np.log(val)
102
103         return lk
104
105     def gaussian_val(self, X_test):
106         n, d = X_test.shape
107         val = np.zeros((n, self.q))
108
109         for i in range(self.q):
110             val[:,i] = self.weights[i]*mvn.pdf(X_test, self.mu[i], self.C[i])
111
112         return np.sum(val, axis=1)
113
114     class GMM_v1():
115         def __init__(self, q):
116             self.q = q
117
118         def fit(self, X, epochs=100, covariance_type="diag", tol=1e-5):
119             """
120             X: n*d

```

```

121     mu: q*d
122     C: q*d*d
123     gamma: n*q
124     """
125     self.n, self.d = X.shape
126     self.X = X
127     self.epochs = epochs
128     self.covariance_type = covariance_type
129     self.initialization()
130     self.lglk_list = []
131     for i in tqdm(range(self.epochs)):
132         self.lglk_list.append(self.log_likelihood(self.X))
133         self.expectation()
134         self.maximization()
135         new_lk = self.log_likelihood(self.X)
136         diff = new_lk - self.lglk_list[-1]
137         if diff < tol:
138             if diff < 0:
139                 print("Difference is less than 0")
140                 break
141
142     def initialization(self):
143         # kmeans = KMeans(n_clusters=self.q, random_state=0).fit(self.X)
144         kmeans = KMeans(n_clusters=self.q).fit(self.X)
145         labels = kmeans.labels_
146         unique, counts = np.unique(labels, return_counts=True)
147
148         self.subcomponents = unique.size
149         self.gamma = np.eye(self.subcomponents)[labels]
150         self.Nq = np.sum(self.gamma, axis=0)
151         self.weights = counts/self.n
152         self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
153         self.C = np.zeros((self.subcomponents, self.d, self.d))
154
155         for i in range(self.q):
156             self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self.mu[i,:]).T@self.X-self.mu[i,:])
157
158             if self.covariance_type == "diag":
159                 self.C[i] = np.diag(np.diag(self.C[i]))
160
161
162     def expectation(self):
163         self.gamma = np.zeros((self.n, self.q))
164
165         for i in range(self.q):
166             try:
167                 self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self.C[i])
168             except:
169                 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)
170                 self.gamma = self.gamma/np.sum(self.gamma, axis=1).reshape(-1,1)
171
172     def maximization(self):
173         # print(np.sum(self.weights))
174         self.Nq = np.sum(self.gamma, axis=0)
175         self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
176
177         for i in range(self.q):
178             self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self.mu[i,:]).T@self.X-self.mu[i,:])
179
180             if self.covariance_type == "diag":
181                 self.C[i] = np.diag(np.diag(self.C[i]))
182
183         self.weights = self.Nq/self.n
184
185     def log_likelihood(self, X_test):

```



```

186         lk = 0
187         n, d = X_test.shape
188         for i in range(n):
189             val = 0
190             for j in range(self.q):
191                 try:
192                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self....
193                         C[j])
194                 except:
195                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j]...
196                         ]+np.eye(self.C[j].shape[0])*1e-3)
197             lk += np.log(val)
198
199         return lk
200
201     def indiv_log_likelihood(self, X_test):
202         n, d = X_test.shape
203         lk = np.zeros((X_test.shape[0], 1))
204         for i in range(n):
205             val = 0
206             for j in range(self.q):
207                 try:
208                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self....
209                         C[j])
210                 except:
211                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self....
212                         C[j]+np.eye(self.C[j].shape[0])*1e-3)
213             lk[i] = np.log(val)
214
215         return lk
216
217     def gaussian_val(self, X_test):
218         n, d = X_test.shape
219         val = np.zeros((n, self.q))
220
221         for i in range(self.q):
222             val[:,i] = self.weights[i]*mvn.pdf(X_test, self.mu[i], self.C[i])
223
224         return np.sum(val, axis=1)
225
226     def probab(self, df):
227         df = pd.DataFrame(df)
228         grouped_df = df.groupby(by=["class", "image"])
229         for key, item in grouped_df:
230             selected_df = grouped_df.get_group(key)
231             X_select = selected_df.drop(["index", "image", "class"], axis=1)....
232                 to_numpy()
233             val = self.gaussian_val(X_select)
234         print(val.shape)

```

2.2 Bayes Classification, GMM, Diagonal Covariance

The GMM diagonal covariance model code is as follows:

```

1  #!/usr/bin/env python
2  # coding: utf-8
3  #####
4  import numpy as np
5  import pandas as pd
6  import matplotlib.pyplot as plt
7
8  #####
9  import pickle
10 from scipy.stats import multivariate_normal as mvn
11
12 #####
13 plt.style.use('science')

```

```

14 plt.rcParams['font.size'] = 18
15 plt.rcParams['axes.grid'] = True
16 plt.rcParams['grid.linestyle'] = (5,9)
17 plt.rcParams['figure.figsize'] = 8,6
18
19 #####
20 import statistics as sts
21 from sklearn.model_selection import train_test_split
22
23 #####
24 from sklearn.cluster import KMeans
25
26 #####
27 from separate_class import Separate
28
29 #####
30 ds2_train = pd.read_csv("train2.csv", header = None)
31
32 #####
33 ds2_test = pd.read_csv("dev2.csv", header = None)
34
35 #####
36 ds2_train.head()
37
38 #####
39 ds2_test.head()
40
41 #####
42 ds2_train.describe()
43
44 #####
45 ds2_test.describe()
46
47 #####
48 X_train = ds2_train.iloc[:,2]
49 Y_train = ds2_train.iloc[:,2]
50
51 #####
52 def gauss(x,m,c,d):
53     return((1/((2*np.pi)*(d/2))*np.sqrt(np.linalg.det(c))))*np.exp(-(x-m).T@np....
54         linalg.inv(c)@(x-m)/2))
55
56 #####
57 sep_train = Separate(ds2_train)
58
59 #####
60 classes_dat = sep_train.classes
61
62 #####
63 X_sep_train = sep_train.get_x()
64 Y_sep_train = sep_train.get_y()
65 dat_sep_train = sep_train.get_separated_data()
66
67 #####
68 pd.DataFrame(X_sep_train[0]).to_csv("X_sep_train.csv")
69
70 #####
71 def likelihood(x,m,W,c):
72     s = 0
73     m = np.array(m)
74     l = len(W)
75     for i in range(l):
76         s += W[i]*gauss(x,m[i],c[i],d)
77     return(s)
78
79 #####
80 plt.rcParams["font.size"] = 18
81 plt.rcParams["axes.grid"] = True
82 plt.rcParams["figure.figsize"] = 8,6

```

```

82 plt.rcParams['font.serif'] = "Cambria"
83 plt.rcParams['font.family'] = "serif"
84
85 #####
86 import time
87
88 #####
89 from multiprocessing import Pool
90
91 #####
92 class_ = 2
93 d = 2
94 threshold = 0.01
95
96 #####
97 # parameter estimation for Bayesian GMM - EM method
98 # training and obtaining parameters for different hparameter values
99
100 #for Q in q:
101 def f(Q):
102     L_old = 0
103     L_new = 1
104     L = []
105     difference = L_new - L_old
106     cond = True
107     # initialization
108     while (cond==True):
109         kmeans = KMeans(n_clusters = Q, random_state = 0).fit(X_sep_train[class_])
110         labels = kmeans.labels_
111         N = np.array([])
112         for i in range(Q):
113             N = np.append(N,np.count_nonzero(labels==i))
114         cond = True in (ele ==1 for ele in N)
115
116     Nt = np.sum(N)
117     w = N/Nt
118     gamma = []
119     for i in range(Q):
120         gamma.append(np.multiply(labels==i,1))
121     mu = kmeans.cluster_centers_
122     n = len(X_sep_train[0])
123     C = np.zeros((Q,d,d))
124     for i in range(Q):
125         for j in range(n):
126             C[i] += gamma[i][j]*np.outer(X_sep_train[class_].iloc[j] - mu[i],...
127                                         X_sep_train[class_].iloc[j] - mu[i])
128             C[i] = np.diag(np.diag(C[i]/N[i]))
129
130     L_old = 0
131     for i in range(n):
132         L_old += np.log(likelihood(X_sep_train[class_].iloc[i],mu,w,C))
133
134     while (difference > threshold):
135
136         #Expectation
137         den = np.zeros(n)
138         for i in range(n):
139             for j in range(Q):
140                 den[i] += w[j]*gaus(X_sep_train[class_].iloc[i],np.array(mu)[j],C[j...
141                                     ],d)
142
143         gamma = np.zeros((Q,n))
144         for i in range(n):
145             for j in range(Q):
146                 gamma[j][i] = w[j]*gaus(X_sep_train[class_].iloc[i], np.array(mu)[j...
147                                     ], C[j],d)/den[i]
148
149         # maximization step

```

```

148     N = []
149     for i in range(Q):
150         N.append(np.sum(gamma[i]))
151     Nt = np.sum(N)
152     w = N/Nt
153     mu = np.divide(gamma@X_sep_train[class_],np.array([N,N]).T)
154     C = np.zeros((Q,d,d))
155     for i in range(Q):
156         for j in range(n):
157             C[i] += gamma[i][j]*np.outer(X_sep_train[class_].iloc[j] - mu.iloc[...
158                 i],X_sep_train[class_].iloc[j] - mu.iloc[i])
159             C[i] = np.diag(np.diag(C[i]/N[i]))
160
161     L_new = 0
162     for i in range(n):
163         L_new += np.log(likelihood(X_sep_train[class_].iloc[i],mu.to_numpy(),w,...
164             C))
165         #print(L_new,L_old)
166         difference = L_new - L_old
167         L_old = L_new
168         L.append(L_new)
169     return([mu,w,C,L])
170     #L_q.append(L)
171     #add accuracy and confusion matrix
172
173 #####
174 pool = Pool(processes=4)
175
176 #####
177 from multiprocessing import cpu_count
178
179 #####
180 cpu_count()
181
182 #####
183 q = list(range(2,10))
184
185 #####
186 t1 = time.time()
187 params = pool.map(f,q)
188 t2 = time.time()
189
190 #####
191 class_2_param = params
192 get_ipython().run_line_magic('store', 'class_2_param')
193
194 #####
195 dbfile = open("class2_1b",'ab')
196 pickle.dump(class_2_param,dbfile)
197 dbfile.close()
198
199 #####
200 dbfile = open("class0_1b",'rb')
201 class_0_param = pickle.load(dbfile)
202 dbfile.close()
203
204 #####
205 dbfile = open("class1_1b",'rb')
206 class_1_param = pickle.load(dbfile)
207 dbfile.close()
208
209 #####
210 dbfile = open("class2_1b",'rb')
211 class_2_param = pickle.load(dbfile)
212 dbfile.close()
213
214 #####
215 parameters = [class_0_param,class_1_param, class_2_param]

```

```

215 #####
216 import accuracy
217
218 #####
219 #predicting training data - selecting max likelihood value
220 d = 2
221 acc_train = []
222 for Q in range(len(q)):
223     y_Pred = []
224     for i in range(600):
225         lst = []
226         for j in range(featvec_length+1):
227             lst.append(likelihood(X_train.iloc[i],parameters[j][Q][0],parameters[j...
228                               ][Q][1],parameters[j][Q][2]))
229             y_Pred.append(lst.index(max(lst)))
230             #print(y_Pred[i])
231         acc_calc = accuracy.Confusion_matrix(y_Pred,Y_train)
232         acc_train.append(acc_calc.accuracy)
233
234 #####
235 df = pd.DataFrame(list(zip(q,acc_train)),columns=["Hyperparameter Value", "Accuracy...
236                "])
237
238 #####
239 acc_train = pd.read_csv("acc1b_train.csv",index_col = 0)
240
241 #####
242 plt.plot(acc_train)
243
244 #####
245 pd.crosstab(ds2_train.iloc[:,featvec_length],y_Pred)
246
247 #####
248 ds2_test = pd.read_csv("dev2.csv", header = None)
249
250 #####
251 X_cv,X_test,y_cv,y_test = train_test_split(ds2_test.iloc[:,2],ds2_test.iloc[:,2], ...
252         test_size=0.3, random_state=0)
253
254 #####
255 acc_cv = []
256 for Q in range(len(q)):
257     y_Pred = []
258     for i in range(len(X_cv)):
259         lst = []
260         for j in range(featvec_length+1):
261             lst.append(likelihood(X_cv.iloc[i],parameters[j][Q][0],parameters[j][Q...
262                               ][1],parameters[j][Q][2]))
263             y_Pred.append(lst.index(max(lst)))
264             #print(y_Pred[i])
265         acc_calc = accuracy.Confusion_matrix(y_Pred,y_cv)
266         acc_cv.append(acc_calc.accuracy)
267
268 #####
269 df = pd.DataFrame(list(zip(q,acc_cv)),columns=["Hyperparameter Value", "Accuracy"])
270 df.to_csv("acc1b_cv.csv")
271
272 #####
273 acc_cv = pd.read_csv("acc1b_cv.csv",index_col=0)
274
275 #####
276 plt.plot(q,acc_train.iloc[:,1],label = "Training Data")
277 plt.plot(q,acc_cv.iloc[:,1],label = "Validation Data")
278 plt.xlabel("No. of Gaussian Components")
279 plt.ylabel("Accuracy")
280 plt.title("Accuracy with hyperparameter values on Training and Validation data")

```

```

280 plt.legend()
281 plt.savefig("acc_1b.png")
282 plt.show()
283
284 #####
285 acc_cv.index(max(acc_cv))
286
287 #####
288 q[3]
289
290 #####
291 acc_train.index(max(acc_train))
292
293 #####
294 # best model, q = 5
295 Q = 3
296 y_Pred = []
297 for i in range(len(X_test)):
298     lst = []
299     for j in range(featvec_length+1):
300         lst.append(likelihood(X_test.iloc[i],parameters[j][Q][0],parameters[j][Q...
301             ][1],parameters[j][Q][2]))
302     y_Pred.append(lst.index(max(lst)))
303     #print(y_Pred[i])
304 acc_calc = accuracy.Confusion_matrix(y_Pred,y_test)
305 acc_test = acc_calc.accuracy
306
307 #####
308 acc_test
309
310 #####
311 Q=3
312 d=2
313 YPredTrain = []
314 for i in range(len(X_train)):
315     lst = []
316     for j in range(3):
317         lst.append(likelihood(X_train.iloc[i],parameters[j][Q][0],parameters[j][Q...
318             ][1],parameters[j][Q][2]))
319     YPredTrain.append(lst.index(max(lst)))
320
321 #####
322 pd.DataFrame(YPredTrain).to_csv("YPredTrain.csv")
323 pd.DataFrame(YPredCV).to_csv("YPredCV.csv")
324 pd.DataFrame(YPredTest).to_csv("YPredTest.csv")
325 pd.DataFrame(yGridPred).to_csv("YPredGrid.csv")
326 pd.DataFrame(acc_cv).to_csv("acc_cv.csv")
327 pd.DataFrame(acc_train).to_csv("acc_train.csv")
328
329 #####
330 YPredCV = []
331 for i in range(len(X_cv)):
332     lst = []
333     for j in range(3):
334         lst.append(likelihood(X_cv.iloc[i],parameters[j][Q][0],parameters[j][Q][1],...
335             parameters[j][Q][2]))
336     YPredCV.append(lst.index(max(lst)))
337
338 YPredTest = []
339 for i in range(len(X_test)):
340     lst = []
341     for j in range(3):
342         lst.append(likelihood(X_test.iloc[i],parameters[j][Q][0],parameters[j][Q...
343             ][1],parameters[j][Q][2]))
344     YPredTest.append(lst.index(max(lst)))
345
346 #####
347 import seaborn as sns
348

```

```

345 #####
346 from sklearn.metrics import confusion_matrix
347
348 #####
349 conf_mat = confusion_matrix(YPredTrain,Y_train)
350 plt.figure()
351 sns.heatmap(conf_mat, annot=True)
352 plt.title("Training Confusion Matrix")
353 plt.xlabel("Predicted Class")
354 plt.ylabel("Actual Class")
355 plt.savefig("conf_train1b.png")
356 plt.show()
357
358 #####
359 conf_Train = ac_train.get_matrix()
360
361 #####
362 pd.DataFrame(conf_Train).to_csv("conf_train_1b.csv")
363
364 #####
365 ac_test = accuracy.Confusion_matrix(YPredTest,y_test)
366 conf_Test = ac_test.get_matrix()
367 pd.DataFrame(conf_Train).to_csv("conf_test_1b.csv")
368
369 #####
370 conf_mat = confusion_matrix(YPredTest,y_test)
371 plt.figure()
372 sns.heatmap(conf_mat, annot=True)
373 plt.title("Test Confusion Matrix")
374 plt.xlabel("Predicted Class")
375 plt.ylabel("Actual Class")
376 plt.savefig("conf_test1b.png")
377 plt.show()
378
379 #####
380 for class_val in range(3):
381     row_idx = np.where(ds2_train.iloc[:,featvec_length]==class_val)
382     plt.scatter(np.array(ds2_train)[row_idx,0],np.array(ds2_train)[row_idx,1])
383 plt.show()
384
385 #####
386 Q = 3
387 d = 2
388
389 #####
390 min_x1 = min(X_train[0])
391 max_x1 = max(X_train[0])
392 min_x2 = min(X_train[1])
393 max_x2 = max(X_train[1])
394
395 x1_range = np.linspace(min_x1,max_x1)
396 x2_range = np.linspace(min_x2,max_x2)
397
398 X1,X2 = np.meshgrid(x1_range,x2_range)
399
400 x1,x2 = X1.flatten(),X2.flatten()
401 x1,x2 = x1.reshape(len(x1),1),x2.reshape(len(x2),1)
402 grid = np.hstack((x1,x2))
403
404 #####
405 yGridPred = []
406 for i in range(len(grid)):
407     lst = []
408     for j in range(3):
409         lst.append(likelihood(grid[i],parameters[j][Q][0],parameters[j][Q][1],...
410                             parameters[j][Q][2]))
411     yGridPred.append(lst.index(max(lst)))
412 yGridPred = np.array(yGridPred).reshape(X1.shape)

```

```

413 #####
414 plt.contourf(X1,X2,yGridPred)
415 for class_val in range(3):
416     row_idx = np.where(ds2_train.iloc[:,featvec_length]==class_val)
417     plt.scatter(np.array(ds2_train)[row_idx,0],np.array(ds2_train)[row_idx,1],label...
418               = "Class " + str(class_val))
419 plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
420 plt.xlabel("X1")
421 plt.ylabel("X2")
422 plt.title("Decision region plot with training data superposed")
423 plt.savefig("decisionReg_ds2.png")
424 plt.show()
425 #####
426
427 Q=3
428 d=2
429 x, y = np.mgrid[-3:3:30j, -3:3:30j]
430 xy = np.column_stack([x.flat, y.flat])
431 z0_val = np.zeros(len(xy))
432 z1_val = np.zeros(len(xy))
433 z2_val = np.zeros(len(xy))
434
435 for i in range(len(xy)):
436     lst = np.array((len(xy)))
437     z0_val[i] = likelihood(xy[i],parameters[0][Q][0],parameters[0][Q][1],parameters...
438                       [0][Q][2])
439     z1_val[i] = likelihood(xy[i],parameters[1][Q][0],parameters[1][Q][1],parameters...
440                       [1][Q][2])
441     z2_val[i] = likelihood(xy[i],parameters[2][Q][0],parameters[2][Q][1],parameters...
442                       [2][Q][2])
443 d = np.hstack((z0_val.reshape(900,-2),z1_val.reshape(900,-2),z2_val.reshape(900,-2)...
444               ))
445 classes = np.argmax(d,axis=1)
446 classes = classes.reshape(x.shape)
447
448 #####
449 def gaussian_val(X_test,w,mu,C):
450     n, d = X_test.shape
451     val = np.zeros((n, 5))
452
453     for i in range(5):
454         val[:,i] = w[i]*mvn.pdf(X_test, mu.iloc[i], C[i])
455
456     return np.sum(val, axis=1)
457
458 #####
459 color_list = ["springgreen", "mediumturquoise", "palevioletred"]
460
461 #####
462 z0 = np.zeros(len(xy))
463 z1 = np.zeros(len(xy))
464 z2 = np.zeros(len(xy))
465
466 z0 = gaussian_val(xy,parameters[0][Q][1],parameters[0][Q][0],parameters[0][Q][2])
467 z1 = gaussian_val(xy,parameters[1][Q][1],parameters[1][Q][0],parameters[1][Q][2])
468 z2 = gaussian_val(xy,parameters[2][Q][1],parameters[2][Q][0],parameters[2][Q][2])
469
470 z0 = z0.reshape(x.shape)
471
472 z1 = z1.reshape(x.shape)
473
474 z2 = z2.reshape(x.shape)
475
476 #####
477 plt.figure()

```



```

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

```

The accuracy module is as follows:

```

1 import numpy as np
2 import pandas as pd
3 from sklearn.metrics import confusion_matrix
4 class Confusion_matrix():
5     def __init__(self,y_pred, y_orig):
6         self.pred = y_pred
7         self.original = y_orig
8         self.length = len(y_pred)
9         self.compare = y_pred == y_orig
10        self.accuracy = np.sum(self.compare)/self.length
11        self.classes = pd.Series(y_orig).unique()[0]
12
13    def get_matrix(self):
14        #mat = np.zeros((1,1))
15        #conf_matrix = pd.crosstab(self.original,self.pred,rownames=["actual"],...
16        #                           colnames = ["predicted"])
17        mat = confusion_matrix(self.original,self.pred)
18        return(mat)

```

2.3 Bayes Classification, KNN

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 #####
5 import pandas as pd
6 import numpy as np
7 import matplotlib.pyplot as plt
8 from collections import Counter
9 get_ipython().run_line_magic('matplotlib', 'inline')
10
11
12 # # For dataset 1b:
13 #####
14 col_names=["x1","x2","y"]
15 data1b=pd.read_csv("train1b.csv",names=col_names)
16 Xtrain_1=data1b["x1"]
17 Xtrain_2=data1b["x2"]
18 Ytrain=np.array(data1b["y"])
19 Xtrain=np.array(data1b.drop("y",axis=1))
20
21 #####
22 data1b_dev=pd.read_csv("dev1b.csv",names=col_names)
23
24 #####
25 plt.figure()
26 plt.scatter(Xtrain_1[Ytrain==0],Xtrain_2[Ytrain==0],label="y=0")
27 plt.scatter(Xtrain_1[Ytrain==1],Xtrain_2[Ytrain==1],label="y=1")
28 plt.scatter(Xtrain_1[Ytrain==2],Xtrain_2[Ytrain==2],label="y=2")
29 plt.legend()
30 plt.xlabel("X1")
31 plt.ylabel("X2")

```

```

32 plt.title("Scatter plot of data 1b")
33 plt.savefig("Scatter plot of data 1b.jpg")
34 plt.show()
35
36 #####
37 ## Shuffles a provided data set and splits it into cross-validation and test ...
   dataset
38
39 def create_datasets(data,cv_size):
40     data.sample(frac=1).reset_index(drop=True)
41     test_size=len(data)-cv_size
42     data_test=data[0:test_size]
43     data_cv=data[test_size:]
44     return(data_cv,data_test)
45
46 #####
47 ## Calculates accuracy of the model
48
49 def accuracy(y_pred,y_actual):
50     true_count=0
51     for i in range(len(y_pred)):
52         if y_pred[i]==y_actual[i]:
53             true_count+=1;
54     return(true_count/len(y_pred))
55
56 #####
57 ## Calculates euclidean distance between two vector points
58
59 def euclidean(p1,p2):
60     d=np.linalg.norm(np.array(p1)-np.array(p2))
61     return d
62
63 #####
64 data1b_cv,data1b_test=create_datasets(data1b_dev,50)
65
66 #####
67 data1b_test=data1b_test.append(data1b.iloc[595:, :]);
68
69 #####
70 def knn(x,y,test,k):
71     distances=[]
72     for i in range(len(x)):
73         d=euclidean(x[i],test)
74         l=(d,x[i],y[i])
75         distances.append(l)
76     distances.sort(key = lambda x:x[0])
77     count=Counter()
78     for i in distances[:k]:
79         count[i[2]]+=1
80     pred=count.most_common(1)[0][0]
81     return(distances[:k],pred)
82
83
84 #####
85 k_list=[1,7,15]
86 Accuracyknn_cv=[]
87 Accuracyknn_train=[]
88 Accuracyknn_test=[]
89
90 #####
91 X_cv=np.array(data1b_cv.drop("y",axis=1))
92 Y_cv=np.array(data1b_cv["y"])
93 X_test=np.array(data1b_test.drop("y",axis=1))
94 Y_test=np.array(data1b_test["y"])
95
96 ## iterating over k-values
97 for i in k_list:
98     ycv_pred=[]
99     for j in X_cv:

```

```

100     ycv_pred.append(knn(Xtrain,Ytrain,j,i)[1])
101     ytest_pred=[]
102     for j in X_test:
103         ytest_pred.append(knn(Xtrain,Ytrain,j,i)[1])
104     ytrain_pred=[]
105     for j in Xtrain:
106         ytrain_pred.append(knn(Xtrain,Ytrain,j,i)[1])
107     Accuracyknn_cv.append(accuracy(Y_cv,ycv_pred))
108     Accuracyknn_test.append(accuracy(Y_test,ytest_pred))
109     Accuracyknn_train.append(accuracy(Ytrain,ytrain_pred))
110
111     #####
112     accuracy_table_KNN=pd.DataFrame(list(zip(k_list,Accuracyknn_train,Accuracyknn_cv,...
113         Accuracyknn_test)),columns=["k-value", "Accuracy train","Accuracy CV","Accuracy ...
114         test"])
115
116     #####
117     ytrainpred_1=[]
118     ytestpred_1=[]
119     for i in Xtrain:
120         ytrainpred_1.append(knn(Xtrain,Ytrain,i,1)[1])
121     for i in X_test:
122         ytestpred_1.append(knn(Xtrain,Ytrain,i,1)[1])
123
124     #####
125     from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
126
127     #####
128     cm_knn_train=confusion_matrix(ytrainpred_1,Ytrain)
129     cm_knn_test=confusion_matrix(ytestpred_1,Y_test)
130     cmd_knn_train=ConfusionMatrixDisplay(cm_knn_train,display_labels=[0.0,1.0,2.0])
131     plt.figure()
132     cmd_knn_train.plot()
133     plt.savefig("1b_cm_knn_train.jpg")
134
135     #####
136     cmd_knn_test=ConfusionMatrixDisplay(cm_knn_test,display_labels=[0.0,1.0,2.0])
137     plt.figure()
138     cmd_knn_test.plot()
139     plt.savefig("1b_cm_knn_test.jpg")
140
141     #####
142     accuracy_table_KNN
143
144     # ## Bayes classifier with KNN to calculate class conditional probabilities
145     #####
146     ## Seperating the rows by class values
147
148     def seperate_by_classval(data):
149         ## the target variable must be stored in a column named "y"
150         class_vals=list(data["y"].unique())
151         seperated=dict()
152         features=data.drop('y',axis=1)
153         Y=np.array(data["y"])
154         ## creates a key value corresponding to each class label
155         for i in class_vals:
156             seperated[i]=features[Y==i];
157         return(seperated)
158
159     #####
160     ## Calculates the prior probability of classes and returns a dictionary such that
161     ## probs[i] is the prior probability of class i
162
163     def priori(data):
164         seperated_data=seperate_by_classval(data)
165         probs=dict()
166         for i in seperated_data.keys():
167             probs[i]=len(seperated_data[i])/len(data);

```

```

167     return probs
168
169 #####
170 ## Calculates the class-conditional probability p(x/yi) using knn method
171 ## the input x is the data points for a particular class i
172 ## Each row of knn_list consists of a nearest neighbour and its distance from the ...
173     test point
174 ## prob is the class conditional probability p(x/yi)
175
176 def knn_prob(x,test,k):
177     distances=[]
178     for i in range(len(x)):
179         d=euclidean(x[i],test)
180         l=(d,x[i])
181         distances.append(l)
182     distances.sort(key = lambda x:x[0])
183     knn_list=distances[:k]
184     r=knn_list[-1][0]
185     prob=k/(np.pi*r**2*len(x))
186     return(knn_list,prob)
187
188 #####
189 ## This uses the above code blocks to evaluate p(yi/x) for all the classes
190 ## Returns a dictionary probabs, such that probabs[i] is the p(yi/x)
191 ## also returns label which is the class label corresponding to the maximum p(yi/x)
192
193 def predictor(train_data,k,test_data):
194     X_train=seperate_by_classval(train_data)
195     p_y=priori(train_data)
196     p=0
197     probabs={}
198     for i in list(priori(train_data).keys()):
199         p_yi=p_y[i]
200         X_traini=X_train[i]
201         px_yi=knn_prob(np.array(X_traini),test_data,k)[1]
202         pyi_x=px_yi*p_yi
203         probabs[i]=pyi_x
204         if probabs[i]>p:
205             p=probabs[i]
206             label=i
207     sum_vals=sum(list(probabs.values()))
208     for i in probabs.keys():
209         probabs[i]=probabs[i]/sum_vals
210     return(probabs,label)
211
212 # ### Predicting for k=10 and k=20
213 #####
214 ypred10_cv=[]
215 ypred10_test=[]
216 ypred20_cv=[]
217 ypred20_test=[]
218 ypred10_train=[]
219 ypred20_train=[]
220 for i in range(len(data1b_cv)):
221     ypred10_cv.append(predictor(data1b,10,data1b_cv.iloc[i,:-1])[1])
222     ypred20_cv.append(predictor(data1b,20,data1b_cv.iloc[i,:-1])[1])
223 for i in range(len(data1b_test)):
224     ypred10_test.append(predictor(data1b,10,data1b_test.iloc[i,:-1])[1])
225     ypred20_test.append(predictor(data1b,20,data1b_test.iloc[i,:-1])[1])
226 for i in range(len(data1b)):
227     ypred10_train.append(predictor(data1b,10,data1b.iloc[i,:-1])[1])
228     ypred20_train.append(predictor(data1b,20,data1b.iloc[i,:-1])[1])
229
230 #####
231 accuracy_table=pd.DataFrame()
232 accuracy_table["k-value"]=[10,20]
233 accuracy_table["Train data"]=[accuracy(ypred10_train,list(data1b.iloc[:,-1])),...
234                                accuracy(ypred20_train,list(data1b.iloc[:,-1]))]

```

```

234 accuracy_table["CV data"]=[accuracy(ypred10_cv,list(data1b_cv.iloc[:,-1])),accuracy...
    (ypred20_cv,list(data1b_cv.iloc[:,-1]))]
235 accuracy_table["Test data"]=[accuracy(ypred10_test,list(data1b_test.iloc[:,-1])),...
    accuracy(ypred20_test,list(data1b_test.iloc[:,-1]))]
236
237 #####
238 accuracy_table
239
240 #####
241 cm_nb_train=confusion_matrix(ypred10_train,Ytrain)
242 cm_nb_test=confusion_matrix(ypred10_test,Y_test)
243 cmd_nb_train=ConfusionMatrixDisplay(cm_nb_train,display_labels=[0.0,1.0,2.0])
244 plt.figure()
245 cmd_nb_train.plot()
246 plt.savefig("1b_cm_nb_train.jpg")
247
248 #####
249 cmd_nb_test=ConfusionMatrixDisplay(cm_nb_test,display_labels=[0.0,1.0,2.0])
250 plt.figure()
251 cmd_nb_test.plot()
252 plt.savefig("1b_cm_nb_test.jpg")
253
254
255 # ### Decision region plots:
256 #####
257 min1,max1=data1b["x1"].min()-1,data1b["x1"].max()+1
258 min2,max2=data1b["x2"].min()-1,data1b["x2"].max()+1
259
260 resolution=0.5
261 x1grid=np.arange(min1,max1,resolution)
262 x2grid=np.arange(min2,max2,resolution)
263
264 xx,yy=np.meshgrid(x1grid,x2grid)
265
266 r1,r2=xx.flatten(),yy.flatten()
267 r1,r2=r1.reshape((len(r1),1)),r2.reshape((len(r2),1))
268
269 grid=np.hstack((r1,r2))
270
271 #####
272 yhat_knn=[]
273 for i in range(len(grid)):
274     yhat_knn.append(knn(Xtrain,Ytrain,grid[i,:],10)[1])
275
276 #####
277 yhat_knn=np.array(yhat_knn)
278 zz_knn=yhat_knn.reshape(xx.shape)
279
280 #####
281 plt.figure()
282 plt.contourf(xx,yy,zz_knn,alpha=0.6,cmap="Paired")
283 plt.scatter(Xtrain_1[Ytrain==0],Xtrain_2[Ytrain==0],label="y=0",c="Blue")
284 plt.scatter(Xtrain_1[Ytrain==1],Xtrain_2[Ytrain==1],label="y=1",c="red")
285 plt.scatter(Xtrain_1[Ytrain==2],Xtrain_2[Ytrain==2],label="y=2",c="Brown")
286 plt.legend()
287 plt.xlabel("X1")
288 plt.ylabel("X2")
289 plt.title("Decision region plot of data 1b, knn classifier")
290 plt.savefig("1b_knn_decision_region.jpg")
291 plt.show()
292
293 #####
294 yhat_nb=[]
295 for i in range(len(grid)):
296     yhat_nb.append(predictor(data1b,10,grid[i,:])[1])
297 yhat_nb=np.array(yhat_nb)
298 zz_nb=yhat_nb.reshape(xx.shape)
299
300 #####

```

```

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

```

3 Dataset 2A

3.1 Bayes Classification, GMM, Full Covariance

The GMM full covariance model code is as follows:

```

1  #!/usr/bin/env python
2  # coding: utf-8
3
4  #####
5  import time
6  import numpy as np
7  import pandas as pd
8  from gmm import GMM
9  from tqdm import tqdm
10 import matplotlib.pyplot as plt
11 from multiprocessing import Pool
12 from collections import defaultdict
13 from scipy.stats import multivariate_normal as mvn
14 from sklearn.model_selection import train_test_split
15
16 plt.rcParams["font.size"] = 18
17 plt.rcParams["axes.grid"] = True
18 plt.rcParams["figure.figsize"] = 8,6
19 plt.rcParams['font.serif'] = "Cambria"
20 plt.rcParams['font.family'] = "serif"
21
22 get_ipython().run_line_magic('load_ext', 'autoreload')
23 get_ipython().run_line_magic('autoreload', '2')
24
25
26 #####
27 df = pd.read_csv("../datasets/2A/consolidated_train.csv")
28 X = df.drop("class", axis=1).to_numpy()
29 df.head()
30
31
32 #####
33 classes = np.unique(df["class"])
34 gmm_list = defaultdict(list)
35 q_list = list(range(2,23))
36
37 for i in classes:
38     print("="*50)
39     df_select = df[df["class"]==i]
40     X_select = df_select.drop("class", axis=1).to_numpy()
41     for q in q_list:
42         gmm = GMM(q=q)
43         gmm.fit(X_select)
44         gmm_list[i].append(gmm)
45
46
47 #####
48 import pickle

```

```

49 fin = open("2a_gmm_results", "wb")
50 pickle.dump(gmm_list, fin)
51 fin.close()
52
53
54 #####
55 df_test = pd.read_csv("../datasets/2A/consolidated_dev.csv")
56 df_cv = df_test.sample(frac=0.7)
57 X_cv = df_cv.drop("class", axis=1).to_numpy()
58 display(df_cv.head())
59 df_test = df_test.drop(df_cv.index)
60 X_test = df_test.drop("class", axis=1).to_numpy()
61 display(df_test.head())
62
63
64 #####
65 accuracy_list = []
66 test_accuracy_list = []
67 for i in tqdm(range(len(q_list))):
68     gmm0 = gmm_list[0][i]
69     gmm1 = gmm_list[1][i]
70     gmm2 = gmm_list[2][i]
71     gmm3 = gmm_list[3][i]
72     gmm4 = gmm_list[4][i]
73
74     # Training
75     a = gmm0.indv_log_likelihood(X)
76     b = gmm1.indv_log_likelihood(X)
77     c = gmm2.indv_log_likelihood(X)
78     d = gmm3.indv_log_likelihood(X)
79     e = gmm4.indv_log_likelihood(X)
80
81     f = np.hstack((a, b, c, d, e))
82     pred = np.argmax(f, axis=1)
83     accuracy_list.append(np.sum(pred == df["class"])/df["class"].size)
84
85     # Testing
86     a = gmm0.indv_log_likelihood(X_test)
87     b = gmm1.indv_log_likelihood(X_test)
88     c = gmm2.indv_log_likelihood(X_test)
89     d = gmm3.indv_log_likelihood(X_test)
90     e = gmm4.indv_log_likelihood(X_test)
91
92     f = np.hstack((a, b, c, d, e))
93     pred = np.argmax(f, axis=1)
94     test_accuracy_list.append(np.sum(pred == df_test["class"])/df_test["class"]....
95         size)
96
97 #####
98 plt.plot(q_list, accuracy_list, '-.')
99 plt.title("Accuracy across varying Q")
100 plt.xlabel("Q for each class")
101 plt.ylabel("Accuracy")
102 plt.show()
103
104 plt.plot(q_list, cv_accuracy_list, '-.')
105 plt.title("CV Accuracy across varying Q")
106 plt.xlabel("Q for each class")
107 plt.ylabel("Accuracy")
108 plt.show()
109
110 plt.plot(q_list, test_accuracy_list, '-.')
111 plt.title("Test Accuracy across varying Q")
112 plt.xlabel("Q for each class")
113 plt.ylabel("Accuracy")
114 plt.show()
115
116

```

```

117 #####
118 import seaborn as sns
119 from sklearn.metrics import confusion_matrix
120
121 best_model = np.argmax(acc["Sum"])
122
123 gmm0 = gmm_list[0][best_model]
124 gmm1 = gmm_list[1][best_model]
125 gmm2 = gmm_list[2][best_model]
126 gmm3 = gmm_list[3][best_model]
127 gmm4 = gmm_list[4][best_model]
128
129 # Training
130 a = gmm0.indv_log_likelihood(X)
131 b = gmm1.indv_log_likelihood(X)
132 c = gmm2.indv_log_likelihood(X)
133 d = gmm3.indv_log_likelihood(X)
134 e = gmm4.indv_log_likelihood(X)
135
136 f = np.hstack((a, b, c, d, e))
137 pred = np.argmax(f, axis=1)
138 conf_mat = confusion_matrix(pred, df["class"])
139 plt.figure()
140 sns.heatmap(conf_mat, annot=True)
141 plt.title("Training Confusion Matrix")
142 plt.xlabel("Predicted Class")
143 plt.ylabel("Actual Class")
144 plt.show()
145
146 # CV
147 a = gmm0.indv_log_likelihood(X_cv)
148 b = gmm1.indv_log_likelihood(X_cv)
149 c = gmm2.indv_log_likelihood(X_cv)
150 d = gmm3.indv_log_likelihood(X_cv)
151 e = gmm4.indv_log_likelihood(X_cv)
152
153 f = np.hstack((a, b, c, d, e))
154 pred = np.argmax(f, axis=1)
155 conf_mat = confusion_matrix(pred, df_cv["class"])
156 plt.figure()
157 sns.heatmap(conf_mat, annot=True)
158 plt.title("Validation Confusion Matrix")
159 plt.xlabel("Predicted Class")
160 plt.ylabel("Actual Class")
161 plt.show()
162
163 # Testing
164 a_test = gmm0.indv_log_likelihood(X_test)
165 b_test = gmm1.indv_log_likelihood(X_test)
166 c_test = gmm2.indv_log_likelihood(X_test)
167 d_test = gmm3.indv_log_likelihood(X_test)
168 e_test = gmm4.indv_log_likelihood(X_test)
169
170 f_test = np.hstack((a_test, b_test, c_test, d_test, e_test))
171 pred_test = np.argmax(f_test, axis=1)
172 conf_mat = confusion_matrix(pred_test, df_test["class"])
173 plt.figure()
174 sns.heatmap(conf_mat, annot=True)
175 plt.title("Testing Confusion Matrix")
176 plt.xlabel("Predicted Class")
177 plt.ylabel("Actual Class")
178 plt.show()
179
180
181 #####
182 import seaborn as sns
183 from sklearn.metrics import confusion_matrix
184
185 gmm0 = gmm_list[0][0]

```



```

186 gmm1 = gmm_list[1][0]
187 gmm2 = gmm_list[2][0]
188 gmm3 = gmm_list[3][4]
189 gmm4 = gmm_list[4][3]
190
191 # Training
192 a = gmm0.indv_log_likelihood(X)
193 b = gmm1.indv_log_likelihood(X)
194 c = gmm2.indv_log_likelihood(X)
195 d = gmm3.indv_log_likelihood(X)
196 e = gmm4.indv_log_likelihood(X)
197
198 f = np.hstack((a, b, c, d, e))
199 pred = np.argmax(f, axis=1)
200 conf_mat = confusion_matrix(pred, df["class"])
201 plt.figure()
202 sns.heatmap(conf_mat, annot=True)
203 plt.title("Training Confusion Matrix")
204 plt.xlabel("Predicted Class")
205 plt.ylabel("Actual Class")
206 plt.show()
207
208 # CV
209 a = gmm0.indv_log_likelihood(X_cv)
210 b = gmm1.indv_log_likelihood(X_cv)
211 c = gmm2.indv_log_likelihood(X_cv)
212 d = gmm3.indv_log_likelihood(X_cv)
213 e = gmm4.indv_log_likelihood(X_cv)
214
215 f = np.hstack((a, b, c, d, e))
216 pred = np.argmax(f, axis=1)
217 conf_mat = confusion_matrix(pred, df_cv["class"])
218 plt.figure()
219 sns.heatmap(conf_mat, annot=True)
220 plt.title("Validation Confusion Matrix")
221 plt.xlabel("Predicted Class")
222 plt.ylabel("Actual Class")
223 plt.show()
224
225 # Testing
226 a_test = gmm0.indv_log_likelihood(X_test)
227 b_test = gmm1.indv_log_likelihood(X_test)
228 c_test = gmm2.indv_log_likelihood(X_test)
229 d_test = gmm3.indv_log_likelihood(X_test)
230 e_test = gmm4.indv_log_likelihood(X_test)
231
232 f_test = np.hstack((a_test, b_test, c_test, d_test, e_test))
233 pred_test = np.argmax(f_test, axis=1)
234 conf_mat = confusion_matrix(pred_test, df_test["class"])
235 plt.figure()
236 sns.heatmap(conf_mat, annot=True)
237 plt.title("Testing Confusion Matrix")
238 plt.xlabel("Predicted Class")
239 plt.ylabel("Actual Class")
240 plt.show()

```

The GMM class module is as follows:

```

1 import numpy as np
2 from tqdm import tqdm
3 from sklearn.cluster import KMeans
4 from scipy.stats import multivariate_normal as mvn
5 import pandas as pd
6
7 class GMM():
8     def __init__(self, q):
9         self.q = q
10
11     def fit(self, X, covariance_type="diag", tol=1e-5):

```

```

12     """
13     X: n*d
14     mu: q*d
15     C: q*d*d
16     gamma: n*q
17     """
18     self.n, self.d = X.shape
19     self.X = X
20     self.covariance_type = covariance_type
21     self.initialization()
22     self.lglk_list = []
23     for i in tqdm(range(100)):
24         self.lglk_list.append(self.log_likelihood(self.X))
25         self.expectation()
26         self.maximization()
27         new_lk = self.log_likelihood(self.X)
28         diff = new_lk - self.lglk_list[-1]
29         if diff < tol:
30             if diff < 0: print("Difference is less than 0")
31             break
32
33
34     def initialization(self):
35         # kmeans = KMeans(n_clusters=self.q, random_state=0).fit(self.X)
36         kmeans = KMeans(n_clusters=self.q).fit(self.X)
37         labels = kmeans.labels_
38         unique, counts = np.unique(labels, return_counts=True)
39
40         self.subcomponents = unique.size
41         self.gamma = np.eye(self.subcomponents)[labels]
42         self.Nq = np.sum(self.gamma, axis=0)
43         self.weights = counts/self.n
44         self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
45         self.C = np.zeros((self.subcomponents, self.d, self.d))
46
47         for i in range(self.q):
48             self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self.mu[i,:])
49                                     .T@self.X-self.mu[i,:])
50
51             if self.covariance_type == "diag":
52                 self.C[i] = np.diag(self.C[i])
53
54     def expectation(self):
55         self.gamma = np.zeros((self.n, self.q))
56
57         for i in range(self.q):
58             try:
59                 self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self.C[i])
60             except:
61                 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)
62         self.gamma = self.gamma/np.sum(self.gamma, axis=1).reshape(-1,1)
63
64     def maximization(self):
65         # print(np.sum(self.weights))
66         self.Nq = np.sum(self.gamma, axis=0)
67         self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
68
69         for i in range(self.q):
70             self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self.mu[i,:])
71                                     .T@self.X-self.mu[i,:])
72
73             if self.covariance_type == "diag":
74                 self.C[i] = np.diag(self.C[i])
75
76         self.weights = self.Nq/self.n

```

```

77     def log_likelihood(self, X_test):
78         lk = 0
79         n, d = X_test.shape
80         for i in range(n):
81             val = 0
82             for j in range(self.q):
83                 try:
84                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j...
                        ])
85                 except:
86                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j...
                        ]+np.eye(self.C[j].shape[0])*1e-7)
87             lk += np.log(val)
88
89         return lk
90
91     def indiv_log_likelihood(self, X_test):
92         n, d = X_test.shape
93         lk = np.zeros((X_test.shape[0], 1))
94         for i in range(n):
95             val = 0
96             for j in range(self.q):
97                 try:
98                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j...
                        ])
99                 except:
100                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j...
                        ]+np.eye(self.C[j].shape[0])*1e-7)
101             lk[i] = np.log(val)
102
103         return lk
104
105     def gaussian_val(self, X_test):
106         n, d = X_test.shape
107         val = np.zeros((n, self.q))
108
109         for i in range(self.q):
110             val[:,i] = self.weights[i]*mvn.pdf(X_test, self.mu[i], self.C[i])
111
112         return np.sum(val, axis=1)
113
114 class GMM_v1():
115     def __init__(self, q):
116         self.q = q
117
118     def fit(self, X, epochs=100, covariance_type="diag", tol=1e-5):
119         """
120         X: n*d
121         mu: q*d
122         C: q*d*d
123         gamma: n*q
124         """
125         self.n, self.d = X.shape
126         self.X = X
127         self.epochs = epochs
128         self.covariance_type = covariance_type
129         self.initialization()
130         self.lglk_list = []
131         for i in tqdm(range(self.epochs)):
132             self.lglk_list.append(self.log_likelihood(self.X))
133             self.expectation()
134             self.maximization()
135             new_lk = self.log_likelihood(self.X)
136             diff = new_lk - self.lglk_list[-1]
137             if diff < tol:
138                 if diff < 0:
139                     print("Difference is less than 0")
140                     break
141

```

```

142 def initialization(self):
143     # kmeans = KMeans(n_clusters=self.q, random_state=0).fit(self.X)
144     kmeans = KMeans(n_clusters=self.q).fit(self.X)
145     labels = kmeans.labels_
146     unique, counts = np.unique(labels, return_counts=True)
147
148     self.subcomponents = unique.size
149     self.gamma = np.eye(self.subcomponents)[labels]
150     self.Nq = np.sum(self.gamma, axis=0)
151     self.weights = counts/self.n
152     self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
153     self.C = np.zeros((self.subcomponents, self.d, self.d))
154
155     for i in range(self.q):
156         self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self....
            mu[i,:])).T@(self.X-self.mu[i,:])
157
158         if self.covariance_type == "diag":
159             self.C[i] = np.diag(np.diag(self.C[i]))
160
161
162 def expectation(self):
163     self.gamma = np.zeros((self.n, self.q))
164
165     for i in range(self.q):
166         try:
167             self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self....
                C[i])
168         except:
169             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)
170             self.gamma = self.gamma/np.sum(self.gamma, axis=1).reshape(-1,1)
171
172 def maximization(self):
173     # print(np.sum(self.weights))
174     self.Nq = np.sum(self.gamma, axis=0)
175     self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
176
177     for i in range(self.q):
178         self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self....
            mu[i,:])).T@(self.X-self.mu[i,:])
179
180         if self.covariance_type == "diag":
181             self.C[i] = np.diag(np.diag(self.C[i]))
182
183     self.weights = self.Nq/self.n
184
185 def log_likelihood(self, X_test):
186     lk = 0
187     n, d = X_test.shape
188     for i in range(n):
189         val = 0
190         for j in range(self.q):
191             try:
192                 val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self....
                    C[j])
193             except:
194                 val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j]...
                    ]+np.eye(self.C[j].shape[0])*1e-3)
195         lk += np.log(val)
196
197     return lk
198
199 def indiv_log_likelihood(self, X_test):
200     n, d = X_test.shape
201     lk = np.zeros((X_test.shape[0], 1))
202     for i in range(n):
203         val = 0
204         for j in range(self.q):

```

```

205         try:
206             val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self....
                C[j])
207         except:
208             val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self....
                C[j]+np.eye(self.C[j].shape[0])*1e-3)
209         lk[i] = np.log(val)
210
211     return lk
212
213 def gaussian_val(self, X_test):
214     n, d = X_test.shape
215     val = np.zeros((n, self.q))
216
217     for i in range(self.q):
218         val[:,i] = self.weights[i]*mvn.pdf(X_test, self.mu[i], self.C[i])
219
220     return np.sum(val, axis=1)
221
222 def probab(self, df):
223     df = pd.DataFrame(df)
224     grouped_df = df.groupby(by=["class", "image"])
225     for key, item in grouped_df:
226         selected_df = grouped_df.get_group(key)
227         X_select = selected_df.drop(["index", "image", "class"], axis=1)....
            to_numpy()
228         val = self.gaussian_val(X_select)
229     print(val.shape)

```

The utils script is as follows:

```

1  import os
2  import numpy as np
3  import pandas as pd
4  from tqdm import tqdm
5
6  def get_consolidated_data2A(classes_present):
7      df = pd.DataFrame()
8      df_test = pd.DataFrame()
9      for i in classes_present:
10         df_new = pd.read_csv("../datasets/2A/"+i+"/train.csv")
11         # df_new = pd.read_csv("../datasets/2A/"+i+"/train.csv", nrows=182)
12         df_new["image_names"] = classes_present[i]
13         df_new = df_new.rename(columns={"image_names": "class"})
14         df = df.append(df_new)
15
16         df_new_test = pd.read_csv("../datasets/2A/"+i+"/dev.csv")
17         # df_new_test = pd.read_csv("../datasets/2A/"+i+"/dev.csv", nrows=52)
18         df_new_test["image_names"] = classes_present[i]
19         df_new_test = df_new_test.rename(columns={"image_names": "class"})
20         df_test = df_test.append(df_new_test)
21
22     df.to_csv("../datasets/2A/consolidated_train.csv", index=False)
23     df_test.to_csv("../datasets/2A/consolidated_dev.csv", index=False)
24     # df.to_csv("../datasets/2A/consolidated_train_small.csv", index=False)
25     # df_test.to_csv("../datasets/2A/consolidated_dev_small.csv", index=False)
26
27 def get_consolidated_data2B(classes_present):
28     df = pd.DataFrame()
29     df_test = pd.DataFrame()
30
31     for i in classes_present:
32         files = os.listdir("../datasets/2B/"+i+"/train/")
33         for k,j in tqdm(enumerate(files)):
34             df_new = pd.read_csv("../datasets/2B/"+i+"/train/"+j, header=None, sep=...
                " ")
35             df_new["class"] = classes_present[i]
36             df_new = df_new.reset_index()
37             df_new["image"] = str(k)

```

```

38         df = df.append(df_new)
39
40     files = os.listdir("../datasets/2B/"+i+"/dev/")
41     for k,j in tqdm(enumerate(files)):
42         df_new_test = pd.read_csv("../datasets/2B/"+i+"/dev/"+j, header=None, ...
43             sep=" ")
44         df_new_test["class"] = classes_present[i]
45         df_new_test = df_new_test.reset_index()
46         df_new_test["image"] = str(k)
47         df_test = df.append(df_new_test)
48
49     df.to_csv("../datasets/2B/consolidated_train.csv", index=False)
50     df_test.to_csv("../datasets/2B/consolidated_dev.csv", index=False)
51
52 if __name__ == "__main__":
53     classes_present = {"coast":0, "highway":1, "mountain":2, "opencountry":3, "...
54         tallbuilding":4}
55     get_consolidated_data2B(classes_present)

```

3.2 Bayes Classification, GMM, Diagonal Covariance

The GMM diagonal covariance model code is as follows:

```

1  #!/usr/bin/env python
2  # coding: utf-8
3
4  #####
5  import numpy as np
6  import pandas as pd
7  import matplotlib.pyplot as plt
8
9
10 #####
11 import seaborn as sns
12
13
14 #####
15 plt.rcParams["font.size"] = 18
16 plt.rcParams["axes.grid"] = True
17 plt.rcParams["figure.figsize"] = 8,6
18 plt.rcParams['font.serif'] = "Cambria"
19 plt.rcParams['font.family'] = "serif"
20
21
22 #####
23 import statistics as sts
24 from sklearn.model_selection import train_test_split
25
26
27 #####
28 from sklearn.cluster import KMeans
29
30
31 #####
32 coast_train = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/coast/train.csv"...
33 )
34 mountain_train = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/mountain/...
35     train.csv")
36 tallbuilding_train = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/...
37     tallbuilding/train.csv")
38 highway_train = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/highway/train....
39     csv")
40 opencountry_train = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/...
41     opencountry/train.csv")
42
43 coast_train.drop(["image_names"],axis = 1,inplace=True)

```

```

39 mountain_train.drop(["image_names"],axis = 1,inplace=True)
40 tallbuilding_train.drop(["image_names"],axis = 1,inplace=True)
41 highway_train.drop(["image_names"],axis = 1,inplace=True)
42 opencountry_train.drop(["image_names"],axis = 1,inplace=True)
43
44
45 #####
46 coast_test = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/coast/dev.csv")
47 mountain_test = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/mountain/dev....
    csv")
48 tallbuilding_test = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/...
    tallbuilding/dev.csv")
49 highway_test = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/highway/dev.csv...
    ")
50 opencountry_test = pd.read_csv("/home/hp/Desktop/acads/PRML/assignment2/opencountry...
    /dev.csv")
51
52 coast_test.drop(["image_names"],axis = 1,inplace=True)
53 mountain_test.drop(["image_names"],axis = 1,inplace=True)
54 tallbuilding_test.drop(["image_names"],axis = 1,inplace=True)
55 highway_test.drop(["image_names"],axis = 1,inplace=True)
56 opencountry_test.drop(["image_names"],axis = 1,inplace=True)
57
58
59 #####
60 coast_train.head()
61
62
63 #####
64 class GMM():
65     def __init__(self, q):
66         self.q = q
67
68     def fit(self, X, tol=1e-3):
69         """
70         X: n*d
71         mu: q*d
72         C: q*d*d
73         gamma: n*q
74         """
75         self.n, self.d = X.shape
76         self.X = X
77         #self.covariance_type = covariance_type
78         self.initialization()
79         self.lglk_list = []
80         for i in tqdm(range(100)):
81             self.lglk_list.append(self.log_likelihood(self.X))
82             self.expectation()
83             self.maximization()
84             new_lk = self.log_likelihood(self.X)
85             if new_lk - self.lglk_list[-1] < tol:
86                 break
87
88
89     def initialization(self):
90         kmeans = KMeans(n_clusters=self.q, random_state=0).fit(self.X)
91         labels = kmeans.labels_
92         unique, counts = np.unique(labels, return_counts=True)
93
94         self.subcomponents = unique.size
95         self.gamma = np.eye(self.subcomponents)[labels]
96         self.Nq = np.sum(self.gamma, axis=0)
97         self.weights = counts/self.n
98         self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
99         self.C = np.zeros((self.subcomponents, self.d, self.d))
100
101         for i in range(self.q):
102             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,:])

```

```

103
104         self.C[i] = np.diag(np.diag(self.C[i]))
105
106
107     def expectation(self):
108         self.gamma = np.zeros((self.n, self.q))
109         for i in range(self.q):
110             try:
111                 self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu.iloc[i], ...
112                     self.C[i])
113             except:
114                 self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu.iloc[i], ...
115                     self.C[i]+np.eye(self.C[i].shape[0])*1e-5)
116         self.gamma = self.gamma/np.sum(self.gamma, axis=1).reshape(-1,1)
117
118     def maximization(self):
119         # print(np.sum(self.weights))
120         self.Nq = np.sum(self.gamma, axis=0)
121         self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
122
123         for i in range(self.q):
124             self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self.mu.iloc[i,:])
125                 .T@(self.X-self.mu.iloc[i,:]))
126
127         self.C[i] = np.diag(np.diag(self.C[i]))
128
129         self.weights = self.Nq/self.n
130
131     def log_likelihood(self, X_test):
132         lk = 0
133         n, d = X_test.shape
134         for i in range(n):
135             val = 0
136             for j in range(self.q):
137                 #self.C[j] += np.eye(self.d)*1e-7
138                 val += self.weights[j]*mvn.pdf(X_test.iloc[i], self.mu.iloc[j], ...
139                     self.C[j])
140             lk += np.log(val)
141
142         return lk
143
144     def indiv_log_likelihood(self, X_test):
145         n, d = X_test.shape
146         lk = np.zeros((X_test.shape[0], 1))
147         for i in range(n):
148             val = 0
149             for j in range(self.q):
150                 val += self.weights[j]*mvn.pdf(X_test.iloc[i], self.mu.iloc[j], ...
151                     self.C[j])
152             lk[i] = np.log(val)
153
154         return lk
155
156     #####
157     gmm_list = defaultdict(list)
158
159     #####
160     Q = list(range(2,15))
161     for q in Q:
162         gmm = GMM(q)
163         gmm.fit(mountain_train)
164         gmm_list[4].append(gmm)
165
166     #####
167     import accuracy

```



```

167
168
169 #####
170 #predicting training data - selecting max likelihood value
171 X = mountain_train
172 ln = len(X)
173 Y_train = np.array([4]*ln)
174 acc_train = []
175 for i in tqdm(range(len(Q))):
176     gmm0 = gmm_list[0][i]
177     gmm1 = gmm_list[1][i]
178     gmm2 = gmm_list[2][i]
179     gmm3 = gmm_list[3][i]
180     gmm4 = gmm_list[4][i]
181
182     # Training
183     a = gmm0.indv_log_likelihood(X)
184     b = gmm1.indv_log_likelihood(X)
185     c = gmm2.indv_log_likelihood(X)
186     d = gmm3.indv_log_likelihood(X)
187     e = gmm4.indv_log_likelihood(X)
188
189     f = np.hstack((a, b, c, d, e))
190     pred = np.argmax(f, axis=1)
191
192     acc_calc = accuracy.Confusion_matrix(pred, Y_train)
193     acc_train.append(acc_calc.accuracy)
194 m_acc_train = acc_train
195
196 #####
197 train_acc = pd.DataFrame([c_acc_train, h_acc_train, t_acc_train, o_acc_train, ...
198     m_acc_train])
199
200 #####
201 get_ipython().run_line_magic('store', 'train_acc')
202
203
204 #####
205
206 from sklearn.metrics import confusion_matrix
207
208 #####
209
210 X = mountain_train
211 ln = len(X)
212 Y_train = np.array([4]*ln)
213 acc_train = []
214 acc_cv = []
215 for i in tqdm(range(len(Q))):
216     gmm0 = gmm_list[0][i]
217     gmm1 = gmm_list[1][i]
218     gmm2 = gmm_list[2][i]
219     gmm3 = gmm_list[3][i]
220     gmm4 = gmm_list[4][i]
221
222     # Training
223     a = gmm0.indv_log_likelihood(X)
224     b = gmm1.indv_log_likelihood(X)
225     c = gmm2.indv_log_likelihood(X)
226     d = gmm3.indv_log_likelihood(X)
227     e = gmm4.indv_log_likelihood(X)
228
229     f = np.hstack((a, b, c, d, e))
230     pred = np.argmax(f, axis=1)
231
232     acc_calc = accuracy.Confusion_matrix(pred, y_cv)
233     acc_cv.append(acc_calc.accuracy)
234 o_acc_cv = acc_cv

```

```

235
236
237 #####
238 cv_acc = pd.DataFrame([c_acc_cv,h_acc_cv,t_acc_cv,o_acc_cv,m_acc_cv])
239
240
241 #####
242 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"])
243 df.to_csv("acc2a.csv")
244
245
246 #####
247 plt.plot(Q,df.iloc[:,2],label="train")
248 plt.plot(Q,df.iloc[:,1],label = "test")
249 plt.title("Accuracy for training and test data 2A")
250 plt.xlabel("no. of components")
251 plt.ylabel("accuracy")
252 plt.legend()
253 plt.savefig("acc_2a.png")
254 plt.show()
255
256
257 #####
258 Q[5]
259
260
261 #####
262 X_test = mountain_test
263 ln = len(X_test)
264
265 Y_test = np.array([4]*ln)
266 X_cv,X_test,y_cv,y_test = train_test_split(X_test,Y_test, test_size=0.3, ...
    random_state=2)
267 ln = len(X_test)
268 X = X_test
269 i = 5
270 gmm0 = gmm_list[0][i]
271 gmm1 = gmm_list[1][i]
272 gmm2 = gmm_list[2][i]
273 gmm3 = gmm_list[3][i]
274 gmm4 = gmm_list[4][i]
275
276 # Training
277 a = gmm0.indv_log_likelihood(X)
278 b = gmm1.indv_log_likelihood(X)
279 c = gmm2.indv_log_likelihood(X)
280 d = gmm3.indv_log_likelihood(X)
281 e = gmm4.indv_log_likelihood(X)
282
283 f = np.hstack((a, b, c, d, e))
284 pred = np.argmax(f, axis=1)
285
286 acc_calc = accuracy.Confusion_matrix(pred,y_test)
287 acc_test.append(acc_calc.accuracy)
288 #o_acc_cv = acc_cv
289
290
291 #####
292 np.mean(np.array(acc_test))
293
294
295 #####
296 X_train = coast_train
297 X_train = X_train.append([highway_train,tallbuilding_train,opencountry_train,...
    mountain_train])
298
299

```

```

300 #####
301 #predicting training data - selecting max likelihood value
302 Y_train = [[0]*len(coast_train),[1]*len(highway_train),[2]*len(tallbuilding_train)...
            ,[3]*len(opencountry_train),[4]*len(mountain_train)]
303 X = X_train
304 #ln = len(X)
305 #Y_train = np.array([4]*ln)
306 acc_train = []
307 i = 5
308 gmm0 = gmm_list[0][i]
309 gmm1 = gmm_list[1][i]
310 gmm2 = gmm_list[2][i]
311 gmm3 = gmm_list[3][i]
312 gmm4 = gmm_list[4][i]
313
314 # Training
315 a = gmm0.indv_log_likelihood(X)
316 b = gmm1.indv_log_likelihood(X)
317 c = gmm2.indv_log_likelihood(X)
318 d = gmm3.indv_log_likelihood(X)
319 e = gmm4.indv_log_likelihood(X)
320
321 f = np.hstack((a, b, c, d, e))
322 pred = np.argmax(f, axis=1)
323
324 #####
325 flat_list = [item for sublist in Y_train for item in sublist]
326 pd.DataFrame(confusion_matrix(pred,flat_list)).to_csv("conf_train_2a.csv")
327
328 #####
329
330 #####
331 X_test = coast_test
332 X_test = X_test.append([highway_test,tallbuilding_test,opencountry_test,...
                        mountain_test])
333 Y_test = [[0]*len(coast_test),[1]*len(highway_test),[2]*len(tallbuilding_test),[3]*...
            len(opencountry_test),[4]*len(mountain_test)]
334
335 #####
336 #####
337 X = X_test
338 i = 5
339 gmm0 = gmm_list[0][i]
340 gmm1 = gmm_list[1][i]
341 gmm2 = gmm_list[2][i]
342 gmm3 = gmm_list[3][i]
343 gmm4 = gmm_list[4][i]
344
345 # Training
346 a = gmm0.indv_log_likelihood(X)
347 b = gmm1.indv_log_likelihood(X)
348 c = gmm2.indv_log_likelihood(X)
349 d = gmm3.indv_log_likelihood(X)
350 e = gmm4.indv_log_likelihood(X)
351
352 f = np.hstack((a, b, c, d, e))
353 pred = np.argmax(f, axis=1)
354
355 #####
356 #####
357 flat_list = [item for sublist in Y_test for item in sublist]
358 pd.DataFrame(confusion_matrix(pred,flat_list)).to_csv("conf_test_2a.csv")
359
360 #####
361 #####
362 conf_train = pd.read_csv("conf_train_2a.csv",index_col = 0)
363 conf_test = pd.read_csv("conf_test_2a.csv",index_col = 0)
364
365

```

```

366 #####
367 plt.figure()
368 sns.heatmap(conf_train, annot=True)
369 plt.title("Training Confusion Matrix")
370 plt.xlabel("Predicted Class")
371 plt.ylabel("Actual Class")
372 plt.savefig("conf_train2a.png")
373 plt.show()
374
375
376 #####
377 plt.figure()
378 sns.heatmap(conf_test, annot=True)
379 plt.title("Test Confusion Matrix")
380 plt.xlabel("Predicted Class")
381 plt.ylabel("Actual Class")
382 plt.savefig("conf_test2a.png")
383 plt.show()
384
385
386 #####

```

4 Dataset 2B

The GMM class module is as follows:

```

1  import numpy as np
2  from tqdm import tqdm
3  from sklearn.cluster import KMeans
4  from scipy.stats import multivariate_normal as mvn
5  import pandas as pd
6
7  class GMM():
8      def __init__(self, q):
9          self.q = q
10
11      def fit(self, X, covariance_type="diag", tol=1e-5):
12          """
13          X: n*d
14          mu: q*d
15          C: q*d*d
16          gamma: n*q
17          """
18          self.n, self.d = X.shape
19          self.X = X
20          self.covariance_type = covariance_type
21          self.initialization()
22          self.lglk_list = []
23          for i in tqdm(range(100)):
24              self.lglk_list.append(self.log_likelihood(self.X))
25              self.expectation()
26              self.maximization()
27              new_lk = self.log_likelihood(self.X)
28              diff = new_lk - self.lglk_list[-1]
29              if diff < tol:
30                  if diff < 0: print("Difference is less than 0")
31                  break
32
33
34      def initialization(self):
35          # kmeans = KMeans(n_clusters=self.q, random_state=0).fit(self.X)
36          kmeans = KMeans(n_clusters=self.q).fit(self.X)
37          labels = kmeans.labels_
38          unique, counts = np.unique(labels, return_counts=True)
39
40          self.subcomponents = unique.size

```

```

41     self.gamma = np.eye(self.subcomponents)[labels]
42     self.Nq = np.sum(self.gamma, axis=0)
43     self.weights = counts/self.n
44     self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
45     self.C = np.zeros((self.subcomponents, self.d, self.d))
46
47     for i in range(self.q):
48         self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self.mu[i,:])).T@self.X-self.mu[i,:])
49
50         if self.covariance_type == "diag":
51             self.C[i] = np.diag(self.C[i])
52
53
54     def expectation(self):
55         self.gamma = np.zeros((self.n, self.q))
56
57         for i in range(self.q):
58             try:
59                 self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self.C[i])
60             except:
61                 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)
62         self.gamma = self.gamma/np.sum(self.gamma, axis=1).reshape(-1,1)
63
64     def maximization(self):
65         # print(np.sum(self.weights))
66         self.Nq = np.sum(self.gamma, axis=0)
67         self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
68
69         for i in range(self.q):
70             self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self.mu[i,:])).T@self.X-self.mu[i,:])
71
72             if self.covariance_type == "diag":
73                 self.C[i] = np.diag(self.C[i])
74
75         self.weights = self.Nq/self.n
76
77     def log_likelihood(self, X_test):
78         lk = 0
79         n, d = X_test.shape
80         for i in range(n):
81             val = 0
82             for j in range(self.q):
83                 try:
84                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j])
85                 except:
86                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j]+np.eye(self.C[j].shape[0])*1e-7)
87             lk += np.log(val)
88
89         return lk
90
91     def indiv_log_likelihood(self, X_test):
92         n, d = X_test.shape
93         lk = np.zeros((X_test.shape[0], 1))
94         for i in range(n):
95             val = 0
96             for j in range(self.q):
97                 try:
98                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j])
99                 except:
100                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j]+np.eye(self.C[j].shape[0])*1e-7)
101             lk[i] = np.log(val)

```

```

102
103     return lk
104
105     def gaussian_val(self, X_test):
106         n, d = X_test.shape
107         val = np.zeros((n, self.q))
108
109         for i in range(self.q):
110             val[:,i] = self.weights[i]*mvn.pdf(X_test, self.mu[i], self.C[i])
111
112         return np.sum(val, axis=1)
113
114     class GMM_v1():
115         def __init__(self, q):
116             self.q = q
117
118         def fit(self, X, epochs=100, covariance_type="diag", tol=1e-5):
119             """
120             X: n*d
121             mu: q*d
122             C: q*d*d
123             gamma: n*q
124             """
125             self.n, self.d = X.shape
126             self.X = X
127             self.epochs = epochs
128             self.covariance_type = covariance_type
129             self.initialization()
130             self.lglk_list = []
131             for i in tqdm(range(self.epochs)):
132                 self.lglk_list.append(self.log_likelihood(self.X))
133                 self.expectation()
134                 self.maximization()
135                 new_lk = self.log_likelihood(self.X)
136                 diff = new_lk - self.lglk_list[-1]
137                 if diff < tol:
138                     if diff < 0:
139                         print("Difference is less than 0")
140                         break
141
142             def initialization(self):
143                 # kmeans = KMeans(n_clusters=self.q, random_state=0).fit(self.X)
144                 kmeans = KMeans(n_clusters=self.q).fit(self.X)
145                 labels = kmeans.labels_
146                 unique, counts = np.unique(labels, return_counts=True)
147
148                 self.subcomponents = unique.size
149                 self.gamma = np.eye(self.subcomponents)[labels]
150                 self.Nq = np.sum(self.gamma, axis=0)
151                 self.weights = counts/self.n
152                 self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
153                 self.C = np.zeros((self.subcomponents, self.d, self.d))
154
155                 for i in range(self.q):
156                     self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self.mu[i,:])).T@self.X-self.mu[i,:])
157
158                     if self.covariance_type == "diag":
159                         self.C[i] = np.diag(np.diag(self.C[i]))
160
161             def expectation(self):
162                 self.gamma = np.zeros((self.n, self.q))
163
164                 for i in range(self.q):
165                     try:
166                         self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self.C[i])
167                     except:

```

```

169         self.gamma[:,i] = self.weights[i]*mvn.pdf(self.X, self.mu[i], self....
170             C[i]+np.eye(self.C[i].shape[0])*1e-3)
171         self.gamma = self.gamma/np.sum(self.gamma, axis=1).reshape(-1,1)
172
173     def maximization(self):
174         # print(np.sum(self.weights))
175         self.Nq = np.sum(self.gamma, axis=0)
176         self.mu = (self.gamma.T @ self.X)/self.Nq.reshape(-1,1)
177
178         for i in range(self.q):
179             self.C[i] = (1/self.Nq[i])*(self.gamma[:,i].reshape(-1,1)*(self.X-self....
180                 mu[i,:])).T@(self.X-self.mu[i,:])
181
182         if self.covariance_type == "diag":
183             self.C[i] = np.diag(np.diag(self.C[i]))
184
185         self.weights = self.Nq/self.n
186
187     def log_likelihood(self, X_test):
188         lk = 0
189         n, d = X_test.shape
190         for i in range(n):
191             val = 0
192             for j in range(self.q):
193                 try:
194                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self....
195                         C[j])
196                 except:
197                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self.C[j]...
198                         ]+np.eye(self.C[j].shape[0])*1e-3)
199             lk += np.log(val)
200
201         return lk
202
203     def indv_log_likelihood(self, X_test):
204         n, d = X_test.shape
205         lk = np.zeros((X_test.shape[0], 1))
206         for i in range(n):
207             val = 0
208             for j in range(self.q):
209                 try:
210                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self....
211                         C[j])
212                 except:
213                     val += self.weights[j]*mvn.pdf(X_test[i], self.mu[j], self....
214                         C[j]+np.eye(self.C[j].shape[0])*1e-3)
215             lk[i] = np.log(val)
216
217         return lk
218
219     def gaussian_val(self, X_test):
220         n, d = X_test.shape
221         val = np.zeros((n, self.q))
222
223         for i in range(self.q):
224             val[:,i] = self.weights[i]*mvn.pdf(X_test, self.mu[i], self.C[i])
225
226         return np.sum(val, axis=1)
227
228     def probab(self, df):
229         df = pd.DataFrame(df)
230         grouped_df = df.groupby(by=["class", "image"])
231         for key, item in grouped_df:
232             selected_df = grouped_df.get_group(key)
233             X_select = selected_df.drop(["index", "image", "class"], axis=1)....
234                 to_numpy()
235             val = self.gaussian_val(X_select)
236             print(val.shape)

```

The code used is as follows:

```
1  #!/usr/bin/env python
2  # coding: utf-8
3  #####
4  import os
5  import numpy as np
6  import pandas as pd
7  from tqdm import tqdm
8  from gmm import GMM_v1
9
10 get_ipython().run_line_magic('load_ext', 'autoreload')
11 get_ipython().run_line_magic('autoreload', '2')
12
13 #####
14 df = pd.read_csv("../datasets/2B/consolidated_train.csv")
15 X = df.drop(["class", "image", "index"], axis=1).to_numpy()
16 print(X.shape)
17 df.head()
18
19 #####
20 classes = np.unique(df["class"])
21 gmm_list = []
22
23 for i in classes:
24     gmm = GMM_v1(q=14)
25     df_selected = df[df["class"]==i]
26
27     X_selected = df_selected.drop(["class", "image", "index"], axis=1).to_numpy()
28     gmm.fit(X_selected, epochs=20)
29     gmm_list.append(gmm)
30
31 #####
32 gmm.probab(df_selected)
33
34 #####
35 gmm.gamma
36
37 #####
```