
Programming Assignment II

CS5691: PATTERN RECOGNITION AND MACHINE LEARNING

TEAM 17 CCS SECTION SPRING 2021

SEPTEMBER 23, 2022

AANAND KRISHNAN
BE17B001

MANORANJAN J
NA17B112

RENEETH KRISHNA MG
BS17B025

Contents

I	Pattern classification on linearly separable data	2
I.1	Python Code	2
II	Pattern classification on non-linearly separable data	9
II.1	K nearest Neighbours Method and Bayes classifier with KNN for density estimation	9
II.1.1	Python Code	9
II.2	Bayes Classifier with GMM	15
II.2.1	Python Code	15
III	Static Pattern Classification on Real World Dataset 2A	24
III.1	Python Code	24
IV	Static Pattern Classification on Real World Dataset 2B	31
IV.1	Python Code	31

I Pattern classification on linearly separable data

I.1 Python Code

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 from sklearn.metrics import confusion_matrix
5 from mlxtend.plotting import plot_confusion_matrix
6 from sklearn.metrics import accuracy_score
7
8 #####
9 # train data
10 f = open('datasets/Dataset_1a/train.csv', 'r')
11
12 length = 0
13 for line in f:
14
15     d = [float(i) for i in line.split(',')]
16
17     if length == 0:
18         data = np.array(d)
19         data = data[np.newaxis,:]
20     else:
21         d = np.array(d)
22         d = d[np.newaxis,:]
23         data = np.append(data,d,axis=0)
24     length = length+1
25
26 f.close()
27
28 train_data = pd.DataFrame(data)
29
30 # shuffle dataset
31 # train_data = train_data.sample(frac=1)
32
33 # get train data
34 train_data = np.array(train_data)
35
36 # test data
37 f = open('datasets/Dataset_1a/dev.csv', 'r')
38
39 length = 0
40 for line in f:
41
42     d = [float(i) for i in line.split(',')]
43
44     if length == 0:
45         data = np.array(d)
46         data = data[np.newaxis,:]
```

```

47     else:
48         d = np.array(d)
49         d = d[np.newaxis,:]
50         data = np.append(data,d,axis=0)
51         length = length+1
52
53 f.close()
54
55 test_data = pd.DataFrame(data)
56
57 # shuffle dataset
58 # test_data = test_data.sample(frac=1)
59
60 # get test data
61 test_data = np.array(test_data)
62 #####
63
64 # Split training data
65 # length of data for fit
66 train_len = int(np.shape(train_data)[0])
67 val_len = int(np.shape(test_data)[0]*0.5)
68 test_len = int(np.shape(test_data)[0]*0.5)
69
70 X_train = train_data[:,0:2]
71 X_val = test_data[0:val_len,0:2]
72 X_test = test_data[val_len:val_len+test_len,0:2]
73
74 y_train = train_data[:,2]
75 y_val = test_data[0:val_len,2]
76 y_test = test_data[val_len:val_len+test_len,2]
77
78 #####
79 # Build KNN classifier
80 y_pred = []
81 def KNN_classifier(K, x):
82     """
83     Parameters
84     -----
85     K : value of nearest neighbours
86     x : feature vector
87
88     Returns
89     -----
90     None.
91     """
92     # find distance between feature vector and training data
93     dist = np.linalg.norm(x-X_train,axis=1)
94
95     # get the top index for the minimum distance
96     min_dist_index = np.argsort(dist)

```

```

97     topk = min_dist_index[0:K]
98
99     # get class of corresponding class
100    K_class = y_train[topk]
101
102    # get count of each class
103    unique_class, counts = np.unique(K_class, return_counts=1)
104
105    # get the index of max counts
106    max_count = np.argmax(counts)
107
108    # choose that as the class
109    y_pred.append(unique_class[max_count])
110
111    K = 15
112    # test on training data
113    for i in range(0, train_len):
114        KNN_classifier(K, X_train[i])
115
116    y_pred = np.array(y_pred)
117
118    train_accuracy = accuracy_score(y_train, y_pred)*100
119    print("training accuracy: " , train_accuracy)
120
121    # plot confusion matrix for training data
122    c_matrix = confusion_matrix(y_train, y_pred)
123    fig, ax = plot_confusion_matrix(conf_mat=c_matrix, figsize=(7,7), cmap=plt.cm.RdYlBu_r)
124    ax.set(title = "Confusion Matrix")
125    plt.show()
126
127    y_pred = []
128    # test on validation data
129    for i in range(0, val_len):
130        KNN_classifier(K, X_val[i])
131
132    y_pred = np.array(y_pred)
133
134    val_accuracy = accuracy_score(y_val, y_pred)*100
135    print("validation accuracy: " , val_accuracy)
136
137    y_pred = []
138    # test on test data
139    for i in range(0, test_len):
140        KNN_classifier(K, X_test[i])
141
142    y_pred = np.array(y_pred)
143
144    test_accuracy = accuracy_score(y_test, y_pred)*100
145    print("validation accuracy: " , test_accuracy)
146

```

```

147 # plot confusion matrix for test data
148 c_matrix = confusion_matrix(y_test, y_pred)
149 fig, ax = plot_confusion_matrix(conf_mat=c_matrix,figsize=(7,7),cmap=plt.cm.RdYlBu_r)
150 ax.set(title = "Confusion Matrix")
151 plt.show()
152
153 # Decision Region Plots #####
154 # define bounds of the domain
155 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
156 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
157
158 # define the x and y scale
159 x1_grid = np.arange(min1, max1, 0.1)
160 x2_grid = np.arange(min2, max2, 0.1)
161
162 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
163
164 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
165 c1, c2 = x1_grid.reshape((len(c1), 1)), x2_grid.reshape((len(c2), 1))
166
167 x = np.hstack((c1,c2))
168
169 y_pred = []
170 for i in range(0, np.shape(x)[0]):
171     KNN_classifier(K, x[i,:])
172
173 y_pred = np.array(y_pred)
174
175 x3_grid = y_pred.reshape(x1_grid.shape)
176
177 fig = plt.figure()
178 ax = fig.add_subplot(111)
179 ax.contourf(x1_grid, x2_grid, x3_grid, cmap='Paired')
180 ax.scatter(X_train[:,0],X_train[:,1],marker='x')
181 ax.set_xlabel('x1',fontsize=20)
182 ax.set_ylabel('x2',fontsize=20)
183 ax.set_title('KNN model with K = 15', fontsize=20)
184
185
186 #####
187
188 # Naive Bayes classifier
189 #  $P(x/y=y_i) = N(x/\mu_i, \sigma_i)$ 
190 unique_class, class_index, counts = np.unique(y_train, return_inverse=1,return_counts=True)
191
192 # Compute mean and variance for each class
193 mu = np.zeros((np.shape(unique_class)[0],2))
194 variance = np.zeros((np.shape(unique_class)[0],2))
195 for i in range(0,np.size(unique_class)):
196     index = np.where(class_index==i)

```

```

197     mu[i,:] = np.mean(X_train[index,:],axis=1)
198     variance[i,:] = np.var(X_train[index,:],axis=1)
199
200     # Gaussian function
201     N = lambda mu,C,x : ((1/(((2*np.pi)**(np.shape(unique_class)[0]/2))
202     *np.linalg.det(C)**(1/2))))*np.exp(-0.5*(((x-mu).T@np.linalg.inv(C)@(x-mu))))
203     #####
204     # Comment all other case when testing 1 case
205
206     # case 1: when covariance matrix is sigma**2[I] #####
207     var_avg = (np.mean(np.sum(variance,axis=1)/2.0))
208
209     covar = np.eye(np.shape(X_train)[1])*var_avg
210     #####
211     # case 2: when covariance matrix is same but has different diagonal elements
212     covar = np.zeros((np.shape(unique_class)[0],
213     np.shape(X_train)[1],np.shape(X_train)[1]))
214     for i in range(0,np.shape(unique_class)[0]):
215         covar[i,:,:] = np.diag(variance[i,:])
216     covar = np.mean(covar,axis=0)
217     #####
218     # case 3: when covariance matrix is different but has diagonal elements
219     covar = np.zeros((np.shape(unique_class)[0],
220     np.shape(X_train)[1],np.shape(X_train)[1]))
221     for i in range(0,np.shape(unique_class)[0]):
222         covar[i,:,:] = np.diag(variance[i,:])
223
224
225     prior = counts/train_len
226
227     # test on train data
228     y_pred = np.zeros((train_len,))
229     for i in range(train_len):
230         decision = []
231         for j in range(0,np.shape(unique_class)[0]):
232             decision.append(N(mu[j],covar[j,:,:],X_train[i,:])*prior[j])
233         y_pred[i] = np.argmax(decision)
234
235     train_accuracy = accuracy_score(y_train,y_pred)*100
236     print("training accuracy: " , train_accuracy)
237
238     # plot confusion matrix for training data
239     c_matrix = confusion_matrix(y_train, y_pred)
240     fig, ax = plot_confusion_matrix(conf_mat=c_matrix,figsize=(7,7),cmap=plt.cm.RdYlBu_r)
241     ax.set(title = "Confusion Matrix")
242     plt.show()
243
244     # test on validation data
245     y_pred = np.zeros((val_len,))
246     for i in range(val_len):

```

```

247     decision = []
248     for j in range(0,np.shape(unique_class)[0]):
249         decision.append(N(mu[j],covar[j,:,:],X_val[i,:])*prior[j])
250     y_pred[i] = np.argmax(decision)
251
252 val_accuracy = accuracy_score(y_val,y_pred)*100
253 print("validation accuracy: " , val_accuracy)
254
255 # test on test data
256 y_pred = np.zeros((test_len,))
257 for i in range(test_len):
258     decision = []
259     for j in range(0,np.shape(unique_class)[0]):
260         decision.append(N(mu[j],covar[j,:,:],X_test[i,:])*prior[j])
261     y_pred[i] = np.argmax(decision)
262
263 y_test = np.array(y_test)
264 test_accuracy = accuracy_score(y_test,y_pred)*100
265 print("test accuracy: " , test_accuracy)
266
267 # plot confusion matrix for test data
268 c_matrix = confusion_matrix(y_test, y_pred)
269 fig, ax = plot_confusion_matrix(conf_mat=c_matrix,figsize=(7,7),cmap=plt.cm.RdYlBu_r)
270 ax.set(title = "Confusion Matrix")
271 plt.show()
272
273 # Decision Region Plots #####
274 # define bounds of the domain
275 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
276 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
277
278 # define the x and y scale
279 x1_grid = np.arange(min1, max1, 0.1)
280 x2_grid = np.arange(min2, max2, 0.1)
281
282 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
283
284 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
285 c1, c2 = x1_grid.reshape((len(c1), 1)), x2_grid.reshape((len(c2), 1))
286
287 x = np.hstack((c1,c2))
288
289 y_pred = np.zeros((np.shape(x)[0],))
290 for i in range(np.shape(x)[0]):
291     decision = []
292     for j in range(0,np.shape(unique_class)[0]):
293         decision.append(N(mu[j],covar[j,:,:],x[i,:])*prior[j])
294     y_pred[i] = np.argmax(decision)
295
296 x3_grid = y_pred.reshape(x1_grid.shape)

```



```

297
298 fig = plt.figure()
299 ax = fig.add_subplot(111)
300 ax.contourf(x1_grid, x2_grid, x3_grid, cmap='Paired')
301 ax.scatter(X_train[:,0],X_train[:,1],marker='x')
302 ax.set_xlabel('x1',fontsize=20)
303 ax.set_ylabel('x2',fontsize=20)
304 ax.set_title('Naiyve Bayes Classifier', fontsize=20)
305
306 # plot levl of curves of the gaussian functions
307 for i in range(0,np.shape(unique_class)[0]):
308     x3_grid = []
309     for j in range(np.shape(x)[0]):
310         x3_grid.append(N(mu[i],covar[i,:,:],x[j,:]))
311 x3_grid = np.array(x3_grid)
312 x3_grid = x3_grid.reshape(x1_grid.shape)
313 contours = ax.contour(x1_grid, x2_grid, x3_grid, cmap='tab20b')
314 ax.clabel(contours, inline=1, fontsize=5)

```

II Pattern classification on non-linearly separable data

II.1 K nearest Neighbours Method and Bayes classifier with KNN for density estimation

II.1.1 Python Code

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 from sklearn.metrics import confusion_matrix
5 from mlxtend.plotting import plot_confusion_matrix
6 from sklearn.metrics import accuracy_score
7 from sklearn.cluster import KMeans
8 from scipy.stats import multivariate_normal
9 #####
10 # train data
11 f = open('datasets/Dataset_1b/train.csv', 'r')
12
13 length = 0
14 for line in f:
15
16     d = [float(i) for i in line.split(',')]
17
18     if length == 0:
19         data = np.array(d)
20         data = data[np.newaxis,:]
21     else:
22         d = np.array(d)
23         d = d[np.newaxis,:]
24         data = np.append(data,d,axis=0)
25     length = length+1
26
27 f.close()
28
29 train_data = pd.DataFrame(data)
30
31 # shuffle dataset
32 # train_data = train_data.sample(frac=1)
33
34 # get train data
35 train_data = np.array(train_data)
36
37 # test data
38 f = open('datasets/Dataset_1b/dev.csv', 'r')
39
40 length = 0
41 for line in f:
42
43     d = [float(i) for i in line.split(',')]
44
```

```

45     if length == 0:
46         data = np.array(d)
47         data = data[np.newaxis,:]
48     else:
49         d = np.array(d)
50         d = d[np.newaxis,:]
51         data = np.append(data,d,axis=0)
52     length = length+1
53
54 f.close()
55
56 test_data = pd.DataFrame(data)
57
58 # shuffle dataset
59 test_data = test_data.sample(frac=1)
60
61 # get test data
62 test_data = np.array(test_data)
63 #####
64
65 # Split training data
66 # length of data for fit
67 train_len = int(np.shape(train_data)[0])
68 val_len = int(np.shape(test_data)[0]*0.5)
69 test_len = int(np.shape(test_data)[0]*0.5)
70
71 X_train = train_data[:,0:2]
72 X_val = test_data[0:val_len,0:2]
73 X_test = test_data[val_len:val_len+test_len,0:2]
74
75 y_train = train_data[:,2]
76 y_val = test_data[0:val_len,2]
77 y_test = test_data[val_len:val_len+test_len,2]
78
79 KNN classifier
80
81 # Build KNN classifier
82 y_pred = []
83 def KNN_classifier(K, x):
84     """
85     Parameters
86     -----
87     K : value of nearest neighbours
88     x : feature vector
89
90     Returns
91     -----
92     None.
93     """
94     # find distance between feature vector and training data

```

```

95     dist = np.linalg.norm(x-X_train,axis=1)
96
97     # get the top index for the minimum distance
98     min_dist_index = np.argsort(dist)
99     topk = min_dist_index[0:K]
100
101     # get class of corresponding class
102     K_class = y_train[topk]
103
104     # get count of each class
105     unique_class, counts = np.unique(K_class, return_counts=1)
106
107     # get the index of max counts
108     max_count = np.argmax(counts)
109
110     # choose that as the class
111     y_pred.append(unique_class[max_count])
112
113 K = 15
114 # test on training data
115 for i in range(0, train_len):
116     KNN_classifier(K, X_train[i])
117
118 y_pred = np.array(y_pred)
119
120 train_accuracy = accuracy_score(y_train,y_pred)*100
121 print("training accuracy: " , train_accuracy)
122
123 # plot confusion matrix for training data
124 c_matrix = confusion_matrix(y_train, y_pred)
125 fig, ax = plot_confusion_matrix(conf_mat=c_matrix,figsize=(7,7),cmap=plt.cm.RdYlBu_r)
126 ax.set(title = "Confusion Matrix")
127 plt.show()
128
129 y_pred = []
130 # test on validation data
131 for i in range(0, val_len):
132     KNN_classifier(K, X_val[i])
133
134 y_pred = np.array(y_pred)
135
136 val_accuracy = accuracy_score(y_val,y_pred)*100
137 print("validation accuracy: " , val_accuracy)
138
139 y_pred = []
140 # test on test data
141 for i in range(0, test_len):
142     KNN_classifier(K, X_test[i])
143
144 y_pred = np.array(y_pred)

```

```

145
146 test_accuracy = accuracy_score(y_test,y_pred)*100
147 print("test accuracy: " , test_accuracy)
148
149 # plot confusion matrix for test data
150 c_matrix = confusion_matrix(y_test, y_pred)
151 fig, ax = plot_confusion_matrix(conf_mat=c_matrix,figsize=(7,7),cmap=plt.cm.RdYlBu_r)
152 ax.set(title = "Confusion Matrix")
153 plt.show()
154
155 # Decision Region Plots #####
156 # define bounds of the domain
157 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
158 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
159
160 # define the x and y scale
161 x1_grid = np.arange(min1, max1, 0.1)
162 x2_grid = np.arange(min2, max2, 0.1)
163
164 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
165
166 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
167 c1, c2 = x1_grid.reshape((len(c1), 1)), x2_grid.reshape((len(c2), 1))
168
169 x = np.hstack((c1,c2))
170
171 y_pred = []
172 for i in range(0, np.shape(x)[0]):
173     KNN_classifier(K, x[i,:])
174
175 y_pred = np.array(y_pred)
176
177 x3_grid = y_pred.reshape(x1_grid.shape)
178
179 fig = plt.figure()
180 ax = fig.add_subplot(111)
181 ax.contourf(x1_grid, x2_grid, x3_grid, cmap='Paired')
182 ax.scatter(X_train[:,0],X_train[:,1],marker='x')
183 ax.scatter(X_test[:,0],X_test[:,1],marker='x')
184 ax.set_xlabel('x1',fontsize=20)
185 ax.set_ylabel('x2',fontsize=20)
186 ax.set_title('KNN model with K = 15', fontsize=20)
187
188 #####
189 # Bayes classifier with KNN for density estimation
190
191 # Build KNN for density estimation
192 y_pred = []
193 def KNN(K, x, X):
194     """

```

```

195     Parameters
196     -----
197     K : value of nearest neighbours
198     x : feature vector
199     X : training data related to particular class
200     Returns
201     -----
202     None.
203     """
204     # find distance between feature vector and training data
205     dist = np.linalg.norm(x-X,axis=1)
206
207     # get the top k index for the minimum distance
208     min_dist_index = np.argsort(dist)
209     topk = min_dist_index[0:K]
210
211     # radius is distance of kth neearest neighbour
212     R = dist[topk[-1]]
213
214     return R
215
216     # split into different classes
217     unique_class,counts = np.unique(y_train,return_counts=1)
218     total_class = len(unique_class)
219
220     class_data = []
221
222     for i in range(0,total_class):
223         class_data.append(X_train[y_train==i])
224
225     K=20
226     # bayes classifier -> this is just the min value of R for all the classes
227     # upon simplification of the actual bayes theorem
228     # test on train data
229     y_pred = np.zeros((train_len,))
230     for i in range(train_len):
231         decision = []
232         for j in range(0,total_class):
233             decision.append(KNN(K,X_train[i],class_data[j]))
234         y_pred[i] = np.argmin(decision)
235
236     train_accuracy = accuracy_score(y_train,y_pred)*100
237     print("training accuracy: " , train_accuracy)
238
239     # plot confusion matrix for training data
240     c_matrix = confusion_matrix(y_train, y_pred)
241     fig, ax = plot_confusion_matrix(conf_mat=c_matrix,figsize=(7,7),cmap=plt.cm.RdYlBu_r)
242     ax.set(title = "Confusion Matrix")
243     plt.show()
244

```

```

245 # test on validation data
246 y_pred = np.zeros((val_len,))
247 for i in range(val_len):
248     decision = []
249     for j in range(0,total_class):
250         decision.append(KNN(K,X_val[i],class_data[j]))
251     y_pred[i] = np.argmin(decision)
252
253 val_accuracy = accuracy_score(y_val,y_pred)*100
254 print("validation accuracy: " , val_accuracy)
255
256 # test on test data
257 y_pred = np.zeros((test_len,))
258 for i in range(test_len):
259     decision = []
260     for j in range(0,total_class):
261         decision.append(KNN(K,X_test[i],class_data[j]))
262     y_pred[i] = np.argmin(decision)
263
264 y_test = np.array(y_test)
265 test_accuracy = accuracy_score(y_test,y_pred)*100
266 print("test accuracy: " , test_accuracy)
267
268 # plot confusion matrix for test data
269 c_matrix = confusion_matrix(y_test, y_pred)
270 fig, ax = plot_confusion_matrix(conf_mat=c_matrix,figsize=(7,7),cmap=plt.cm.RdYlBu_r)
271 ax.set(title = "Confusion Matrix")
272 plt.show()
273
274
275 # Decision Region Plots #####
276 # define bounds of the domain
277 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
278 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
279
280 # define the x and y scale
281 x1_grid = np.arange(min1, max1, 0.1)
282 x2_grid = np.arange(min2, max2, 0.1)
283
284 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
285
286 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
287 c1, c2 = x1_grid.reshape((len(c1), 1)), x2_grid.reshape((len(c2), 1))
288
289 x = np.hstack((c1,c2))
290
291 y_pred = np.zeros((np.shape(x)[0],))
292 for i in range(np.shape(x)[0]):
293     decision = []
294     for j in range(0,total_class):

```

```

295         decision.append(KNN(K,x[i],class_data[j]))
296     y_pred[i] = np.argmin(decision)
297
298 x3_grid = y_pred.reshape(x1_grid.shape)
299
300 fig = plt.figure()
301 ax = fig.add_subplot(111)
302 ax.contourf(x1_grid, x2_grid, x3_grid, cmap='Paired')
303 ax.scatter(X_train[:,0],X_train[:,1],marker='x')
304 ax.set_xlabel('x1',fontsize=20)
305 ax.set_ylabel('x2',fontsize=20)
306 ax.set_title('Bayes Classifier with KNN density estimation', fontsize=20)

```

II.2 Bayes Classifier with GMM

II.2.1 Python Code

```

1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[1]:
5
6
7  import pandas as pd
8  import numpy as np
9  import matplotlib.pyplot as plt
10 import seaborn as sns #A statistical plotting library
11 from sklearn.cluster import KMeans
12 from kneed import KneeLocator #A function that helps in optimization of
13                                #number of clusters from an error curve
14 from scipy.stats import multivariate_normal as mvn
15 from mlxtend.plotting import plot_confusion_matrix
16 from sklearn.metrics import confusion_matrix
17
18
19 # In[2]:
20
21
22 header_names = ['x1', 'x2' , 'Class']
23 D = pd.read_csv('datasets/Dataset_1b/train.csv', header = None, names = header_names)
24 D.head()
25
26
27 # In[3]:
28
29
30 L_df = D.loc[:,['x1','x2']]
31 Unlab_Data = L_df.to_numpy()
32
33 lab_df = D.loc[:, 'Class']

```



```

34 labels = lab_df.to_numpy()
35
36 #Training Dataset for Class 0
37 L0 = (D['Class'] == 0.0)
38 L0_df = D.loc[L0 , ['x1', 'x2']]
39 Class0 = L0_df.to_numpy()
40
41 #Training Dataset for Class 1
42 L1 = (D['Class'] == 1.0)
43 L1_df = D.loc[L1 , ['x1', 'x2']]
44 Class1 = L1_df.to_numpy()
45
46 #Training Dataset for Class 2
47 L2 = (D['Class'] == 2.0)
48 L2_df = D.loc[L2, ['x1', 'x2']]
49 Class2 = L2_df.to_numpy()
50 labels.shape
51
52
53 # In[4]:
54
55
56 #KMeans implementation for initialization and optimization of the number of
57 #clusters.
58 #Number of clusters for each class equals the number of gaussian componenets
59 #to be fitted for that class.
60 def K_Clustering(Class,M):
61     #Dictionary of the arguments for scikit.KMeans
62     KMeans_args = {
63         "init" : "random",
64         "n_init" : 10,
65         "max_iter" : 300,
66         "random_state" : 0,
67     }
68     #Estimation of the optimum number of clusters using elbow method
69     std_error = []
70     for cluster in range(1,11):
71         kmeans = KMeans(n_clusters = cluster , **KMeans_args)
72         kmeans.fit(Class)
73         std_error.append(kmeans.inertia_)
74     if M==0:
75         #detecting the elbow point of the curve of 's_err vs K' using kneed, which
76         #gives the optimum number of clusters
77         curve = KneeLocator(range(1,11), std_error, curve="convex",
78             direction = "decreasing")
79         K_opt = curve.elbow
80     else:
81         #Using Manually entered value for K_opt
82         K_opt = M
83     #clustering the class in to K_opt clusters

```

```

84     kmeans = KMeans(n_clusters = K_opt , **KMeans_args)
85     kmeans.fit(Class)
86     labels = kmeans.labels_
87     centers = kmeans.cluster_centers_
88     return K_opt,labels,centers
89
90
91 # In[5]:
92
93 #initialization of the parameters using K-Clusters
94
95 def Parameters_old(Class,M):
96     #Will return a mean(mu)-(K,d) array;
97     N,d = np.shape(Class)
98     K,lab,mu = K_Clustering(Class,M)
99     #gamma contains initial responsibilty values for an example w.r.t
100    #each clusters as columns
101    gamma = np.array([ [0]*K for i in range(N)])
102    for example in range(N):
103        for cluster in range(K):
104            if lab[example] == cluster:
105                gamma[example][cluster]= 1
106    return K,mu,gamma
107
108
109 # In[6]:
110
111
112
113 #Defining the Gaussian Mixture Model as a class
114
115 class Gaussian_Mixture_Model:
116     #Class - Examples of the class to which the Gaussian Componenets
117     #need to be fitted
118     #Class - N x d matrix, where N is the number of examples and
119     #d is the number of features for each example
120     #K - Number of Gaussian Components that needs to be fitted
121
122     def __init__(self,Class,K,MU,GAMMA,f):
123         self.Class = Class
124         self.K = K      #Attribute for Number of clusters
125         self.GAMMA = GAMMA      #Attribute for NxK array of posterior
126                                #prob. / responsibilty term.
127         self.MU = MU      #Attribute for the mean values. An Kxd array.
128         self.SIGMA = None  #Attribute for (K,d,d) array of covariances
129         self.W = None      #Attribute for prior probabilty,
130                            #an array of length K
131         #self.max_iter = max_iter      #Attribute for the number of iterations
132         self.N = len(self.Class)      #Attribute for number of examples available
133         self.d = len(self.Class[0])  #Attribute for the number of features

```

```

134                                     #in each example
135     self.f = f                       #Attribute that acts as switch between
136                                     #diagonal and full covariance matrices
137     self.mean_shift = np.reshape(self.Class, (self.N, 1, self.d) ) -
138                             np.reshape(self.MU, (1, self.K, self.d) )
139
140     def Prior_Probability(self):
141         #A function to estimate the (K,) array of prior prob.
142         self.W = np.einsum("ij -> j",self.GAMMA) / self.N
143
144     def Mean(self):
145         # A function to calculate mean
146         self.MU = ((self.GAMMA).T) @ (self.Class) / np.reshape((self.W*self.N),
147             (self.K, 1))
148
149     def Covariance_Matrix_Array(self):
150         # A function to calculate covariances of the features of the examples
151
152         Nk = np.einsum("ij -> j",self.GAMMA)
153         self.mean_shift = np.reshape(self.Class, (self.N, 1, self.d) ) -
154             np.reshape(self.MU, (1, self.K, self.d) )
155
156         sigma = np.einsum("nki,nkj->kij", np.einsum("nk,nki->nki", self.GAMMA,
157             self.mean_shift), self.mean_shift) / np.reshape(Nk,
158             (self.K, 1, 1))
159
160         if self.f==1: #Case where we use full diagonal covariance matrix
161             self.SIGMA = sigma
162
163         if self.f==0: #Case where we use a diagonal covariance matrix
164             I = np.identity(self.d,dtype=int) #An identity matrix of the size
165             #equal to number of feature
166
167             self.SIGMA = np.einsum("kij,ij -> kij",sigma,I)
168
169
170     def Gaussian_Prob(self):
171         #This function accounts for our assumption that the conditional
172         #distribution of an example is a Gaussian.
173
174         self.Covariance_Matrix_Array()           #SIGMA gets updated to the
175                                                    #full covariance matrix
176         SIGMA_inv = np.linalg.inv(self.SIGMA)     #Inverse of the covariance matrix
177
178         #Normalisation term of the Gaussian dist.
179         norm = np.sqrt(((2*np.pi)**self.d)*np.linalg.det(self.SIGMA))
180
181
182         #Exponential term of the Gaussian
183         expo = np.exp(-0.5*(np.einsum("nkj,nkj->nk", np.einsum("nki,kij->nkj",

```

```

184         self.mean_shift, SIGMA_inv),self.mean_shift)))
185
186     #Prob_mat is an (NxK)-array that contains Gaussian Prob. of the
187     #various examples to belong to respective clusters
188     Prob_mat = expo / norm
189     return Prob_mat
190
191     def Expectation_Step(self):
192         #In this step we update the values of the responsibilty term
193
194         N = self.Gaussian_Prob()
195         #Prior probability array
196         self.W = np.einsum("ij -> j",self.GAMMA) / self.N
197         Num = N * self.W
198         Den = np.reshape(np.sum(Num, axis=1), (self.N, 1) )
199         self.GAMMA = Num/Den
200
201     def Maximization_Step(self):
202         #In this step we updtae the various parameters
203
204         #Updation of GAMMA
205         self.Expectation_Step()
206
207         #Updation of W
208         self.Prior_Probability()
209
210         #Updation of Mean MU
211         self.Mean()
212
213         #Updation of Covariance Matrix SIGMA
214         self.Covariance_Matrix_Array()
215
216
217     def Log_Likelihood(self):
218
219         llhd = np.sum(np.log(self.Gaussian_Prob() @ self.W))
220
221         return llhd
222
223
224     def fit(self,max_iter,threshold):
225
226         log_likelihoods = [] #Attribute for 1D array that contains Log_Likelihood
227                             # values.
228                             #Size depends on the number iterations required
229                             # to converge
230
231
232         for i in range(max_iter):
233             self.Expectation_Step() #Updates Gamma

```

```

234         self.Maximization_Step() #Updates all the other parameters
235         log_likelihoods.append(self.Log_Likelihood())
236         #An if conditional for the requirement of convergence
237         if (i!=0) & ((log_likelihoods[i] - log_likelihoods[i-1]) < threshold):
238             break
239
240     print("Number of iterations to converge:" ,i)
241
242     def plot(self,ax,x1_grid,x2_grid):
243         # Plotting log_likelihood vs iterations, comment out if not needed
244         # sns.set_style("darkgrid") #setting the plot style
245         # fig = plt.figure(figsize=(10,10))
246         # ax0 = fig.add_subplot(111)
247         # ax0.set_title('Log-Likelihood')
248         # ax0.plot(range(i+1),log_likelihoods)
249
250         #Plot of the fitted Gaussians for each class
251         XY = np.array([x1_grid.flatten(),x2_grid.flatten()]).T
252
253         for mu,sigma in zip(self.MU,self.SIGMA):
254             multi_normal = mvn(mean=mu,cov=sigma)
255             contours = ax.contour(x1_grid, x2_grid, multi_normal.pdf(XY).reshape(
256                 len(x1_grid),len(x1_grid)),cmap='hsv',levels=4,extend='min')
257             ax.clabel(contours, inline=1, fontsize=5)
258
259     def Class_Prob(self,Y):
260         #A function that returns Prob.
261         # for a unlabelled vector Y to belong to a class
262         #Pred_Prob = []
263         Multi_Gauss = []
264         for mu,sigma in zip(self.MU,self.SIGMA):
265             Multi_Gauss.append(mvn(mean=mu,cov=sigma).pdf(Y))
266             #An array of Multi-Variate Gaussian Prob of various clusters
267         Wt_Gauss = np.einsum("i,i->i",self.W,Multi_Gauss)
268         #An array of weighted probabilities
269         Pred_Prob =np.sum(Wt_Gauss)
270         return Pred_Prob
271
272     # In[7]:
273
274
275     #Fitting gaussian mixtures for Class0
276     K,MU,GAMMA = Parameters_old(Class0,10)
277     #0 as the second argument chooses by default K_opt estimated using elbow method.
278     #If not pass the number of clusters needed
279
280     gmm0 = Gaussian_Mixture_Model(Class0,K,MU,GAMMA,1)
281     # 0 as the last argument -> diagonal covariance matrix.
282     # 1-> full covariance matrix.
283     gmm0.fit(max_iter=100,threshold = 1e-10)

```

```

284
285
286 # In[8]:
287
288
289 #Fitting gaussian mixtures for Class1
290 K,MU,GAMMA = Parameters_old(Class1,10)
291 gmm1 = Gaussian_Mixture_Model(Class1,K,MU,GAMMA,1)
292 gmm1.fit(max_iter=100,threshold = 1e-10)
293
294
295 # In[9]:
296
297
298 #Fitting gaussian mixtures for Class2
299 K,MU,GAMMA = Parameters_old(Class2,10)
300 gmm2 = Gaussian_Mixture_Model(Class2,K,MU,GAMMA,1)
301 gmm2.fit(max_iter=100,threshold = 1e-10)
302
303
304 # In[10]:
305 l1 = len(Class0)
306 l2 = len(Class1)
307 l3 = len(Class2)
308 total = l1+l2+l3
309
310 prior = []
311 prior.append(l1/total)
312 prior.append(l2/total)
313 prior.append(l3/total)
314
315 # We have fitted gaussians to each class and now we would like to make prediction
316 # for unlabelled points
317 def Class_Prediction(Y):
318     # gmm0,gmm1,gmm2 are the instances of class 0, class 1 and class 2 respectively
319     n = len(Y) #number of unlabelled examples
320     prediction = []
321     for example in range(n):
322         Prob=[]
323         Prob = [gmm0.Class_Prob(Y[example,:])*prior[0], gmm1.Class_Prob(Y[example,:])]
324         prediction.append(np.argmax(Prob))
325     # print("Labels for the given dataset:", prediction)
326     return prediction
327
328 # In[11]:
329
330 header_names = ['x1', 'x2' , 'Class']
331 D = pd.read_csv('datasets/Dataset_1b/dev.csv', header = None, names = header_names)
332 D.head()
333

```

```

334 # In[12]:
335
336 L_df = D.loc[:,['x1','x2']]
337 X_dev = L_df.to_numpy()
338
339 lab_df = D.loc[:, 'Class']
340 y_dev = lab_df.to_numpy()
341
342 # In[12]:
343
344 # Divide into test and validation set
345 from sklearn.model_selection import train_test_split
346 X_val,X_test,y_val,y_test = train_test_split(X_dev,y_dev, test_size=0.5)
347
348 # In[14]:
349 from sklearn.metrics import accuracy_score
350 y_train = labels
351 X_train = np.concatenate((Class0,Class1,Class2))
352
353 predictions = Class_Prediction(X_train)
354
355 train_accuracy = accuracy_score(y_train,predictions)*100
356 print("train accuracy: " , train_accuracy)
357
358 # plot confusion matrix for training data
359 c_matrix = confusion_matrix(y_train, predictions)
360 fig, ax = plot_confusion_matrix(conf_mat=c_matrix,figsize=(7,7),cmap=plt.cm.RdYlBu_r)
361 ax.set(title = "Confusion Matrix")
362 plt.show()
363
364 # In[15]:
365 predictions = Class_Prediction(X_val)
366
367 val_accuracy = accuracy_score(y_val,predictions)*100
368 print("val accuracy: " , val_accuracy)
369
370 # In[16]:
371 predictions = Class_Prediction(X_test)
372
373 test_accuracy = accuracy_score(y_test,predictions)*100
374 print("test accuracy: " , test_accuracy)
375
376 # plot confusion matrix for test data
377 c_matrix = confusion_matrix(y_test, predictions)
378 fig, ax = plot_confusion_matrix(conf_mat=c_matrix,figsize=(7,7),cmap=plt.cm.RdYlBu_r)
379 ax.set(title = "Confusion Matrix")
380 plt.show()
381
382 # In[16]:
383

```

```

384 # Plot decision surface with training data and GMM superimposed
385 # define bounds of the domain
386 min1, max1 = X_train[:, 0].min()-1, X_train[:, 0].max()+1
387 min2, max2 = X_train[:, 1].min()-1, X_train[:, 1].max()+1
388
389 # define the x and y scale
390 x1_grid = np.arange(min1, max1, 0.1)
391 x2_grid = np.arange(min2, max2, 0.1)
392
393 x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
394
395 c1, c2 = x1_grid.flatten(), x1_grid.flatten()
396 c1, c2 = x1_grid.reshape((len(c1), 1)), x2_grid.reshape((len(c2), 1))
397
398 x = np.hstack((c1,c2))
399
400 # predict
401 predictions = np.array(Class_Prediction(x))
402
403 x3_grid = predictions.reshape(x1_grid.shape)
404
405 # plot decision surfaxe
406 fig = plt.figure(figsize=[13,13])
407 ax = fig.add_subplot(111)
408 ax.contourf(x1_grid, x2_grid, x3_grid, cmap='Pastel1')
409 ax.scatter(X_train[:,0],X_train[:,1],marker='x')
410 gmm0.plot(ax,x1_grid,x2_grid) # call to plot gaussian functions of class 0
411 gmm1.plot(ax,x1_grid,x2_grid) # call to plot gaussian functions of class 1
412 gmm2.plot(ax,x1_grid,x2_grid) # call to plot gaussian functions of class 2
413 ax.set_xlabel('x1',fontsize=20)
414 ax.set_ylabel('x2',fontsize=20)
415 ax.set_title('Bayes Classifier with GMM', fontsize=20)

```

III Static Pattern Classification on Real World Dataset 2A

III.1 Python Code

```
1 import numpy as np
2 import pandas as pd
3 from sklearn import preprocessing
4 from sklearn.cluster import KMeans
5 from matplotlib import pyplot as plt
6 import math
7
8 #scaling factor
9 scalingfac = 1
10
11 #extracting and parsing data
12 #preprocessing : quantile transformation
13 f = pd.read_csv('Dataset_2A/coast/train.csv')
14 f1 = f.to_numpy()
15 data1 = f1[:,1:]
16 data1 = data1.astype('float')
17 data1 = np.apply_along_axis(lambda x:np.append(x,np.array([1,0,0,0,0])),1,data1)
18 X1 = data1[:, :-5]
19 X1 = scalingfac*X1
20 X1 = (preprocessing.QuantileTransformer(random_state=0)).fit_transform(X1)
21 f = pd.read_csv('Dataset_2A/forest/train.csv')
22 f1 = f.to_numpy()
23 data2 = f1[:,1:]
24 data2 = data2.astype('float')
25 data2 = np.apply_along_axis(lambda x:np.append(x,np.array([0,1,0,0,0])),1,data2)
26 X2 = data2[:, :-5]
27 X2 = scalingfac*X2
28 X2 = (preprocessing.QuantileTransformer(random_state=0)).fit_transform(X2)
29 f = pd.read_csv('Dataset_2A/mountain/train.csv')
30 f1 = f.to_numpy()
31 data3 = f1[:,1:]
32 data3 = data3.astype('float')
33 data3 = np.apply_along_axis(lambda x:np.append(x,np.array([0,0,1,0,0])),1,data3)
34 X3 = data3[:, :-5]
35 X3 = scalingfac*X3
36 X3 = (preprocessing.QuantileTransformer(random_state=0)).fit_transform(X3)
37 f = pd.read_csv('Dataset_2A/opencountry/train.csv')
38 f1 = f.to_numpy()
39 data4 = f1[:,1:]
40 data4 = data4.astype('float')
41 data4 = np.apply_along_axis(lambda x:np.append(x,np.array([0,0,0,1,0])),1,data4)
42 X4 = data4[:, :-5]
43 X4 = scalingfac*X4
44 X4 = (preprocessing.QuantileTransformer(random_state=0)).fit_transform(X4)
45 f = pd.read_csv('Dataset_2A/street/train.csv')
46 f1 = f.to_numpy()
```

```

47 data5 = f1[:,1:]
48 data5 = data5.astype('float')
49 data5 = np.apply_along_axis(lambda x:np.append(x,np.array([0,0,0,0,1])),1,data5)
50 X5 = data5[:, :-5]
51 X5 = scalingfac*X5
52 X5 = (preprocessing.QuantileTransformer(random_state=0)).fit_transform(X5)
53
54 data = np.concatenate((data1,data2,data3,data4,data5),axis=0)
55
56 #prior probabilities
57 py1 = len(data1)/len(data)
58 py2 = len(data2)/len(data)
59 py3 = len(data3)/len(data)
60 py4 = len(data4)/len(data)
61 py5 = len(data5)/len(data)
62 ppy = np.array([py1,py2,py3,py4,py5])
63
64 #hyperparameters
65 threshold = 1e-12
66 nclasses = 5
67 #option for diagonal(opt=0) or full(opt=1) covariance matrix
68 opt = int(input('covariance'))
69 #number of clusters in each class
70 q = list(map(int,input('clusters').split(' ')))
71
72 #initialisation of parameters
73 fweights = {}
74 fmeans = {}
75 fvariances = {}
76
77 #GMM for each class
78 def GMMperclass(X,q,threshold,opt): #X = data q = cluster #hyperparameter
79     dim = X.shape[1]
80     N = len(X)
81
82     #kmeans clustering
83     kmeans = KMeans(init='random',n_clusters=q,n_init=20,max_iter=800)
84     kmeans.fit(X)
85     response = np.zeros((N,q))
86     labl = kmeans.labels_
87
88     #gamma matrix
89     for i in range(len(labl)):
90         response[i,labl[i]] = 1
91
92     #gaussian for individual datapoint
93     def gauss(x,u,v):
94         dim = len(x)
95         num = ((np.reshape((x-u),(1,dim)))@((np.linalg.inv(v))@
96             (np.reshape((x-u),(dim,1)))))

```

```

97         num = -num/2
98         den = np.sqrt(((2*np.pi)**dim)*(np.linalg.det(v)))
99         return ((np.exp(num))/den)
100
101     #gaussian for whole datapoints
102     def gaussmat(X,q,means,variances):
103         N,dim = X.shape
104         #Inverse of the covariance matrix
105         sigma_inv = np.linalg.inv(variances)
106         mean_shift = np.reshape(X, (N, 1, dim) ) - np.reshape(means, (1,q, dim) )
107         #Normalisation term of the Gaussian dist
108         norm = np.sqrt(((2*np.pi)**dim)*np.linalg.det(variances))
109         #Exponential term of the Gaussian
110         expo = np.exp(-0.5*(np.einsum("nkj,nkj->nk", np.einsum("nki,kij->nkj",
111             mean_shift, sigma_inv),mean_shift)))
112         return expo/norm
113
114     #updating gamma=response update
115     def responseupdate(weights,X,q,means,variances):
116         N,dim = X.shape
117         No = gaussmat(X,q,means,variances)
118         Num = No * weights
119         Den = np.reshape(np.sum(Num, axis=1), (N, 1) )
120         return Num/Den
121
122     #estimation of log likelihoods
123     def llhd(weights,X,q,means,variances):
124         return np.sum(np.log(gaussmat(X,q,means,variances) @ weights))
125
126     ##initialisation
127     weights = np.einsum("ij -> j",response) / N
128     means = (((response).T) @ X) / np.reshape((weights*N), (q, 1))
129     Nki = N*weights
130     mean_shifti = np.reshape(X, (N, 1, dim) ) - np.reshape(means, (1, q, dim) )
131     sigmai = np.einsum("nki,nkj->kij", np.einsum("nk,nki->nki", response,
132         mean_shifti), mean_shifti) / np.reshape(Nki, (q, 1, 1))
133
134     #full covariance matrix
135     if opt == 1:
136         variances = sigmai
137     #diagonal covariance matrix
138     if opt == 0:
139         I = np.identity(dim)
140         variances = np.einsum("kij,ij -> kij",sigmai,I)
141
142     #initial log likelihood
143     NLL = llhd(weights,X,q,means,variances)
144     nweights = weights
145     nmeans = means
146     nvariances = variances

```

```

147     OLL = NLL+10
148     ite = 0
149
150     #EM maximisation
151     while abs(NLL-OLL)>=threshold:
152         OLL = NLL
153         #gamma calculation
154         nresponse = responseupdate(nweights,X,q,nmeans,nvariances)
155         #updatation
156         nweights = np.einsum("ij -> j",nresponse)/N
157         nmeans = (((nresponse).T) @X) / np.reshape((nweights*N), (q, 1))
158         Nk = N*nweights
159         mean_shift = np.reshape(X, (N, 1, dim) ) - np.reshape(nmeans, (1, q, dim) )
160         sigma = np.einsum("nki,nkj->kij", np.einsum("nk,nki->nki",
161             nresponse, mean_shift), mean_shift) / np.reshape(Nk, (q, 1, 1))
162
163         #full covariance matrix
164         if opt == 1:
165             nvariances = sigma
166
167         #full diagonal matrix
168         if opt == 0:
169             I = np.identity(dim)
170             nvariances = np.einsum("kij,ij -> kij",sigma,I)
171
172
173         NLL = llhd(nweights,X,q,nmeans,nvariances)
174         ite+=1
175     print("iterations=%f"%ite)
176
177     #final parameters
178     fweights = nweights
179     fmeans = nmeans
180     fvariances = nvariances
181     return fweights,fmeans,fvariances
182
183 ##training
184
185 fweights[0],fmeans[0],fvariances[0] = GMMperclass(X1,q[0],threshold,opt)
186 fweights[1],fmeans[1],fvariances[1] = GMMperclass(X2,q[1],threshold,opt)
187 fweights[2],fmeans[2],fvariances[2] = GMMperclass(X3,q[2],threshold,opt)
188 fweights[3],fmeans[3],fvariances[3] = GMMperclass(X4,q[3],threshold,opt)
189 fweights[4],fmeans[4],fvariances[4] = GMMperclass(X5,q[4],threshold,opt)
190
191 #modeloutput
192
193 def bayesclf(x,ppy,nclasses,q,fweights,fmeans,fvariances):
194     def gauss(x,u,v):
195         dim = len(x)
196         num = ((np.reshape((x-u),(1,dim)))@((np.linalg.inv(v))@

```

```

197         (np.reshape((x-u),(dim,1))))))
198     num = -num/2
199     den = np.sqrt(((2*np.pi)**dim)*(np.linalg.det(v)))
200     return ((np.exp(num))/den)
201 def gaussmat(X,q,means,variances):
202     N,dim = X.shape
203     sigma_inv = np.linalg.inv(variances)
204     mean_shift = np.reshape(X, (N, 1, dim) ) - np.reshape(means, (1,q, dim) )
205     norm = np.sqrt(((2*np.pi)**dim)*np.linalg.det(variances))
206     expo = np.exp(-0.5*(np.einsum("nkj,nkj->nk", np.einsum("nki,kij->nkj",
207         mean_shift, sigma_inv),mean_shift)))
208     return expo/norm
209 pxy = np.zeros((nclasses,1))
210 pyx = np.zeros((nclasses,1))
211 res = np.zeros((nclasses,1))
212 for i in range(nclasses):
213     pxy[i] = np.sum((gaussmat(np.reshape(x,(1,len(x))),q[i],fmeans[i],
214         fvariances[i]))*(fweights[i]))
215 for i in range(nclasses):
216     pyx[i] = pxy[i]*ppy[i]
217 pyx = pyx/np.sum(pyx)
218 res[np.argmax(pyx)] = 1
219 return np.transpose(res)
220
221 #train labels
222 ytr = data[:,-5:]
223 #predicted train labels
224 ytrp = np.apply_along_axis(lambda x: bayesclf(x,ppy,nclasses,q,fweights,fmeans,
225     fvariances),1,np.concatenate((X1,X2,X3,X4,X5),axis=0))
226 c = 0
227 for i in range(len(ytr)):
228     if np.linalg.norm((ytrp[i,:]-ytr[i,:])) < 1:
229         c +=1
230 print('training accuracy = %f'%(c/len(ytr)))
231
232 #####test
233
234 #extracting test data
235 #preprocessing : quantile transformation
236 f = pd.read_csv('Dataset_2A/coast/dev.csv')
237 f1 = f.to_numpy()
238 datat1 = f1[:,1:]
239 datat1 = datat1.astype('float')
240 datat1 = np.apply_along_axis(lambda x: np.append(x,np.array([1,0,0,0,0])),1,datat1)
241 Xt1 = datat1[:, :-5]
242 Xt1 = scalingfac*Xt1
243 Xt1 = ((preprocessing.QuantileTransformer(random_state=0)).fit(Xt1)).transform(Xt1)
244 f = pd.read_csv('Dataset_2A/forest/dev.csv')
245 f1 = f.to_numpy()
246 datat2 = f1[:,1:]

```

```

247 datat2 = datat2.astype('float')
248 datat2 = np.apply_along_axis(lambda x:np.append(x,np.array([0,1,0,0,0])),1,datat2)
249 Xt2 = datat2[:, :-5]
250 Xt2 = scalingfac*Xt2
251 Xt2 = ((preprocessing.QuantileTransformer(random_state=0)).fit(Xt2)).transform(Xt2)
252 f = pd.read_csv('Dataset_2A/mountain/dev.csv')
253 f1 = f.to_numpy()
254 datat3 = f1[:, 1:]
255 datat3 = datat3.astype('float')
256 datat3 = np.apply_along_axis(lambda x:np.append(x,np.array([0,0,1,0,0])),1,datat3)
257 Xt3 = datat3[:, :-5]
258 Xt3 = scalingfac*Xt3
259 Xt3 = ((preprocessing.QuantileTransformer(random_state=0)).fit(Xt3)).transform(Xt3)
260 f = pd.read_csv('Dataset_2A/opencountry/dev.csv')
261 f1 = f.to_numpy()
262 datat4 = f1[:, 1:]
263 datat4 = datat4.astype('float')
264 datat4 = np.apply_along_axis(lambda x:np.append(x,np.array([0,0,0,1,0])),1,datat4)
265 Xt4 = datat4[:, :-5]
266 Xt4 = scalingfac*Xt4
267 Xt4 = ((preprocessing.QuantileTransformer(random_state=0)).fit(Xt4)).transform(Xt4)
268 f = pd.read_csv('Dataset_2A/street/dev.csv')
269 f1 = f.to_numpy()
270 datat5 = f1[:, 1:]
271 datat5 = datat5.astype('float')
272 datat5 = np.apply_along_axis(lambda x:np.append(x,np.array([0,0,0,0,1])),1,datat5)
273 Xt5 = datat5[:, :-5]
274 Xt5 = scalingfac*Xt5
275 Xt5 = ((preprocessing.QuantileTransformer(random_state=0)).fit(Xt5)).transform(Xt5)
276
277 datat = np.concatenate((datat1,datat2,datat3,datat4,datat5),axis=0)
278 Xt = np.concatenate((Xt1,Xt2,Xt3,Xt4,Xt5),axis=0)
279 datat = np.concatenate((Xt,datat[:, -5:]),axis=1)
280
281 #test labels
282 yte = datat[:, -5:]
283
284 #predicted test labels
285 ytep = np.apply_along_axis(lambda x:bayesclf(x,ppy,nclasses,q,fweights,fmeans,
286                                     fvariances),1,datat[:, :-5])
287
288 cte = 0
289
290 for i in range(len(ytep)):
291     if np.linalg.norm((ytep[i, :] - yte[i, :])) < 1:
292         cte += 1
293
294 print('test accuracy = %f'%(cte/len(ytep)))
295
296

```

```

297
298 #plotting confusion matrix
299 from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
300
301 def clf_label(res):
302     if np.argmax(res) == 0:
303         return 'coast'
304     if np.argmax(res) == 1:
305         return 'forest'
306     if np.argmax(res) == 2:
307         return 'mountain'
308     if np.argmax(res) == 3:
309         return 'opencountry'
310     if np.argmax(res) == 4:
311         return 'street'
312
313 train_confusion = confusion_matrix([clf_label(ytr[i,:]) for i in range(len(ytr))],
314                                   [clf_label(ytrp[i,:]) for i in range(len(ytrp))], labels =
315                                   ['coast', 'forest', 'mountain', 'opencountry', 'street'])
316
317 test_confusion = confusion_matrix([clf_label(yte[i,:]) for i in range(len(yte))],
318                                  [clf_label(ytep[i,:]) for i in range(len(ytep))], labels =
319                                  ['coast', 'forest', 'mountain', 'opencountry', 'street'])
320
321 train_tab = ConfusionMatrixDisplay(train_confusion, display_labels =
322                                   ['coast', 'forest', 'mountain', 'opencountry', 'street'] )
323 plt.figure(1)
324 train_tab.plot()
325 plt.title('Training Data')
326 test_tab = ConfusionMatrixDisplay(test_confusion, display_labels =
327                                   ['coast', 'forest', 'mountain', 'opencountry', 'street'] )
328 plt.figure(2)
329 test_tab.plot()
330 plt.title('Test Data')

```

IV Static Pattern Classification on Real World Dataset 2B

IV.1 Python Code

```
1 import numpy as np
2 import pandas as pd
3 import os
4 from sklearn import preprocessing
5 from sklearn.cluster import KMeans
6 from matplotlib import pyplot as plt
7
8 #extracting and parsing files tr
9 #preprocessing : quantile transformation
10 directory = 'Dataset_2B/coast/train'
11 dataX = []
12 for filename in os.listdir(directory):
13     f = open(directory+'/'+filename)
14     data = []
15     for line in f:
16         data.append([float(x) for x in line.strip().split(' ')])
17     dataX+=(data)
18 data1 = np.array(dataX)
19 data1 = np.apply_along_axis(lambda x:np.append(x,np.array([1,0,0,0,0])),1,data1)
20 X1 = data1[:, :-5]
21 X1 = (preprocessing.QuantileTransformer(random_state=0)).fit_transform(X1)
22 directory = 'Dataset_2B/forest/train'
23 dataX = []
24 for filename in os.listdir(directory):
25     f = open(directory+'/'+filename)
26     data = []
27     for line in f:
28         data.append([float(x) for x in line.strip().split(' ')])
29     dataX+=(data)
30 data2 = np.array(dataX)
31 data2 = np.apply_along_axis(lambda x:np.append(x,np.array([0,1,0,0,0])),1,data2)
32 X2 = data2[:, :-5]
33 X2 = (preprocessing.QuantileTransformer(random_state=0)).fit_transform(X2)
34 directory = 'Dataset_2B/mountain/train'
35 dataX = []
36 for filename in os.listdir(directory):
37     f = open(directory+'/'+filename)
38     data = []
39     for line in f:
40         data.append([float(x) for x in line.strip().split(' ')])
41     dataX+=(data)
42 data3 = np.array(dataX)
43 data3 = np.apply_along_axis(lambda x:np.append(x,np.array([0,0,1,0,0])),1,data3)
44 X3 = data3[:, :-5]
45 X3 = (preprocessing.QuantileTransformer(random_state=0)).fit_transform(X3)
46 directory = 'Dataset_2B/opencountry/train'
```



```

47 dataX = []
48 for filename in os.listdir(directory):
49     f = open(directory+'/'+filename)
50     data = []
51     for line in f:
52         data.append([float(x) for x in line.strip().split(' ')])
53     dataX+=(data)
54 data4 = np.array(dataX)
55 data4 = np.apply_along_axis(lambda x:np.append(x,np.array([0,0,0,1,0])),1,data4)
56 X4 = data4[:, :-5]
57 X4 = (preprocessing.QuantileTransformer(random_state=0)).fit_transform(X4)
58 directory = 'Dataset_2B/street/train'
59 dataX = []
60 for filename in os.listdir(directory):
61     f = open(directory+'/'+filename)
62     data = []
63     for line in f:
64         data.append([float(x) for x in line.strip().split(' ')])
65     dataX+=(data)
66 data5 = np.array(dataX)
67 data5 = np.apply_along_axis(lambda x:np.append(x,np.array([0,0,0,0,1])),1,data5)
68 X5 = data5[:, :-5]
69 X5 = (preprocessing.QuantileTransformer(random_state=0)).fit_transform(X5)
70
71 data = np.concatenate((data1,data2,data3,data4,data5),axis=0)
72
73 #prior probabilities
74 py1 = len(data1)/len(data)
75 py2 = len(data2)/len(data)
76 py3 = len(data3)/len(data)
77 py4 = len(data4)/len(data)
78 py5 = len(data5)/len(data)
79 ppy = np.array([py1,py2,py3,py4,py5])
80
81 #hyperparameters
82 threshold = 1e-10
83 nclasses = 5
84 #option for diagonal(opt=0) or full(opt=1) covariance matrix
85 opt = int(input('covariance'))
86 #number of clusters in each class
87 q = list(map(int,input('clusters').split(' ')))
88
89 # initialisation of parameters
90 fweights = {}
91 fmeans = {}
92 fvariances = {}
93 #GMM for each class
94 def GMMperclass(X,q,threshold,opt): #X = data q = cluster #hyperparameter
95     dim = X.shape[1]
96     N = len(X)

```

```

97
98 #kmeans clustering
99 kmeans = KMeans(init='random',n_clusters=q,n_init=20,max_iter=800)
100 kmeans.fit(X)
101 response = np.zeros((N,q))
102 labl = kmeans.labels_
103
104 #gamma matrix
105 for i in range(len(labl)):
106     response[i,labl[i]] = 1
107
108 #gaussian for individual datapoint
109 def gauss(x,u,v):
110     dim = len(x)
111     num = ((np.reshape((x-u),(1,dim)))@((np.linalg.inv(v))@
112                                         (np.reshape((x-u),(dim,1)))))
113     num = -num/2
114     den = np.sqrt(((2*np.pi)**dim)*(np.linalg.det(v)))
115     return ((np.exp(num))/den)
116
117 #gaussian for whole datapoints
118 def gaussmat(X,q,means,variances):
119     N,dim = X.shape
120     #Inverse of the covariance matrix
121     sigma_inv = np.linalg.inv(variances)
122     mean_shift = np.reshape(X, (N, 1, dim) ) - np.reshape(means, (1,q, dim) )
123     #Normalisation term of the Gaussian dist
124     norm = np.sqrt(((2*np.pi)**dim)*np.linalg.det(variances))
125     #Exponential term of the Gaussian
126     expo = np.exp(-0.5*(np.einsum("nkj,nkj->nk", np.einsum("nki,kij->nkj",
127     mean_shift, sigma_inv),mean_shift)))
128     return expo/norm
129
130 #updating gamma=response update
131 def responseupdate(weights,X,q,means,variances):
132     N,dim = X.shape
133     No = gaussmat(X,q,means,variances)
134     Num = No * weights
135     Den = np.reshape(np.sum(Num, axis=1), (N, 1) )
136     return Num/Den
137
138 #estimation of log likelihoods
139 def llhd(weights,X,q,means,variances):
140     return np.sum(np.log(gaussmat(X,q,means,variances) @ weights))
141
142 ##initialisation
143 weights = np.einsum("ij -> j",response) / N
144 means = (((response).T) @ X) / np.reshape((weights*N), (q, 1))
145 Nki = N*weights
146 mean_shifti = np.reshape(X, (N, 1, dim) ) - np.reshape(means, (1, q, dim) )

```

```

147     sigmai = np.einsum("nki,nkj->kij", np.einsum("nk,nki->nki", response,
148         mean_shifti), mean_shifti) / np.reshape(Nki, (q, 1, 1))
149
150     #full covariance matrix
151     if opt == 1:
152         variances = sigmai
153     #diagonal covariance matrix
154     if opt == 0:
155         I = np.identity(dim)
156         variances = np.einsum("kij,ij -> kij",sigmai,I)
157
158     #initial log likelihood
159     NLL = llhd(weights,X,q,means,variances)
160     nweights = weights
161     nmeans = means
162     nvariances = variances
163     OLL = NLL+10
164     ite = 0
165
166     #EM maximisation
167     while abs(NLL-OLL)>=threshold:
168         OLL = NLL
169         #gamma calculation
170         nresponse = responseupdate(nweights,X,q,nmeans,nvariances)
171         #updation
172         nweights = np.einsum("ij -> j",nresponse)/N
173         nmeans = (((nresponse).T) @X) / np.reshape((nweights*N), (q, 1))
174         Nk = N*nweights
175         mean_shift = np.reshape(X, (N, 1, dim) ) - np.reshape(nmeans, (1, q, dim) )
176         sigma = np.einsum("nki,nkj->kij", np.einsum("nk,nki->nki",
177             nresponse, mean_shift), mean_shift)/ np.reshape(Nk, (q, 1, 1))
178
179         #full covariance matrix
180         if opt == 1:
181             nvariances = sigma
182
183         #full diagonal matrix
184         if opt == 0:
185             I = np.identity(dim)
186             nvariances = np.einsum("kij,ij -> kij",sigma,I)
187
188
189         NLL = llhd(nweights,X,q,nmeans,nvariances)
190         ite+=1
191     print("iterations=%f"%ite)
192
193     #final parameters
194     fweights = nweights
195     fmeans = nmeans
196     fvariances = nvariances

```

```

197     return fweights,fmeans,fvariances
198
199 ##training
200
201 fweights[0],fmeans[0],fvariances[0] = GMMperclass(X1,q[0],threshold,opt)
202 fweights[1],fmeans[1],fvariances[1] = GMMperclass(X2,q[1],threshold,opt)
203 fweights[2],fmeans[2],fvariances[2] = GMMperclass(X3,q[2],threshold,opt)
204 fweights[3],fmeans[3],fvariances[3] = GMMperclass(X4,q[3],threshold,opt)
205 fweights[4],fmeans[4],fvariances[4] = GMMperclass(X5,q[4],threshold,opt)
206
207 # classification for variable length features
208
209 def bayesclfvarlength(X,ppy,nclasses,q,fweights,fmeans,fvariances):
210     def gauss(x,u,v):
211         dim = len(x)
212         num = ((np.reshape((x-u),(1,dim)))@((np.linalg.inv(v))@
213             (np.reshape((x-u),(dim,1))))))
214         num = -num/2
215         den = np.sqrt(((2*np.pi)**dim)*(np.linalg.det(v)))
216         return ((np.exp(num))/den)
217     def gaussmat(X,q,means,variances):
218         N,dim = X.shape
219         sigma_inv = np.linalg.inv(variances) #Inverse of the covariance matrix
220         mean_shift = np.reshape(X, (N, 1, dim) ) - np.reshape(means, (1,q, dim) )
221         norm = np.sqrt(((2*np.pi)**dim)*np.linalg.det(variances))
222         #Normalisation term of the Gaussian dist.
223         #Exponential term of the Gaussian
224         expo = np.exp(-0.5*(np.einsum("nkj,nkj->nk", np.einsum("nki,kij->nkj",
225             mean_shift, sigma_inv),mean_shift)))
226         return expo/norm
227     pxy = np.zeros((nclasses,1))
228     pyx = np.zeros((nclasses,1))
229     res = np.zeros((nclasses,1))
230     for i in range(nclasses):
231         pxy[i] = np.prod(np.sum(gaussmat(X,q[i],fmeans[i],fvariances[i])*
232             fweights[i],axis=1))
233     for i in range(nclasses):
234         pyx[i] = pxy[i]*ppy[i]
235     pyx = pyx/np.sum(pyx)
236     #res[np.argmax(pyx)] = 1
237     #return np.transpose(res)
238     return np.argmax(pyx)
239
240 #prediction of training labels
241 #reorganising of data points to images
242 Xn1 = []
243 for i in range(int(X1.shape[0]/36)):
244     Xn1.append(X1[(36*i):(36*(i+1)),:])
245 Xn1 = np.array(Xn1)
246 Xn2 = []

```

```

247 for i in range(int(X2.shape[0]/36)):
248     Xn2.append(X2[(36*i):(36*(i+1)),:])
249 Xn2 = np.array(Xn2)
250 Xn3 = []
251 for i in range(int(X3.shape[0]/36)):
252     Xn3.append(X3[(36*i):(36*(i+1)),:])
253 Xn3 = np.array(Xn3)
254 Xn4 = []
255 for i in range(int(X4.shape[0]/36)):
256     Xn4.append(X4[(36*i):(36*(i+1)),:])
257 Xn4 = np.array(Xn4)
258 Xn5 = []
259 for i in range(int(X5.shape[0]/36)):
260     Xn5.append(X5[(36*i):(36*(i+1)),:])
261 Xn5 = np.array(Xn5)
262
263
264
265 c = 0
266 for i in range(Xn1.shape[0]):
267     if bayesclfvarlength(Xn1[i,:,:],ppy,nclasses,q,fweights,fmeans,fvariances) == 0:
268         c += 1
269 for i in range(Xn2.shape[0]):
270     if bayesclfvarlength(Xn2[i,:,:],ppy,nclasses,q,fweights,fmeans,fvariances) == 1:
271         c += 1
272 for i in range(Xn3.shape[0]):
273     if bayesclfvarlength(Xn3[i,:,:],ppy,nclasses,q,fweights,fmeans,fvariances) == 2:
274         c += 1
275 for i in range(Xn4.shape[0]):
276     if bayesclfvarlength(Xn4[i,:,:],ppy,nclasses,q,fweights,fmeans,fvariances) == 3:
277         c += 1
278 for i in range(Xn5.shape[0]):
279     if bayesclfvarlength(Xn5[i,:,:],ppy,nclasses,q,fweights,fmeans,fvariances) == 4:
280         c += 1
281
282 print('training accuracy =
283 %f'%(c/((X1.shape[0]+X2.shape[0]+X3.shape[0]+X4.shape[0]+X5.shape[0])/36)))
284
285
286
287 #####testing
288
289 directory = 'Dataset_2B/coast/dev'
290 dataX = []
291 for filename in os.listdir(directory):
292     f = open(directory+'/'+filename)
293     data =[]
294     for line in f:
295         data.append([float(x) for x in line.strip().split(' ')])
296     dataX+=(data)

```

```

297 datat1 = np.array(dataX)
298 datat1 = np.apply_along_axis(lambda x:np.append(x,np.array([1,0,0,0,0])),1,datat1)
299 Xt1 = datat1[:, :-5]
300 Xt1 = ((preprocessing.QuantileTransformer(random_state=0)).fit(Xt1)).transform(Xt1)
301 directory = 'Dataset_2B/forest/dev'
302 dataX = []
303 for filename in os.listdir(directory):
304     f = open(directory+'/'+filename)
305     data = []
306     for line in f:
307         data.append([float(x) for x in line.strip().split(' ')])
308     dataX+=(data)
309 datat2 = np.array(dataX)
310 datat2 = np.apply_along_axis(lambda x:np.append(x,np.array([0,1,0,0,0])),1,datat2)
311 Xt2 = datat2[:, :-5]
312 Xt2= ((preprocessing.QuantileTransformer(random_state=0)).fit(Xt2)).transform(Xt2)
313 directory = 'Dataset_2B/mountain/dev'
314 dataX = []
315 for filename in os.listdir(directory):
316     f = open(directory+'/'+filename)
317     data = []
318     for line in f:
319         data.append([float(x) for x in line.strip().split(' ')])
320     dataX+=(data)
321 datat3 = np.array(dataX)
322 datat3 = np.apply_along_axis(lambda x:np.append(x,np.array([0,0,1,0,0])),1,datat3)
323 Xt3 = datat3[:, :-5]
324 Xt3= ((preprocessing.QuantileTransformer(random_state=0)).fit(Xt3)).transform(Xt3)
325 directory = 'Dataset_2B/opencountry/dev'
326 dataX = []
327 for filename in os.listdir(directory):
328     f = open(directory+'/'+filename)
329     data = []
330     for line in f:
331         data.append([float(x) for x in line.strip().split(' ')])
332     dataX+=(data)
333 datat4 = np.array(dataX)
334 datat4 = np.apply_along_axis(lambda x:np.append(x,np.array([0,0,0,1,0])),1,datat4)
335 Xt4 = datat4[:, :-5]
336 Xt4= ((preprocessing.QuantileTransformer(random_state=0)).fit(Xt4)).transform(Xt4)
337 directory = 'Dataset_2B/street/dev'
338 dataX = []
339 for filename in os.listdir(directory):
340     f = open(directory+'/'+filename)
341     data = []
342     for line in f:
343         data.append([float(x) for x in line.strip().split(' ')])
344     dataX+=(data)
345 datat5 = np.array(dataX)
346 datat5 = np.apply_along_axis(lambda x:np.append(x,np.array([0,0,0,0,1])),1,datat5)

```

```

347 Xt5 = datat5[:, :-5]
348 Xt5= ((preprocessing.QuantileTransformer(random_state=0)).fit(Xt5)).transform(Xt5)
349
350
351 #prediction of test labels
352 #reorganising of data points to images
353 Xtn1 = []
354 for i in range(int(Xt1.shape[0]/36)):
355     Xtn1.append(Xt1[(36*i):(36*(i+1)),:])
356 Xtn1 = np.array(Xtn1)
357 Xtn2 = []
358 for i in range(int(Xt2.shape[0]/36)):
359     Xtn2.append(Xt2[(36*i):(36*(i+1)),:])
360 Xtn2 = np.array(Xtn2)
361 Xtn3 = []
362 for i in range(int(Xt3.shape[0]/36)):
363     Xtn3.append(Xt3[(36*i):(36*(i+1)),:])
364 Xtn3 = np.array(Xtn3)
365 Xtn4 = []
366 for i in range(int(Xt4.shape[0]/36)):
367     Xtn4.append(Xt4[(36*i):(36*(i+1)),:])
368 Xtn4 = np.array(Xtn4)
369 Xtn5 = []
370 for i in range(int(Xt5.shape[0]/36)):
371     Xtn5.append(Xt5[(36*i):(36*(i+1)),:])
372 Xtn5 = np.array(Xtn5)
373
374
375
376 ct = 0
377 for i in range(Xtn1.shape[0]):
378     if bayesclfvarlength(Xtn1[i,:,:],ppy,nclasses,q,fweights,fmeans,fvariances) == 0:
379         ct += 1
380 for i in range(Xtn2.shape[0]):
381     if bayesclfvarlength(Xtn2[i,:,:],ppy,nclasses,q,fweights,fmeans,fvariances) == 1:
382         ct += 1
383 for i in range(Xtn3.shape[0]):
384     if bayesclfvarlength(Xtn3[i,:,:],ppy,nclasses,q,fweights,fmeans,fvariances) == 2:
385         ct += 1
386 for i in range(Xtn4.shape[0]):
387     if bayesclfvarlength(Xtn4[i,:,:],ppy,nclasses,q,fweights,fmeans,fvariances) == 3:
388         ct += 1
389 for i in range(Xtn5.shape[0]):
390     if bayesclfvarlength(Xtn5[i,:,:],ppy,nclasses,q,fweights,fmeans,fvariances) == 4:
391         ct += 1
392
393 print('test accuracy
394 = %f'%(ct/((Xt1.shape[0]+Xt2.shape[0]+Xt3.shape[0]+Xt4.shape[0]+Xt5.shape[0])/36)))
395
396 ##confusion matrix

```

```

397 img_labels = ['coast', 'forest', 'mountain', 'opencountry', 'street']
398 yt = []
399 ytp = []
400 for i in range(Xn1.shape[0]):
401     yt.append(0)
402     ytp.append(bayescfvarlength(Xn1[i,:,:],ppy,nclasses,q,fweights,fmeans,
403                               fvariances))
404 for i in range(Xn2.shape[0]):
405     yt.append(1)
406     ytp.append(bayescfvarlength(Xn2[i,:,:],ppy,nclasses,q,fweights,fmeans,
407                               fvariances))
408 for i in range(Xn3.shape[0]):
409     yt.append(2)
410     ytp.append(bayescfvarlength(Xn3[i,:,:],ppy,nclasses,q,fweights,fmeans,
411                               fvariances))
412 for i in range(Xn4.shape[0]):
413     yt.append(3)
414     ytp.append(bayescfvarlength(Xn4[i,:,:],ppy,nclasses,q,fweights,fmeans,
415                               fvariances))
416 for i in range(Xn5.shape[0]):
417     yt.append(4)
418     ytp.append(bayescfvarlength(Xn5[i,:,:],ppy,nclasses,q,fweights,fmeans,
419                               fvariances))
420
421 yte = []
422 ytep = []
423 for i in range(Xtn1.shape[0]):
424     yte.append(0)
425     ytep.append(bayescfvarlength(Xtn1[i,:,:],ppy,nclasses,q,fweights,fmeans,
426                               fvariances))
427 for i in range(Xtn2.shape[0]):
428     yte.append(1)
429     ytep.append(bayescfvarlength(Xtn2[i,:,:],ppy,nclasses,q,fweights,fmeans,
430                               fvariances))
431 for i in range(Xtn3.shape[0]):
432     yte.append(2)
433     ytep.append(bayescfvarlength(Xtn3[i,:,:],ppy,nclasses,q,fweights,fmeans,
434                               fvariances))
435 for i in range(Xtn4.shape[0]):
436     yte.append(3)
437     ytep.append(bayescfvarlength(Xtn4[i,:,:],ppy,nclasses,q,fweights,fmeans,
438                               fvariances))
439 for i in range(Xtn5.shape[0]):
440     yte.append(4)
441     ytep.append(bayescfvarlength(Xtn5[i,:,:],ppy,nclasses,q,fweights,fmeans,
442                               fvariances))
443
444 from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
445
446 train_confusion = confusion_matrix(yt,ytp)

```



```
447
448 test_confusion = confusion_matrix(yte,ytep)
449
450 train_tab = ConfusionMatrixDisplay(train_confusion,display_labels =
451 ['coast','forest','mountain','opencountry','street'] )
452 plt.figure(1)
453 train_tab.plot()
454 plt.title('Training Data')
455 test_tab = ConfusionMatrixDisplay(test_confusion,display_labels =
456 ['coast','forest','mountain','opencountry','street'] )
457 plt.figure(2)
458 test_tab.plot()
459 plt.title('Test Data')
```
