Pattern Recognition and Machine Learning

Assignment 2
Code

Group No. 19

Ranjith Tevnan	Sai Bandawar	Jay shah
EE18B146	EE18B150	EE18B158

22,April 2021

1 Dataset 1: 2-dimensional artificial data

1.(a) Linearly seperable data set for static pattern classification

1.(a).1 Code for K-nearest neighbours classifier, for K=1, K=7, and K=15

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from statistics import mode
from sklearn.metrics import accuracy_score
from joblib import Parallel, delayed
import multiprocessing
import matplotlib
import matplotlib.patches as mpatches
from sklearn.metrics import confusion_matrix
import seaborn as sn

plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['font.size'] = 15
plt.rcParams["figure.figsize"] = (8,8)

def KNN(X_train,Y_train,k,X):
    predicted=[]
    for p in range(X.shape[0]):
        test=X[p]

        le=np.sum((test-X_train)**2,axis=1)
        distances=list()

        for i in range(len(Y_train)):
            distances.append((Y_train[i],le[i]))

        distances.sort(key=lambda tup: tup[1])
        neighbors = list()
        for i in range(k):
            neighbors.append(distances[i][0])
        predicted.append(mode(neighbors))


    return predicted


def KNN_single(X_train,Y_train,k,X):
    predicted=[]
    for p in range(1):
        test=X

        le=np.sum((test-X_train)**2,axis=1)
```

```
        distances=list()

        for i in range(len(Y_train)):
            distances.append((Y_train[i],le[i]))

        distances.sort(key=lambda tup: tup[1])
        neighbors = list()
        for i in range(k):
            neighbors.append(distances[i][0])
        predicted.append(mode(neighbors))

    return predicted[0]



#Taking input from csv file and taking x and y out
data=pd.read_csv("19/train.csv",header=None)

data=data.to_numpy()


X_train=data[:,0:2]
Y_train=data[:,2]


data=pd.read_csv("19/dev.csv",header=None)
data=data.to_numpy()

X_valid=data[0:60,0:2]
Y_valid=data[0:60,2]

X_test=data[60:120,0:2]
Y_test=data[60:120,2]


K=[1,7,15]

#KNN classifier



for k in K:
    predicted=KNN(X_train,Y_train,k,X_valid)
    print("accuracy for k="+str(k)+" on validation set is "+str(accuracy_score(Y_valid,pr

for k in K:
    predicted=KNN(X_train,Y_train,k,X_train)
    print("accuracy for k="+str(k)+" on training set is "+str(accuracy_score(Y_train,pred

k=7
print("accuracy for k="+str(7)+" (Best Model) on testing set is "+str(accuracy_score(Y_te
```

```
predicted=KNN(X_train,Y_train,k,X_train)
confuse=confusion_matrix(Y_train,predicted)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for KNN with k=7 on Training data')
plt.savefig('Confusion_train_1.png')

plt.show()

predicted=KNN(X_train,Y_train,k,X_test)
confuse=confusion_matrix(Y_test,predicted)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for KNN with k=7 on Testing data')
plt.savefig('Confusion_test_1.png')

plt.show()



x1=np.linspace(-15,14,num=350)
x2=np.linspace(-3,14,num=350)
xx1, xx2 = np.meshgrid(x1, x2)
r1, r2 = xx1.flatten(), xx2.flatten()
r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
grid = np.hstack((r1,r2))
#print(grid)
#predicted.clear()

num_cores = multiprocessing.cpu_count()
#print(X_valid[0].shape[1])
#print(KNN(X_train,Y_train,15,X_valid[0]))
predicted = Parallel(n_jobs=num_cores)(delayed(KNN_single)(X_train,Y_train,7,grid[i]) for
#predicted = Parallel(n_jobs=num_cores)(delayed(knn)(grid[i],means,covdif,counts) for i i
# predicted=KNN(X_train,Y_train,7,grid)
#print(predicted)
predicted=np.array(predicted)


predicted=predicted.reshape(xx1.shape)


# In[2]:
```

```
fig = plt.figure(figsize=(8,8))
plt.contourf(xx1, xx2, predicted, cmap='RdBu')
colors = ['green','red','blue','purple']
plt.scatter(X_train[:,0], X_train[:,1], c=Y_train, cmap=matplotlib.colors.ListedColormap(
#plt.scatter(X_train[:,0], X_train[:,1], c=Y_train, cmap='RdBu')

plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.title('Decision Region plot for k=7 for each class',fontsize=20)

"""recs = []
for i in range(0,len(colors)):
    recs.append(mpatches.Rectangle((0,0),1,1,fc=colors[i]))
plt.legend(recs,unique,loc=4)
"""

plt.savefig('plot_KNN_1_part1.png')
plt.show()
```

1.(a).2 Code for Naive-Bayes classifier with a Gaussian distribution for each class

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from statistics import mode
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import matplotlib
import matplotlib.patches as mpatches
from joblib import Parallel, delayed
import multiprocessing
from scipy.stats import multivariate_normal
import seaborn as sn

def predict(x,means,sigma2,counts):  #S
tp=np.sum((x-means)**2,axis=1)
tp=tp/(-2*sigma2)
tp=tp+np.log(counts)
return np.argmax(tp)

def predict_cov(x,means,cov,counts):
tp=x-means
tp=np.dot(np.dot(tp,np.linalg.inv(cov)),tp.T)
value=[]
for i in range(tp.shape[0]):
value.append(tp[i][i])
value=np.array(value)
value=value/-2
return np.argmax(value+np.log(counts))

def predict_covdif(x,means,covdif,counts):
tp=x-means
value=[]
for i in range(tp.shape[0]):
le=tp[i,:].reshape(len(tp[i,:]),1)
value.append((np.dot(np.dot(le.T,np.linalg.inv(covdif[i,:,:])),le))[0][0]+np.log(np.linal
value=np.array(value)
value=value/-2
return np.argmax(value+np.log(counts))




#Taking input from csv file and taking x and y out
data=pd.read_csv("19/train.csv",header=None)

data=data.to_numpy()
```

```
X_train=data[:,0:2]
Y_train=data[:,2]


data=pd.read_csv("19/dev.csv",header=None)
data=data.to_numpy()

X_valid=data[0:60,0:2]
Y_valid=data[0:60,2]

X_test=data[60:120,0:2]
Y_test=data[60:120,2]


#print(type(Y_train))
(unique, counts) = np.unique(Y_train, return_counts=True)
#print(unique.shape[0])
means=np.zeros((unique.shape[0],X_train.shape[1]))
#print(means)

for i in range(X_train.shape[0]):
means[np.where(unique==Y_train[i]),:]+=X_train[i]
#print(means)
means=means/(np.tile(counts,(means.shape[1],1)).T)
#print(means)
sum=0
#print(X_train[0]-means[np.where(unique==Y_train[0]),:])
for i in range(X_train.shape[0]):
le=(X_train[i]-means[np.where(unique==Y_train[i]),:])
sum=sum+np.dot(le[0],le[0].T)

#print(sum)
sigma2=sum[0][0]/(X_train.shape[0]*X_train.shape[1])
predicted=[]
for i in range(len(Y_train)):
predicted.append(predict(X_train[i],means,sigma2,counts))

print("accuracy on training set with covariance matrix as same sigma2 is "+str(accuracy_s

predicted.clear()

for i in range(len(Y_valid)):
predicted.append(predict(X_valid[i],means,sigma2,counts))

print("accuracy on validation set with covariance matrix as same sigma2 is "+str(accuracy

cov=np.zeros((X_train.shape[1],X_train.shape[1]))
#print(cov)
for i in range(X_train.shape[0]):
tp=(X_train[i]-means[np.where(unique==Y_train[i]),:])
```

```
le=np.dot(tp[0].T,tp[0])
cov=cov+le

cov=cov/(X_train.shape[0])
d= np.diag(cov)
cov=np.diag(d)

#print((X_train-means).reshape(len(X_train[0]),1))
#print(predict_cov(X_train[5],means,cov,counts))
predicted.clear()

for i in range(len(Y_train)):
predicted.append(predict_cov(X_train[i],means,cov,counts))

print("accuracy on training set with covariance matrix as same cov is "+str(accuracy_scor

predicted.clear()

for i in range(len(Y_valid)):
predicted.append(predict_cov(X_valid[i],means,cov,counts))

print("accuracy on validation set with covariance matrix as same cov is "+str(accuracy_sc

covdif=np.zeros((len(unique),X_train.shape[1],X_train.shape[1]))
#print(covdif)
for i in range(X_train.shape[0]):
tp=(X_train[i]-means[np.where(unique==Y_train[i]),:])
le=np.dot(tp[0].T,tp[0])
covdif[np.where(unique==Y_train[i]),:,:]=covdif[np.where(unique==Y_train[i]),:,:]+le


for i in range(len(unique)):
covdif[i,:,:]=covdif[i,:,:]/counts[i]
d= np.diag(covdif[i,:,:])
covdif[i,:,:]=np.diag(d)


predicted.clear()

for i in range(len(Y_train)):
predicted.append(predict_covdif(X_train[i],means,covdif,counts))

print("accuracy on training set with covariance matrix as difcov is "+str(accuracy_score(
#print("Confusion Matrix for training data is: ")
#print(confusion_matrix(Y_train, predicted))

predicted.clear()

for i in range(len(Y_valid)):
predicted.append(predict_covdif(X_valid[i],means,covdif,counts))
```

```
print("accuracy on validation set with covariance matrix as difcov is "+str(accuracy_scor
#print("Confusion Matrix for validation data is: ")
#print(confusion_matrix(Y_valid, predicted))

predicted.clear()

for i in range(len(Y_test)):
predicted.append(predict_covdif(X_test[i],means,covdif,counts))

print("accuracy on test set with different covariance matrix(Best Model) is "+str(accurac


confuse=confusion_matrix(Y_test,predicted)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for Guassian Distribution with different covariance matrix on
plt.savefig('Confusion_test_2.png')

plt.show()

predicted.clear()

for i in range(len(Y_train)):
predicted.append(predict_covdif(X_train[i],means,covdif,counts))

confuse=confusion_matrix(Y_train,predicted)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for Guassian Distribution with different covariance matrix on
plt.savefig('Confusion_train_2.png')

plt.show()


x1=np.linspace(-15,15,num=400)
x2=np.linspace(-3,15,num=400)
xx1, xx2 = np.meshgrid(x1, x2)
r1, r2 = xx1.flatten(), xx2.flatten()
r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
grid = np.hstack((r1,r2))
#print(grid)
predicted.clear()
num_cores = multiprocessing.cpu_count()

predicted = Parallel(n_jobs=num_cores)(delayed(predict_covdif)(grid[i],means,covdif,count
```

```
pos=np.empty(xx1.shape+(2,))
pos[:,:,0]=xx1
pos[:,:,1]=xx2


predicted=np.array(predicted)
predicted=predicted.reshape(xx1.shape)
fig = plt.figure(figsize=(8,8))
plt.contourf(xx1, xx2, predicted, cmap='RdBu')
colors = ['green','red','blue','purple']
plt.scatter(X_train[:,0], X_train[:,1], c=Y_train, cmap=matplotlib.colors.ListedColormap(
#plt.scatter(X_train[:,0], X_train[:,1], c=Y_train, cmap='RdBu')
for i in range(covdif.shape[0]):
    mid=multivariate_normal(mean=means[i],cov=covdif[i])
    plt.contour(xx1,xx2,mid.pdf(pos),[0.0075,0.05,0.1,0.15,0.2,0.23])

plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Decision Region plot with different covariance matrix for each class')

plt.savefig('plot_Gaussian_2.png')
plt.show()
```

1.(b) Nonlinearly separable data set for static pattern classification

1.(b).1 Code for K-nearest neighbours classifier, for K=1, K=7, K=15

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from statistics import mode
from sklearn.metrics import accuracy_score
from joblib import Parallel, delayed
import multiprocessing
import matplotlib
import matplotlib.patches as mpatches
import seaborn as sn
from sklearn.metrics import confusion_matrix

plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['font.size'] = 15
plt.rcParams["figure.figsize"] = (8,8)


def KNN(X_train,Y_train,k,X):
    predicted=[]
    for p in range(X.shape[0]):
        test=X[p]

        le=np.sum((test-X_train)**2,axis=1)
        distances=list()

        for i in range(len(Y_train)):
            distances.append((Y_train[i],le[i]))

        distances.sort(key=lambda tup: tup[1])
        neighbors = list()
        for i in range(k):
            neighbors.append(distances[i][0])
        predicted.append(mode(neighbors))


    return predicted


def KNN_single(X_train,Y_train,k,X):
    predicted=[]
    for p in range(1):
        test=X

        le=np.sum((test-X_train)**2,axis=1)
        distances=list()
```

```
        for i in range(len(Y_train)):
            distances.append((Y_train[i],le[i]))

        distances.sort(key=lambda tup: tup[1])
        neighbors = list()
        for i in range(k):
            neighbors.append(distances[i][0])
        predicted.append(mode(neighbors))

    return predicted[0]



#Taking input from csv file and taking x and y out
data=pd.read_csv("19/train.csv",header=None)

data=data.to_numpy()


X_train=data[:,0:2]
Y_train=data[:,2]


data=pd.read_csv("19/dev.csv",header=None)

X_valid=data.loc[np.r_[0:15, 30:45, 60:75],:]

X_valid=X_valid.to_numpy()


Y_valid=X_valid[:,-1]
X_valid=X_valid[:,0:2]

X_test=data.loc[np.r_[15:30, 45:60, 75:90],:]

X_test=X_test.to_numpy()


Y_test=X_test[:,-1]
X_test=X_test[:,0:2]



K=[1,7,15]

#KNN classifier


for k in K:
    predicted=KNN(X_train,Y_train,k,X_valid)
```

```
    print("accuracy for k="+str(k)+" on validation set is "+str(accuracy_score(Y_valid,pr

for k in K:
    predicted=KNN(X_train,Y_train,k,X_train)
    print("accuracy for k="+str(k)+" on training set is "+str(accuracy_score(Y_train,pred

k=7
predicted=[]
predicted=KNN(X_train,Y_train,k,X_train)
print("accuracy for k="+str(7)+" (Best Model) on testing set is "+str(accuracy_score(Y_te

confuse=confusion_matrix(Y_train,predicted)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for KNN with k=7 on Training data')
plt.savefig('Confusion_train_1.png')

plt.show()

predicted.clear()

predicted=KNN(X_train,Y_train,k,X_test)
confuse=confusion_matrix(Y_test,predicted)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for KNN with k=7 on Testing data')
plt.savefig('Confusion_test_1.png')

plt.show()


x1=np.linspace(-4,4,num=200)
x2=np.linspace(-3,3,num=200)
xx1, xx2 = np.meshgrid(x1, x2)
r1, r2 = xx1.flatten(), xx2.flatten()
r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
grid = np.hstack((r1,r2))
#print(grid)
predicted.clear()


# In[5]:


num_cores = multiprocessing.cpu_count()
```

```
#print(X_valid[0].shape[1])
predicted = Parallel(n_jobs=num_cores)(delayed(KNN_single)(X_train,Y_train,7,grid[i]) for
#predicted = Parallel(n_jobs=num_cores)(delayed(knn)(grid[i],means,covdif,counts) for i i
#predicted=KNN(X_train,Y_train,15,grid)
#print(predicted)
predicted=np.array(predicted)


predicted=predicted.reshape(xx1.shape)
fig = plt.figure(figsize=(8,8))
plt.contourf(xx1, xx2, predicted, cmap='RdBu')
colors = ['green','red','blue','purple']
plt.scatter(X_train[:,0], X_train[:,1], c=Y_train, cmap=matplotlib.colors.ListedColormap(
#plt.scatter(X_train[:,0], X_train[:,1], c=Y_train, cmap='RdBu')

plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.title('Decision Region plot for k=7 for each class',fontsize=20)

"""recs = []
for i in range(0,len(colors)):
    recs.append(mpatches.Rectangle((0,0),1,1,fc=colors[i]))
plt.legend(recs,unique,loc=4)
"""

plt.savefig('plot_KNN_1.png')
plt.show()
```

1.(b).2 Bayes Classifier with a GMM for each class , using full and diagonal covariance matrices

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn.metrics import confusion_matrix
import matplotlib
import matplotlib.patches as mpatches
from joblib import Parallel, delayed
import multiprocessing
from scipy.stats import multivariate_normal
import seaborn as sn
from matplotlib.collections import QuadMesh
import matplotlib.font_manager as fm

plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['font.size'] = 15
plt.rcParams["figure.figsize"] = (8,8)



def gauss(X, mean_vector, covariance_matrix):
    if (np.abs(np.linalg.det(covariance_matrix))==0):
        print("ERROR")
    # a= (2*np.pi)**(-len(X)/2)*np.abs(np.prod((np.linalg.eigvals(covariance_matrix))))**
    b= (2*np.pi)**(-len(X)/2)*(np.linalg.det(covariance_matrix))**(-1/2)*np.exp(-np.dot(n
    # c= ((1/(((2*math.pi)**(X.shape[0]/2))*((np.linalg.det(covariance_matrix))**0.5)))*m
    # return (2*np.pi)**(-len(X)/2)*np.linalg.det(covariance_matrix)**(-1/2)*np.exp(-np.d

    return b

def KNN_class(data,k):
    #k=4 # number of clusters
    #print(data.shape[0])
    #np.random.seed(0)
    means = data[np.random.choice(range(data.shape[0]), k, replace=False),:]
    #print(means[0])

    z_prev=np.zeros([data.shape[0],k])

    convergence=True
    count=0
    while(convergence):
     tp=np.zeros([data.shape[0],k])
     for i in range(data.shape[0]):
     list=np.empty([k,1])
     for p in range(k):
     list[p]=(np.sum((data[i,:]-means[p])**2))
```

```
      tp[i][np.argmin(list,axis=0)]=1
      #print(tp)
      #print(means)

      for i in range(k):
      b=np.where(tp[:,i]==1)
      #print(np.sum(np.sum(data[b,:],axis=0),axis=0).shape)
      #print(len(b[0]))
      #print(data[b,:].shape)
      #print(means[i])
      means[i]=np.sum(np.sum(data[b,:],axis=0),axis=0)
      #print(means[i])
      #print(means[i]/len(b[0]))
      means[i]=means[i]/len(b[0])

      comparison= tp==z_prev
      if comparison.all():
      break
      else:
      count+=1
      z_prev=tp.copy()
    return means,z_prev

def GMM_classifier(X,means,weights,cov,k):
    ll_n=[]
    for i in range(3):
#            ll= np.log(sum([weights[i][j]*gauss(X_valid[n], means[i][j], cov[i][j])  fo
        ll= np.log(sum([weights[i][j]*gauss(X, means[i][j], cov[i][j])  for j in range(k)
        ll_n.append(ll)
    ll_n=np.array(ll_n)
#         print(ll_n)

    return np.argmax(ll_n)



K=[2,3,5,10,4]

size_best=[]
weights_best=[]
cov_best=[]
means_best=[]


for k in K:

    data=pd.read_csv("19/train.csv",header=None)

    data.columns =['x1', 'x2','Class']

    data1= data[data['Class']==0]
```

```
data2= data[data['Class']==1]
data3= data[data['Class']==2]

X_data=[data1, data2, data3]



size=[]
weights=[]
cov=[]
means=[]


# The only hyperparameter is k ( no.of components for each class)


for c, X in enumerate (X_data):
    X= X.to_numpy()
    X= X[:,:-1]
#     print(X)
    size.append(len(X))
    print(f"\nClass {c}\n")
#     print(size)

    means_old,r_old=KNN_class(X,k)

    N=len(X)

    Nq_old=np.sum(r_old,axis=0) # sum conatins the number of elements belonging
                                 # to each cluster

    # Initialization

    #cov2 is a 3-d array containing the covariance matrix of each cluster
    cov_old=np.zeros([k,X.shape[1],X.shape[1]])
    Wq_old =np.zeros([k,1]) ## weight of each cluster

    for i in range(k):
        Nq=Nq_old[i]
        Wq_old[i]= Nq/N
        tp=np.zeros([X.shape[1],X.shape[1]])

        for p in range(X.shape[0]):
            le=X[p,:]-means_old[i]
            le=np.reshape(le,[le.shape[0],1])
            tp=tp+r_old[p,i]*(np.dot(le,le.T))
        tp=tp/Nq

#         d= np.diag(tp)
#         tp=np.diag(d)
        cov_old[i,:,:]=tp.copy()
```

```
ll_old= 0.0
for n in range(len(X)):
    ll_old = ll_old + np.log(sum([Wq_old[j]*gauss(X[n], means_old[j], cov_old[j])

#print(ll_old)

convergence=False
iter_convergence=0
run=0
runs=1000
epsilon=100

while (convergence == False and run<runs):

    # ''' --------------------------   E - STEP   -------------------------- '''

    # Initiating the r matrix, every row contains the probabilities
    # for every cluster for this row

    r_new = np.zeros((len(X), k))  # responsibilty matrix

    # Calculating the r matrix
    for n in range(len(X)):
        for i in range(k):
            r_new[n][i] = Wq_old[i] * gauss(X[n], means_old[i], cov_old[i])
            r_new[n][i] /= sum([Wq_old[j]*gauss(X[n], means_old[j], cov_old[j]) f

    # Calculating the N effective elemts fro each component
    Nq_new = np.sum(r_new, axis=0)


    # ''' --------------------------   M - STEP   -------------------------- '''


    # Updating the weights list
    Wq_new =np.zeros([k,1]) ## weight of each cluster
    for i in range(k):
        Wq_new[i]= Nq_new[i]/ N


    # Initializing the mean vector as a zero vector
    means_new = np.zeros((k, len(X[0])))

    # Updating the mean vector
    for i in range(k):
        for n in range(len(X)):
            means_new[i] = means_new[i] + r_new[n][i] * X[n]
        means_new[i] = means_new [i]/Nq_new[i]
```

```
    # Initiating the list of the covariance matrixes
    cov_new =np.zeros([k,X.shape[1],X.shape[1]])

    # Updating the covariance matrices
    for i in range(k):
        Nq=Nq_new[i]
        tp=np.zeros([X.shape[1],X.shape[1]])

        for p in range(X.shape[0]):
            le=X[p,:]-means_new[i]
            le=np.reshape(le,[le.shape[0],1])
            tp=tp+r_new[p,i]*(np.dot(le,le.T))

        tp=tp/Nq
        #d= np.diag(tp)
        #tp=np.diag(d)
        cov_new[i,:,:]=tp.copy()




    # Calculating log-likelhood
    ll_new=0
    for n in range(len(X)):
        ll_new = ll_new + np.log(sum([Wq_new[j]*gauss(X[n], means_new[j], cov_new

#     print(ll_new)
    diff=ll_new-ll_old

    #print(diff)

    #Convergence condition
    if diff < 1e-2:
        iter_convergence=run
        convergence=True
        break

    else:
        ll_old=ll_new.copy()
        Wq_old= Wq_new.copy()
        means_old=means_new.copy()
        cov_old=cov_new.copy()

    run= run +1

if convergence==True and run!=runs:
    print("Iterations for convergence=",iter_convergence)
else:
    print("Estimate has not converged yet, more runs needed")
```

```
    #print(ll_new)


    weights.append(Wq_new)
    means.append(means_new)
    cov.append(cov_new)

print("#############################################################################


data=pd.read_csv("19/train.csv",header=None)


data.columns =['x1', 'x2','Class']



data1= data[data['Class']==0]
data2= data[data['Class']==1]
data3= data[data['Class']==2]



X_data=[data1, data2, data3]

prob=0
tot=len(data)
predicted=[]
real=[]
for ind, X_valid in enumerate(X_data):

    X_valid= X_valid.to_numpy()
    X_valid= X_valid[:,:-1]



    index=[]

    for n in range(len(X_valid)):
        ll_n=[]
        for i in range(3):
#             ll= np.log(sum([weights[i][j]*gauss(X_valid[n], means[i][j], cov[i][j])
            ll= np.log(sum([weights[i][j]*gauss(X_valid[n], means[i][j], cov[i][j])
            ll_n.append(ll)
        ll_n=np.array(ll_n)
#         print(ll_n)
        index.append(np.argmax(ll_n))
        predicted.append(np.argmax(ll_n))
        real.append(ind)
#     print(len(index))
#     print(index)
```

```
    p=index.count(ind)
    prob+=p
    #print(prob)

#print(X_valid)
print("accuracy for k="+str(k)+" using GMM with full covariance matrix on Training se

if k==4:
    confuse=confusion_matrix(real,predicted)

    sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
        fmt='.2%', cmap='Blues',cbar=False)
    plt.xlabel('Predicted Class')
    plt.ylabel("Actual Class")
    plt.title('Confusion Matrix for GMM with k=4 on Training data')
    plt.savefig('Confusion_train_2.png')

    plt.show()




data=pd.read_csv("19/dev.csv",header=None)
data.columns =['x1', 'x2','Class']
#print(len(data))

X_valid=data.loc[np.r_[0:15, 30:45, 60:75],:]

data1= X_valid[X_valid['Class']==0]
data2= X_valid[X_valid['Class']==1]
data3= X_valid[X_valid['Class']==2]

X_data=[data1, data2, data3]
prob=0
tot=len(X_valid)
for ind, X_valid in enumerate(X_data):

    X_valid= X_valid.to_numpy()
    X_valid= X_valid[:,:-1]



    index=[]

    for n in range(len(X_valid)):
        ll_n=[]
        for i in range(3):
#             ll= np.log(sum([weights[i][j]*gauss(X_valid[n], means[i][j], cov[i][j])
            ll= np.log(sum([weights[i][j]*gauss(X_valid[n], means[i][j], cov[i][j])
            ll_n.append(ll)
        ll_n=np.array(ll_n)
```

```
    #        print(ll_n)
            index.append(np.argmax(ll_n))
    #     print(len(index))
    #     print(index)
        p=index.count(ind)
        prob+=p
        #print(prob)

    #print(X_valid)
    print("accuracy for k="+str(k)+" using GMM with full covariance matrix on validation
    print("################################################################################
    size_best=size.copy()
    weights_best=weights.copy()
    cov_best=cov.copy()
    means_best=means.copy()



data=pd.read_csv("19/dev.csv",header=None)
data.columns =['x1', 'x2','Class']
#print(len(data))
k=4
X_valid=data.loc[np.r_[15:30, 45:60, 75:90],:]

data1= X_valid[X_valid['Class']==0]
data2= X_valid[X_valid['Class']==1]
data3= X_valid[X_valid['Class']==2]

X_data=[data1, data2, data3]
prob=0
tot=len(X_valid)
predicted=[]
real=[]
for ind, X_valid in enumerate(X_data):

    X_valid= X_valid.to_numpy()
    X_valid= X_valid[:,:-1]



    index=[]

    for n in range(len(X_valid)):
        ll_n=[]
        for i in range(3):
#            ll= np.log(sum([weights[i][j]*gauss(X_valid[n], means[i][j], cov[i][j])  fo
            ll= np.log(sum([weights[i][j]*gauss(X_valid[n], means_best[i][j], cov_best[i]
            #print(X_valid[n])
            ll_n.append(ll)
        ll_n=np.array(ll_n)
        index.append(np.argmax(ll_n))
```

```
        predicted.append(np.argmax(ll_n))
        real.append(ind)

    p=index.count(ind)
    prob+=p
#     print(len(index))
    #print(prob)


print("accuracy for k="+str(4)+" using GMM with full covariance matrix on Testing set is
confuse=confusion_matrix(real,predicted)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for GMM with k=4 on Testing data')
plt.savefig('Confusion_test_2.png')

plt.show()




x1=np.linspace(-4,4,num=200)
x2=np.linspace(-3,3,num=200)
xx1, xx2 = np.meshgrid(x1, x2)
r1, r2 = xx1.flatten(), xx2.flatten()
r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
grid = np.hstack((r1,r2))
#print(grid)
predicted.clear()
num_cores = multiprocessing.cpu_count()
#GMM_classifier(X,means,cov,k)
predicted = Parallel(n_jobs=num_cores)(delayed(GMM_classifier)(grid[i],means_best,weights
pos=np.empty(xx1.shape+(2,))
pos[:,:,0]=xx1
pos[:,:,1]=xx2


predicted=np.array(predicted)
predicted=predicted.reshape(xx1.shape)
fig = plt.figure(figsize=(8,8))
plt.contourf(xx1, xx2, predicted, cmap='RdBu')
colors = ['green','red','blue','purple']

data=pd.read_csv("19/train.csv",header=None)
X_train= data.to_numpy()
#data.columns =['x1', 'x2','Class']


plt.scatter(X_train[:,0], X_train[:,1], c=X_train[:,2], cmap=matplotlib.colors.ListedColo
```

```
#plt.scatter(X_train[:,0], X_train[:,1], c=Y_train, cmap='RdBu')

for i in range(3):
    for j in range(k):
        mid=multivariate_normal(mean=means_best[i][j],cov=cov_best[i][j])
        plt.contour(xx1,xx2,mid.pdf(pos),[0.3,0.5,0.7,0.8,0.85,0.9,0.95,1,1.1,1.15])

plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Decision Region plot for GMM with k=4 for each class using full covariance mat

plt.savefig('plot_GMM_Full_2.png')
plt.show()
```

1.(b).3 Code for Bayes Classifier with K-nearest neighbours method for estimation of class-conditional probability density function, for K=10 and K=20

```
    import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from statistics import mode
from sklearn.metrics import accuracy_score
from joblib import Parallel, delayed
import multiprocessing
import matplotlib
import matplotlib.patches as mpatches
from collections import defaultdict
from sklearn.metrics import confusion_matrix
import seaborn as sn

plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['font.size'] = 15
plt.rcParams["figure.figsize"] = (8,8)

def KNN(X_train,Y_train,k,X):
    predicted=[]
    for p in range(X.shape[0]):
        test=X[p]

        le=np.sum((test-X_train)**2,axis=1)
        #distances=list()
        distances=defaultdict(list)
        for i in range(len(Y_train)):
            distances[Y_train[i]].append(le[i])

        #distances.sort(key=lambda tup: tup[0])
        neighbors = list()
        for l, v in distances.items():  #.iteritems for lower python versions
            distances[l].sort()
            neighbors.append(distances[l][k-1])
        #if p==0:
        #    print(distances)
        predicted.append(np.argmin(neighbors,axis=0))


    return predicted


def KNN_single(X_train,Y_train,k,X):
    predicted=[]
    for p in range(1):
        test=X

        le=np.sum((test-X_train)**2,axis=1)
```

```
        #distances=list()
        distances=defaultdict(list)
        for i in range(len(Y_train)):
            distances[Y_train[i]].append(le[i])

        #distances.sort(key=lambda tup: tup[0])
        neighbors = list()
        for l, v in distances.items():  #.iteritems for lower python versions
            distances[l].sort()
            neighbors.append(distances[l][k-1])
        #if p==0:
        #    print(distances)
        predicted.append(np.argmin(neighbors,axis=0))


    return predicted[0]



#Taking input from csv file and taking x and y out
data=pd.read_csv("19/train.csv",header=None)

data=data.to_numpy()


X_train=data[:,0:2]
Y_train=data[:,2]


data=pd.read_csv("19/dev.csv",header=None)
X_test=data.loc[np.r_[15:30, 45:60, 75:90],:]
X_test=X_test.to_numpy()
Y_test=X_test[:,2]
X_test=X_test[:,0:2]
#X_valid=data[0:int(data.shape[0]/2),0:2]
#Y_valid=data[0:int(data.shape[0]/2),2]

X_valid=data.loc[np.r_[0:15, 30:45, 60:75],:]
X_valid=X_valid.to_numpy()
Y_valid=X_valid[:,2]
X_valid=X_valid[:,0:2]


K=[10,20]

#KNN classifier
for k in K:
    predicted=KNN(X_train,Y_train,k,X_valid)
    print("accuracy for k="+str(k)+" on validation set is "+str(accuracy_score(Y_valid,pr

for k in K:
```

```
    predicted=KNN(X_train,Y_train,k,X_train)
    print("accuracy for k="+str(k)+" on training set is "+str(accuracy_score(Y_train,pred

k=10

predicted=KNN(X_train,Y_train,k,X_test)
print("accuracy for k="+str(k)+" (best model) on testing set is "+str(accuracy_score(Y_te


confuse=confusion_matrix(Y_test,predicted)
#print(confuse)
sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for Bayes KNN with k=10 on Testing data')
plt.savefig('Confusion_test_Bayes_KNN_4.png')

plt.show()

predicted=KNN(X_train,Y_train,k,X_train)

confuse=confusion_matrix(Y_train,predicted)
#print(confuse)
sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False)
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for Bayes KNN with k=10 on Training data')
plt.savefig('Confusion_train_Bayes_KNN_4.png')

plt.show()


x1=np.linspace(-4,4,num=200)
x2=np.linspace(-3,3,num=200)
xx1, xx2 = np.meshgrid(x1, x2)
r1, r2 = xx1.flatten(), xx2.flatten()
r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))
grid = np.hstack((r1,r2))
#print(grid)
predicted.clear()

num_cores = multiprocessing.cpu_count()
#print(X_valid[0].shape[1])
predicted = Parallel(n_jobs=num_cores)(delayed(KNN_single)(X_train,Y_train,10,grid[i]) fo
#predicted = Parallel(n_jobs=num_cores)(delayed(knn)(grid[i],means,covdif,counts) for i i
#predicted=KNN(X_train,Y_train,15,grid)
#print(predicted)
predicted=np.array(predicted)
```

```
predicted=predicted.reshape(xx1.shape)
fig = plt.figure(figsize=(8,8))
plt.contourf(xx1, xx2, predicted, cmap='RdBu')
colors = ['green','red','blue','purple']
plt.scatter(X_train[:,0], X_train[:,1], c=Y_train, cmap=matplotlib.colors.ListedColormap(
#plt.scatter(X_train[:,0], X_train[:,1], c=Y_train, cmap='RdBu')

plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.title('Decision Region plot for k=10 for each class',fontsize=20)


plt.savefig('plot_KNN_Bayes_4.png')
plt.show()
```

Dataset 2(A) Code for Bayes Classifier with a GMM for each class using full covariance matrices

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn.cluster import KMeans
from scipy.stats import multivariate_normal

plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['font.size'] = 15
plt.rcParams["figure.figsize"] = (9,9)



size=[]
weights=[]
cov=[]
means=[]

def gauss(X, mean_vector, covariance_matrix):
    if (np.abs(np.linalg.det(covariance_matrix))==0):
        print("ERROR")
     # a= (2*np.pi)**(-len(X)/2)*np.abs(np.prod((np.linalg.eigvals(covariance_matrix
    b= (2*np.pi)**(-len(X)/2)*(np.linalg.det(covariance_matrix))**(-1/2)*np.exp(-np.
     # c= ((1/(((2*math.pi)**(X.shape[0]/2))*((np.linalg.det(covariance_matrix))**0.
     # return (2*np.pi)**(-len(X)/2)*np.linalg.det(covariance_matrix)**(-1/2)*np.exp

    return b

# The only hyperparameter is k ( no.of components for each class)
k=3
train=['coast','forest','opencountry','street','tallbuilding']

for c, train_file in enumerate(train):
    data=pd.read_csv('dataset/'+train_file+'/train.csv')
    data=data.to_numpy()
    X=data[:,1:]
    size.append(len(X))
    print(f"\n\n\nClass {c}")
#     print(size)
    kmeans=KMeans(n_clusters=k,random_state=0).fit(X)
    # kmeans=KMeans(n_clusters=k).fit(X)
    means_old=kmeans.cluster_centers_
    labels=kmeans.labels_


    N=len(X)
    r_old=np.zeros((len(X),k)) # form a Z ( indicator ) matrix

    for i in range(len(X)):
        r_old[i,labels[i]]=1

    Nq_old=np.sum(r_old,axis=0) # sum conatins the number of elements belonging
                                 # to each cluster

    print("\nOriginal effective number of elements in each cluster")
    print(Nq_old)
    # Initialization
```

```
    #cov2 is a 3-d array containing the covariance matrix of each cluster
    cov_old=np.zeros([k,X.shape[1],X.shape[1]])
    Wq_old =np.zeros([k,1]) ## weight of each cluster

    for i in range(k):
        Nq=Nq_old[i]
        Wq_old[i]= Nq/N
        tp=np.zeros([X.shape[1],X.shape[1]])

        for p in range(X.shape[0]):
            le=X[p,:]-means_old[i]
            le=np.reshape(le,[le.shape[0],1])
            tp=tp+r_old[p,i]*(np.dot(le,le.T))
        tp=tp/Nq

#          d= np.diag(tp)
#          tp=np.diag(d)
        cov_old[i,:,:]=tp.copy()

    ll_old= 0.0
    for n in range(len(X)):
        ll_old = ll_old + np.log(sum([Wq_old[j]*multivariate_normal.pdf(X[n], means_

    print(f"\nInitial log-likehood = {ll_old}")

    convergence=False
    iter_convergence=0
    run=0
    runs=1000
    epsilon=100

    while (convergence == False and run<runs):

        # '''  --------------------------   E - STEP   -------------------------- '''

        # Initiating the r matrix, every row contains the probabilities
        # for every cluster for this row

        r_new = np.zeros((len(X), k))  # responsibilty matrix

        # Calculating the r matrix
        for n in range(len(X)):
            for i in range(k):
                r_new[n][i] = Wq_old[i] * multivariate_normal.pdf(X[n], means_old[i]
                r_new[n][i] /= sum([Wq_old[j]*multivariate_normal.pdf(X[n], means_ol

        # Calculating the N effective elemts fro each component
        Nq_new = np.sum(r_new, axis=0)


        # '''  --------------------------   M - STEP   -------------------------- '''


        # Updating the weights list
        Wq_new =np.zeros([k,1]) ## weight of each cluster
        for i in range(k):
            Wq_new[i]= Nq_new[i]/ N


        # Initializing the mean vector as a zero vector
        means_new = np.zeros((k, len(X[0])))

        # Updating the mean vector
        for i in range(k):
            for n in range(len(X)):
```

```
                means_new[i] = means_new[i] + r_new[n][i] * X[n]
            means_new[i] = means_new [i]/Nq_new[i]



        # Initiating the list of the covariance matrixes
        cov_new =np.zeros([k,X.shape[1],X.shape[1]])

        # Updating the covariance matrices
        for i in range(k):
            Nq=Nq_new[i]
            tp=np.zeros([X.shape[1],X.shape[1]])

            for p in range(X.shape[0]):
                le=X[p,:]-means_new[i]
                le=np.reshape(le,[le.shape[0],1])
                tp=tp+r_new[p,i]*(np.dot(le,le.T))

            tp=tp/Nq
#             d= np.diag(tp)
#             tp=np.diag(d)
            cov_new[i,:,:]=tp.copy()


        # print(f"\nRun= {run}\n")
#         print(np.sum(Nq_new))
#         print("\nWeights\n")
#         print(np.sum(Wq_new))
#         print(Wq_new)
#         print(np.sum(r_new))
#         print("\n------------------")

        # Calculating log-likelhood
        ll_new=0
        for n in range(len(X)):
            ll_new = ll_new + np.log(sum([Wq_new[j]*multivariate_normal.pdf(X[n], me

    #     print(ll_new)
        diff=ll_new-ll_old

    #     print(diff)

        #Convergence condition
        if diff < 1e-3:
            iter_convergence=run
            convergence=True
            break

        else:
            ll_old=ll_new.copy()
            Wq_old= Wq_new.copy()
            means_old=means_new.copy()
            cov_old=cov_new.copy()

        run= run +1

    if convergence==True and run!=runs:
        print("Iterations for convergence=",iter_convergence)
    else:
        print("Estimate has not converged yet, more runs needed")
    print(f"Final log-likehood = {ll_new}")

    print("\nEffective number of elements in each cluster is")
    print(Nq_new)
#     ass=np.sum(Nq_new)
```

```
#     print(ass)
    weights.append(Wq_new)
    means.append(means_new)
    cov.append(cov_new)

print("\n###########################################################################
```

```
Class 0

Original effective number of elements in each cluster
[ 88.  56. 107.]

Initial log-likehood = [6292.33397387]
Iterations for convergence= 14
Final log-likehood = [6473.1081942]

Effective number of elements in each cluster is
[91.78637783 63.19850117 96.01512101]



Class 1

Original effective number of elements in each cluster
[94. 60. 75.]

Initial log-likehood = [9178.14317148]
Iterations for convergence= 56
Final log-likehood = [9330.70296603]

Effective number of elements in each cluster is
[64.05643279 95.8423791  69.10118811]



Class 2

Original effective number of elements in each cluster
[93. 97. 97.]

Initial log-likehood = [7356.51573774]
Iterations for convergence= 21
Final log-likehood = [7529.79526103]

Effective number of elements in each cluster is
[ 84.86538213 104.37967119  97.75494668]



Class 3

Original effective number of elements in each cluster
[75. 86. 43.]

Initial log-likehood = [7853.03451778]
Iterations for convergence= 6
Final log-likehood = [7904.37594683]

Effective number of elements in each cluster is
[72.10947993 89.88461139 42.00590868]



Class 4

Original effective number of elements in each cluster
[59. 98. 92.]
```

```
Initial log-likehood = [7434.4176154]
Iterations for convergence= 29
Final log-likehood = [7738.19258261]

Effective number of elements in each cluster is
[80.08498994 71.1641749  97.75083516]

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

In [2]:

```python
size=np.array(size)
prior_class=size/np.sum(size)

validation_set=['coast','forest','opencountry','street','tallbuilding']
# validation_set=['train_1.csv','train_2.csv','train_3.csv','train_4.csv','train_5.c


valid_data=pd.DataFrame()
test_data=pd.DataFrame()
train_data=pd.DataFrame()
for ind, valid_file in enumerate(validation_set):

    X_valid=pd.read_csv('dataset/'+valid_file+'/dev.csv')
    X_valid['y']=ind
    msk = np.random.rand(len(X_valid)) < 0.5  #50-50 splits
    #print(X_valid[msk])
    valid_data=pd.concat([valid_data,X_valid[msk]],ignore_index=True)
    test_data=pd.concat([test_data,X_valid[~msk]],ignore_index=True)

for ind, valid_file in enumerate(validation_set):

    X_valid=pd.read_csv('dataset/'+valid_file+'/train.csv')
    X_valid['y']=ind

    train_data=pd.concat([train_data,X_valid],ignore_index=True)
```

In [3]:

```python
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import seaborn as sn
predicted=[]
real=[]

for i in range(len(valid_data)):

    X_valid=(valid_data.loc[i,:]).to_numpy()
    #print(X_valid)
    X_valid=X_valid[1:]
    Y_valid=X_valid[-1]
    X_valid=X_valid[:-1]
    real.append(Y_valid)


    #for n in range(len(X_valid)):
    ll_n=[]
    for i in range(len(validation_set)):
        ll= np.log(sum([weights[i][j]*multivariate_normal.pdf(X_valid, means[i][j],
        ll_n.append(ll)
    ll_n=np.array(ll_n)
    predicted.append(np.argmax(ll_n))

    #p=index.count(ind)
```

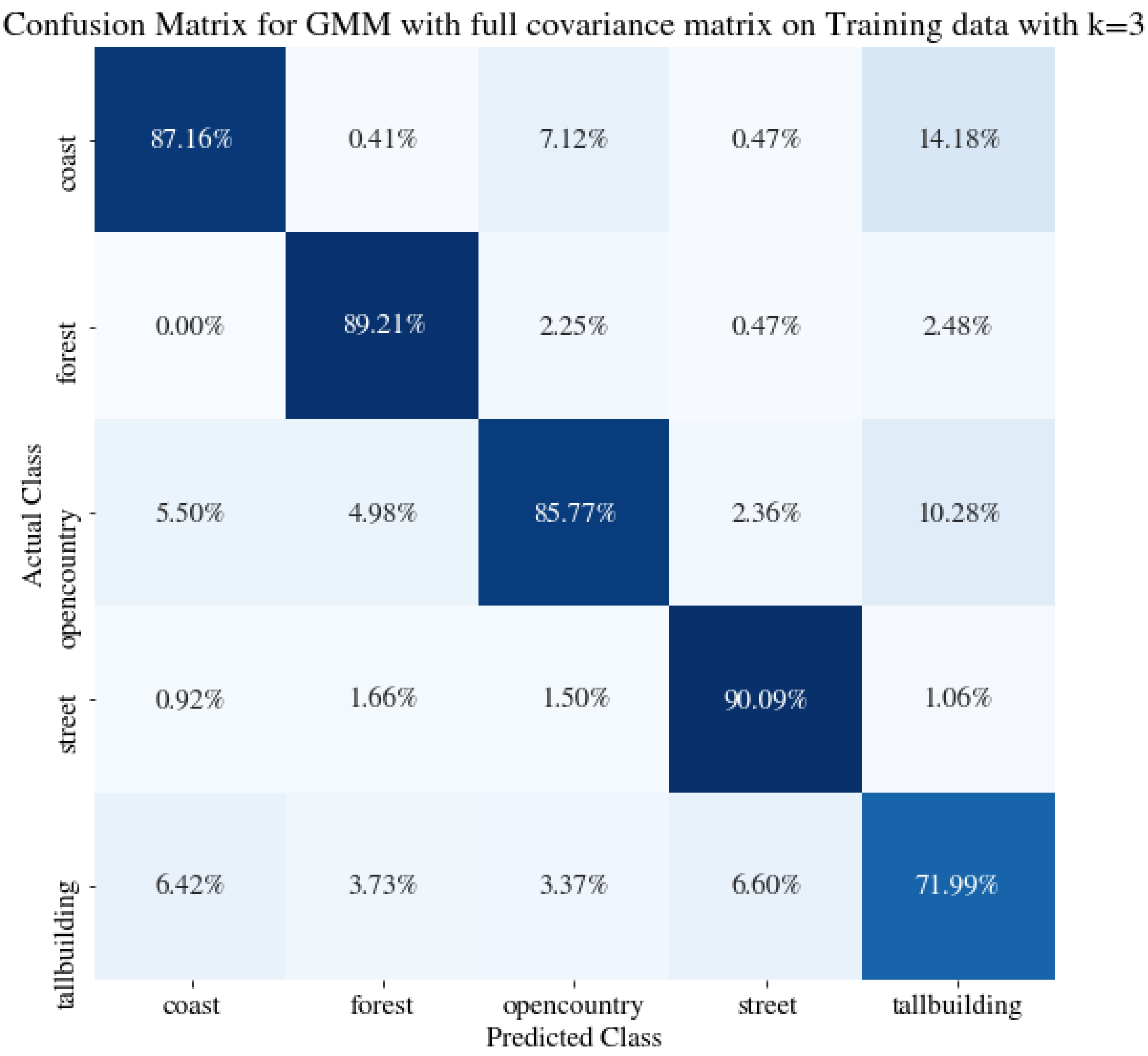
```
    #prob=p/len(index)

    #print(prob)

print("accuracy on validation set using full covariance matrix and k="+str(k)+ " is
```

```
<ipython-input-3-0577f94f3200>:20: RuntimeWarning: divide by zero encountered in log
  ll= np.log(sum([weights[i][j]*multivariate_normal.pdf(X_valid, means[i][j], cov[i]
[j],allow_singular=True)  for j in range(k)])) + np.log(prior_class[i])
accuracy on validation set using full covariance matrix and k=3 is 60.54054054054055
```

In [4]:

```
predicted=[]
real=[]
#k=3
#print(len(validation_set))

for i in range(len(train_data)):

    X_valid=(train_data.loc[i,:]).to_numpy()
    #print(X_valid)
    X_valid=X_valid[1:]
    Y_valid=X_valid[-1]
    X_valid=X_valid[:-1]
    real.append(Y_valid)


    #for n in range(len(X_valid)):
    ll_n=[]
    for i in range(len(validation_set)):
        ll= np.log(sum([weights[i][j]*multivariate_normal.pdf(X_valid, means[i][j],
        ll_n.append(ll)
    ll_n=np.array(ll_n)
    predicted.append(np.argmax(ll_n))

    #p=index.count(ind)
    #prob=p/len(index)

    #print(prob)

print("accuracy on Training set using full covariance matrix and k="+str(k)+ " is "

if k==3:
    confuse=confusion_matrix(real,predicted)

    sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
        fmt='.2%', cmap='Blues',cbar=False,xticklabels=validation_set,yticklabels=va
    plt.xlabel('Predicted Class')
    plt.ylabel("Actual Class")
    plt.title('Confusion Matrix for GMM with full covariance matrix on Training data
    plt.savefig('Confusion_train_1.png')
    #plt.xaxis.set_ticklabels(validation_set);
    #ax.yaxis.set_ticklabels(validation_set[::-1]);
    plt.show()
```

```
<ipython-input-4-3e05713b4d2d>:19: RuntimeWarning: divide by zero encountered in log
  ll= np.log(sum([weights[i][j]*multivariate_normal.pdf(X_valid, means[i][j], cov[i]
[j],allow_singular=True)  for j in range(k)])) + np.log(prior_class[i])
accuracy on Training set using full covariance matrix and k=3 is 84.26229508196721
```

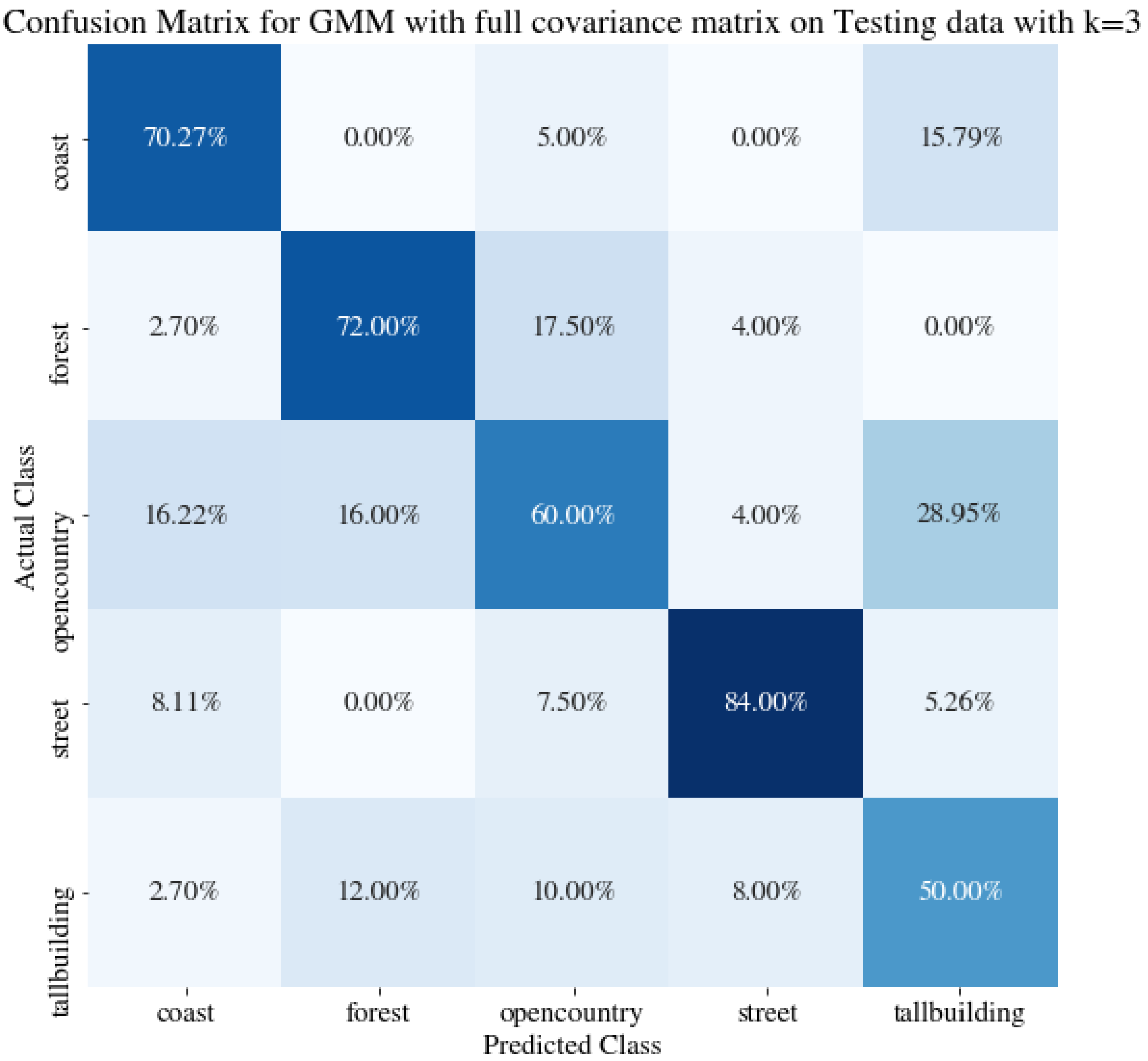


In [5]:

```
predicted=[]
real=[]

for i in range(len(test_data)):

    X_valid=(test_data.loc[i,:]).to_numpy()
    #print(X_valid)
    X_valid=X_valid[1:]
    Y_valid=X_valid[-1]
    X_valid=X_valid[:-1]
    real.append(Y_valid)


    #for n in range(len(X_valid)):
    ll_n=[]
    for i in range(len(validation_set)):
        ll= np.log(sum([weights[i][j]*multivariate_normal.pdf(X_valid, means[i][j],
        ll_n.append(ll)
    ll_n=np.array(ll_n)
    predicted.append(np.argmax(ll_n))

    #p=index.count(ind)
    #prob=p/len(index)

    #print(prob)

print("accuracy on Testing set using full covariance matrix and k="+str(k)+ " is " +
```

```
if k==3:

    confuse=confusion_matrix(real,predicted)

    sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
        fmt='.2%', cmap='Blues',cbar=False,xticklabels=validation_set,yticklabels=va
    plt.xlabel('Predicted Class')
    plt.ylabel("Actual Class")
    plt.title('Confusion Matrix for GMM with full covariance matrix on Testing data
    plt.savefig('Confusion_test_1.png')
    #plt.xaxis.set_ticklabels(validation_set);
    #ax.yaxis.set_ticklabels(validation_set[::-1]);
    plt.show()
```

```
accuracy on Testing set using full covariance matrix and k=3 is 65.45454545454545
```



In [ ]:

Dataset 2(A) Code for Bayes Classifier with a GMM for each class, using diagonal covariance matrices

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn.cluster import KMeans
from scipy.stats import multivariate_normal

plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['font.size'] = 15
plt.rcParams["figure.figsize"] = (9,9)



size=[]
weights=[]
cov=[]
means=[]

def gauss(X, mean_vector, covariance_matrix):
    if (np.abs(np.linalg.det(covariance_matrix))==0):
        print("ERROR")
     # a= (2*np.pi)**(-len(X)/2)*np.abs(np.prod((np.linalg.eigvals(covariance_matrix
    b= (2*np.pi)**(-len(X)/2)*(np.linalg.det(covariance_matrix))**(-1/2)*np.exp(-np.
     # c= ((1/(((2*math.pi)**(X.shape[0]/2))*((np.linalg.det(covariance_matrix))**0.
     # return (2*np.pi)**(-len(X)/2)*np.linalg.det(covariance_matrix)**(-1/2)*np.exp

    return b

# The only hyperparameter is k ( no.of components for each class)
k=5
train=['coast','forest','opencountry','street','tallbuilding']

for c, train_file in enumerate(train):
    data=pd.read_csv('dataset/'+train_file+'/train.csv')
    data=data.to_numpy()
    X=data[:,1:]
    size.append(len(X))
    print(f"\n\n\nClass {c}")
#     print(size)
    kmeans=KMeans(n_clusters=k,random_state=0).fit(X)
    # kmeans=KMeans(n_clusters=k).fit(X)
    means_old=kmeans.cluster_centers_
    labels=kmeans.labels_


    N=len(X)
    r_old=np.zeros((len(X),k)) # form a Z ( indicator ) matrix

    for i in range(len(X)):
        r_old[i,labels[i]]=1

    Nq_old=np.sum(r_old,axis=0) # sum conatins the number of elements belonging
                                 # to each cluster

    print("\nOriginal effective number of elements in each cluster")
    print(Nq_old)
    # Initialization
```

```
#cov2 is a 3-d array containing the covariance matrix of each cluster
cov_old=np.zeros([k,X.shape[1],X.shape[1]])
Wq_old =np.zeros([k,1]) ## weight of each cluster

for i in range(k):
    Nq=Nq_old[i]
    Wq_old[i]= Nq/N
    tp=np.zeros([X.shape[1],X.shape[1]])

    for p in range(X.shape[0]):
        le=X[p,:]-means_old[i]
        le=np.reshape(le,[le.shape[0],1])
        tp=tp+r_old[p,i]*(np.dot(le,le.T))
    tp=tp/Nq

    d= np.diag(tp)
    tp=np.diag(d)
    cov_old[i,:,:]=tp.copy()

ll_old= 0.0
for n in range(len(X)):
    ll_old = ll_old + np.log(sum([Wq_old[j]*multivariate_normal.pdf(X[n], means_

print(f"\nInitial log-likehood = {ll_old}")

convergence=False
iter_convergence=0
run=0
runs=1000
epsilon=100

while (convergence == False and run<runs):

    # '''  --------------------------   E - STEP   -------------------------- '''

    # Initiating the r matrix, every row contains the probabilities
    # for every cluster for this row

    r_new = np.zeros((len(X), k))  # responsibilty matrix

    # Calculating the r matrix
    for n in range(len(X)):
        for i in range(k):
            r_new[n][i] = Wq_old[i] * multivariate_normal.pdf(X[n], means_old[i]
            r_new[n][i] /= sum([Wq_old[j]*multivariate_normal.pdf(X[n], means_ol

    # Calculating the N effective elemts fro each component
    Nq_new = np.sum(r_new, axis=0)


    # '''  --------------------------   M - STEP   -------------------------- '''


    # Updating the weights list
    Wq_new =np.zeros([k,1]) ## weight of each cluster
    for i in range(k):
        Wq_new[i]= Nq_new[i]/ N


    # Initializing the mean vector as a zero vector
    means_new = np.zeros((k, len(X[0])))

    # Updating the mean vector
    for i in range(k):
        for n in range(len(X)):
```

```
                means_new[i] = means_new[i] + r_new[n][i] * X[n]
            means_new[i] = means_new [i]/Nq_new[i]



        # Initiating the list of the covariance matrixes
        cov_new =np.zeros([k,X.shape[1],X.shape[1]])

        # Updating the covariance matrices
        for i in range(k):
            Nq=Nq_new[i]
            tp=np.zeros([X.shape[1],X.shape[1]])

            for p in range(X.shape[0]):
                le=X[p,:]-means_new[i]
                le=np.reshape(le,[le.shape[0],1])
                tp=tp+r_new[p,i]*(np.dot(le,le.T))

            tp=tp/Nq
            d= np.diag(tp)
            tp=np.diag(d)
            cov_new[i,:,:]=tp.copy()


        # print(f"\nRun= {run}\n")
#         print(np.sum(Nq_new))
#         print("\nWeights\n")
#         print(np.sum(Wq_new))
#         print(Wq_new)
#         print(np.sum(r_new))
#         print("\n------------------")

        # Calculating log-likelhood
        ll_new=0
        for n in range(len(X)):
            ll_new = ll_new + np.log(sum([Wq_new[j]*multivariate_normal.pdf(X[n], me

    #     print(ll_new)
        diff=ll_new-ll_old

    #     print(diff)

        #Convergence condition
        if diff < 1e-3:
            iter_convergence=run
            convergence=True
            break

        else:
            ll_old=ll_new.copy()
            Wq_old= Wq_new.copy()
            means_old=means_new.copy()
            cov_old=cov_new.copy()

        run= run +1

    if convergence==True and run!=runs:
        print("Iterations for convergence=",iter_convergence)
    else:
        print("Estimate has not converged yet, more runs needed")
    print(f"Final log-likehood = {ll_new}")

    print("\nEffective number of elements in each cluster is")
    print(Nq_new)
#     ass=np.sum(Nq_new)
```

```
#     print(ass)
    weights.append(Wq_new)
    means.append(means_new)
    cov.append(cov_new)

print("\n################################################################################
```

```
Class 0

Original effective number of elements in each cluster
[51. 55. 36. 48. 61.]

Initial log-likehood = [6644.56793165]
Iterations for convergence= 35
Final log-likehood = [6989.07946055]

Effective number of elements in each cluster is
[71.39136033 54.93077138 36.90132483 40.69025424 47.08628921]



Class 1

Original effective number of elements in each cluster
[24. 53. 43. 70. 39.]

Initial log-likehood = [8410.07720573]
Iterations for convergence= 43
Final log-likehood = [8728.77721154]

Effective number of elements in each cluster is
[29.46181747 51.44054737 38.94724932 53.1388349  56.01155094]



Class 2

Original effective number of elements in each cluster
[61. 39. 76. 71. 40.]

Initial log-likehood = [7592.62443417]
Iterations for convergence= 43
Final log-likehood = [7896.6036196]

Effective number of elements in each cluster is
[57.76236741 47.8324425  48.11819658 74.75620343 58.53079009]



Class 3

Original effective number of elements in each cluster
[39. 49. 24. 47. 45.]

Initial log-likehood = [7166.81527559]
Iterations for convergence= 22
Final log-likehood = [7298.28866049]

Effective number of elements in each cluster is
[42.86134831 47.59694932 34.01570974 41.48789368 38.03809894]



Class 4

Original effective number of elements in each cluster
[34. 54. 58. 48. 55.]
```

```
Initial log-likehood = [7504.76678148]
Iterations for convergence= 21
Final log-likehood = [7883.57970786]

Effective number of elements in each cluster is
[44.23143498 38.76756392 68.51009317 53.99148447 43.49942346]

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

In [2]:

```python
size=np.array(size)
prior_class=size/np.sum(size)

validation_set=['coast','forest','opencountry','street','tallbuilding']
# validation_set=['train_1.csv','train_2.csv','train_3.csv','train_4.csv','train_5.c


valid_data=pd.DataFrame()
test_data=pd.DataFrame()
train_data=pd.DataFrame()
for ind, valid_file in enumerate(validation_set):

    X_valid=pd.read_csv('dataset/'+valid_file+'/dev.csv')
    X_valid['y']=ind
    msk = np.random.rand(len(X_valid)) < 0.5  #50-50 splits
    #print(X_valid[msk])
    valid_data=pd.concat([valid_data,X_valid[msk]],ignore_index=True)
    test_data=pd.concat([test_data,X_valid[~msk]],ignore_index=True)

for ind, valid_file in enumerate(validation_set):

    X_valid=pd.read_csv('dataset/'+valid_file+'/train.csv')
    X_valid['y']=ind

    #print(X_valid[msk])
    #valid_data=pd.concat([valid_data,X_valid[msk]],ignore_index=True)
    #test_data=pd.concat([test_data,X_valid[~msk]],ignore_index=True)
    train_data=pd.concat([train_data,X_valid],ignore_index=True)
```

In [3]:

```python
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import seaborn as sn
predicted=[]
real=[]
#k=3
#print(len(validation_set))
for i in range(len(valid_data)):

    X_valid=(valid_data.loc[i,:]).to_numpy()
    #print(X_valid)
    X_valid=X_valid[1:]
    Y_valid=X_valid[-1]
    X_valid=X_valid[:-1]
    real.append(Y_valid)


    #for n in range(len(X_valid)):
    ll_n=[]
    for i in range(len(validation_set)):
        ll= np.log(sum([weights[i][j]*multivariate_normal.pdf(X_valid, means[i][j],
        ll_n.append(ll)
```

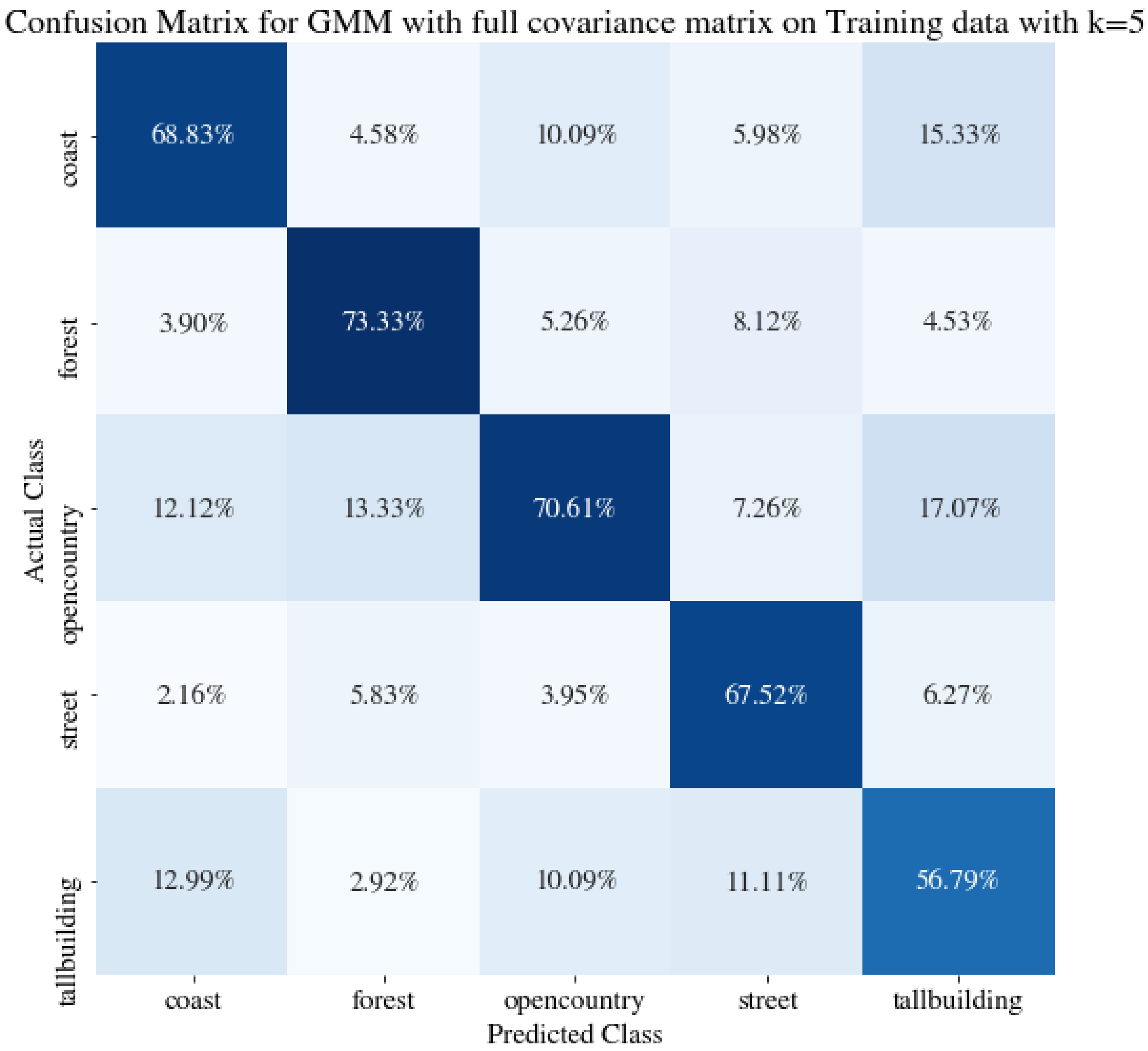
```
    ll_n=np.array(ll_n)
    predicted.append(np.argmax(ll_n))


print("accuracy on validation set using full covariance matrix and k="+str(k)+ " is
```

```
accuracy on validation set using full covariance matrix and k=5 is 57.42574257425742
```

In [6]:

```
predicted=[]
real=[]


for i in range(len(train_data)):

    X_valid=(train_data.loc[i,:]).to_numpy()
    #print(X_valid)
    X_valid=X_valid[1:]
    Y_valid=X_valid[-1]
    X_valid=X_valid[:-1]
    real.append(Y_valid)


    #for n in range(len(X_valid)):
    ll_n=[]
    for i in range(len(validation_set)):
        ll= np.log(sum([weights[i][j]*multivariate_normal.pdf(X_valid, means[i][j],
        ll_n.append(ll)
    ll_n=np.array(ll_n)
    predicted.append(np.argmax(ll_n))

    #p=index.count(ind)
    #prob=p/len(index)

    #print(prob)

print("accuracy on Training set using full covariance matrix and k="+str(k)+ " is "

if k==5:

    confuse=confusion_matrix(real,predicted)

    sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
        fmt='.2%', cmap='Blues',cbar=False,xticklabels=validation_set,yticklabels=va
    plt.xlabel('Predicted Class')
    plt.ylabel("Actual Class")
    plt.title('Confusion Matrix for GMM with full covariance matrix on Training data
    plt.savefig('Confusion_train_2.png')
    #plt.xaxis.set_ticklabels(validation_set);
    #ax.yaxis.set_ticklabels(validation_set[::-1]);
    plt.show()
```

```
accuracy on Training set using full covariance matrix and k=5 is 66.9672131147541
```



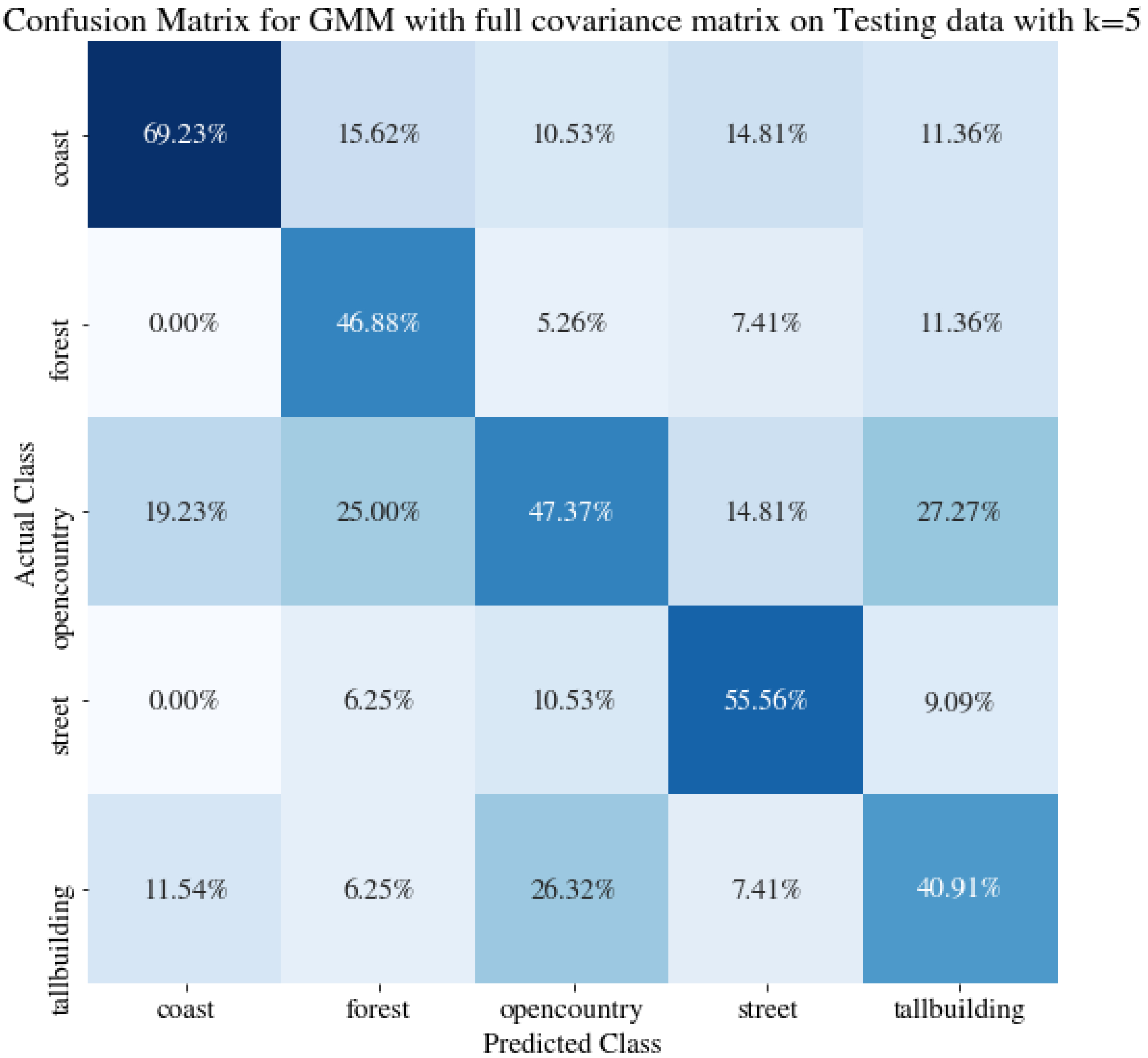
In [ ]:

In [7]:

```
predicted=[]
real=[]
#k=3
#print(len(validation_set))

for i in range(len(test_data)):

    X_valid=(test_data.loc[i,:]).to_numpy()
    #print(X_valid)
    X_valid=X_valid[1:]
    Y_valid=X_valid[-1]
    X_valid=X_valid[:-1]
    real.append(Y_valid)


    #for n in range(len(X_valid)):
    ll_n=[]
    for i in range(len(validation_set)):
        ll= np.log(sum([weights[i][j]*multivariate_normal.pdf(X_valid, means[i][j],
        ll_n.append(ll)
    ll_n=np.array(ll_n)
    predicted.append(np.argmax(ll_n))

    #p=index.count(ind)
    #prob=p/len(index)
```

```
    #print(prob)

print("accuracy on Testing set using full covariance matrix and k="+str(k)+ " is " +

if k==5:

    confuse=confusion_matrix(real,predicted)

    sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
        fmt='.2%', cmap='Blues',cbar=False,xticklabels=validation_set,yticklabels=va
    plt.xlabel('Predicted Class')
    plt.ylabel("Actual Class")
    plt.title('Confusion Matrix for GMM with full covariance matrix on Testing data
    plt.savefig('Confusion_test_2.png')
    #plt.xaxis.set_ticklabels(validation_set);
    #ax.yaxis.set_ticklabels(validation_set[::-1]);
    plt.show()
```

```
accuracy on Testing set using full covariance matrix and k=5 is 50.67567567567568
```




```
In [ ]:
```

Dataset 2(B) Code for Bayes Classifier with a GMM for each class using full covariance matrices

In [8]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn.cluster import KMeans
from scipy.stats import multivariate_normal
import os


plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['font.size'] = 15
plt.rcParams["figure.figsize"] = (9,9)



size=[]
weights=[]
cov=[]
means=[]

def gauss(X, mean_vector, covariance_matrix):
    if (np.abs(np.linalg.det(covariance_matrix))==0):
        print("ERROR")
     # a= (2*np.pi)**(-len(X)/2)*np.abs(np.prod((np.linalg.eigvals(covariance_matrix
    b= (2*np.pi)**(-len(X)/2)*(np.linalg.det(covariance_matrix))**(-1/2)*np.exp(-np.
     # c= ((1/(((2*math.pi)**(X.shape[0]/2))*((np.linalg.det(covariance_matrix))**0.
     # return (2*np.pi)**(-len(X)/2)*np.linalg.det(covariance_matrix)**(-1/2)*np.exp

    return b

# The only hyperparameter is k ( no.of components for each class)
k=2
train=['coast','forest','opencountry','street','tallbuilding']

for c, train_file in enumerate(train):
    arr = os.listdir('./'+train_file+'/train')
    data=pd.DataFrame()

    for i in range(len(arr)):
        data_2=pd.read_csv(train_file+'/train/'+arr[i],header=None,delim_whitespace=
#         print(data.shape)
        #coast_train.concat(data)
        data=pd.concat([data,data_2],ignore_index=True)


    #data=pd.read_csv('dataset/'+train_file+'/train.csv')
    data=data.to_numpy()
    X=data
    size.append(len(X))
    print(f"\n\n\nClass {c}")
#     print(size)
#     kmeans=KMeans(n_clusters=k,random_state=0).fit(X)
    kmeans=KMeans(n_clusters=k).fit(X)
    means_old=kmeans.cluster_centers_
    labels=kmeans.labels_


    N=len(X)
```

```python
    r_old=np.zeros((len(X),k)) # form a Z ( indicator ) matrix

    for i in range(len(X)):
        r_old[i,labels[i]]=1

    Nq_old=np.sum(r_old,axis=0) # sum conatins the number of elements belonging
                                # to each cluster

    print("\nOriginal effective number of elements in each cluster")
    print(Nq_old)

    # Initialization

    #cov2 is a 3-d array containing the covariance matrix of each cluster
    cov_old=np.zeros([k,X.shape[1],X.shape[1]])
    Wq_old =np.zeros([k,1]) ## weight of each cluster

    for i in range(k):
        Nq=Nq_old[i]
        Wq_old[i]= Nq/N
        tp=np.zeros([X.shape[1],X.shape[1]])

        for p in range(X.shape[0]):
            le=X[p,:]-means_old[i]
            le=np.reshape(le,[le.shape[0],1])
            tp=tp+r_old[p,i]*(np.dot(le,le.T))
        tp=tp/Nq

#         d= np.diag(tp)
#         tp=np.diag(d)
        cov_old[i,:,:]=tp.copy()

    ll_old= 0.0
    for n in range(len(X)):
        ll_old = ll_old + np.log(sum([Wq_old[j]*multivariate_normal.pdf(X[n], means_

    print(f"\nInitial log-likehood = {ll_old}")

    convergence=False
    iter_convergence=0
    run=0
    runs=1000

    while (convergence == False and run<runs):

        # ''' --------------------------   E - STEP   -------------------------- '''

        # Initiating the r matrix, every row contains the probabilities
        # for every cluster for this row

        r_new = np.zeros((len(X), k))  # responsibilty matrix

        # Calculating the r matrix
        for n in range(len(X)):
            for i in range(k):
                r_new[n][i] = Wq_old[i] * multivariate_normal.pdf(X[n], means_old[i]
                r_new[n][i] /= sum([Wq_old[j]*multivariate_normal.pdf(X[n], means_ol

        # Calculating the N effective elemts fro each component
        Nq_new = np.sum(r_new, axis=0)


        # ''' --------------------------   M - STEP   -------------------------- '''
```

```
        # Updating the weights list
        Wq_new =np.zeros([k,1]) ## weight of each cluster
        for i in range(k):
            Wq_new[i]= Nq_new[i]/ N


        # Initializing the mean vector as a zero vector
        means_new = np.zeros((k, len(X[0])))

        # Updating the mean vector
        for i in range(k):
            for n in range(len(X)):
                means_new[i] = means_new[i] + r_new[n][i] * X[n]
            means_new[i] = means_new [i]/Nq_new[i]



        # Initiating the list of the covariance matrixes
        cov_new =np.zeros([k,X.shape[1],X.shape[1]])

        # Updating the covariance matrices
        for i in range(k):
            Nq=Nq_new[i]
            tp=np.zeros([X.shape[1],X.shape[1]])

            for p in range(X.shape[0]):
                le=X[p,:]-means_new[i]
                le=np.reshape(le,[le.shape[0],1])
                tp=tp+r_new[p,i]*(np.dot(le,le.T))

            tp=tp/Nq
#             d= np.diag(tp)
#             tp=np.diag(d)
            cov_new[i,:,:]=tp.copy()


        # print(f"\nRun= {run}\n")
#         print(np.sum(Nq_new))
#         print("\nWeights\n")
#         print(np.sum(Wq_new))
#         print(Wq_new)
#         print(np.sum(r_new))
#         print("\n------------------")

        # Calculating log-likelhood
        ll_new=0
        for n in range(len(X)):
            ll_new = ll_new + np.log(sum([Wq_new[j]*multivariate_normal.pdf(X[n], me

    #     print(ll_new)
        diff=ll_new-ll_old

    #     print(diff)

        #Convergence condition
        if diff < 1e-3:
            iter_convergence=run
            convergence=True
            break

        else:
            ll_old=ll_new.copy()
            Wq_old= Wq_new.copy()
            means_old=means_new.copy()
            cov_old=cov_new.copy()
```

```
        run= run +1

    if convergence==True and run!=runs:
        print("Iterations for convergence=",iter_convergence)
    else:
        print("Estimate has not converged yet, more runs needed")
    print(f"Final log-likehood = {ll_new}")

    print("\nEffective number of elements in each cluster is")
    print(Nq_new)
#     ass=np.sum(Nq_new)
#     print(ass)
    weights.append(Wq_new)
    means.append(means_new)
    cov.append(cov_new)

print("\n##############################################################################
```

```
Class 0

Original effective number of elements in each cluster
[3127. 5909.]

Initial log-likehood = [523273.96692113]
Iterations for convergence= 2
Final log-likehood = [495726.8663807]

Effective number of elements in each cluster is
[2449.80274617 6586.19725383]



Class 1

Original effective number of elements in each cluster
[7245.  999.]

Initial log-likehood = [526735.12958772]
Iterations for convergence= 0
Final log-likehood = [471122.80058415]

Effective number of elements in each cluster is
[7160.78802321 1083.21197679]



Class 2

Original effective number of elements in each cluster
[7839. 2493.]

Initial log-likehood = [621385.50613882]
Iterations for convergence= 0
Final log-likehood = [579457.04604456]

Effective number of elements in each cluster is
[7672.79625626 2659.20374374]



Class 3

Original effective number of elements in each cluster
[1522. 5822.]
```

```
Initial log-likehood = [425236.79316343]
Iterations for convergence= 0
Final log-likehood = [390531.95426097]

Effective number of elements in each cluster is
[1531.69426195 5812.30573805]



Class 4

Original effective number of elements in each cluster
[2937. 6027.]

Initial log-likehood = [497354.33301084]
Iterations for convergence= 0
Final log-likehood = [465338.73866488]

Effective number of elements in each cluster is
[2932.10442955 6031.89557045]

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

In [9]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix


size=np.array(size)
prior_class=size/np.sum(size)

validation_set=['coast','forest','opencountry','street','tallbuilding']


real=[]
predicted=[]

for c, train_file in enumerate(validation_set):
    arr = os.listdir('./'+train_file+'/dev')

    for i in range(int(len(arr)/2)):  #only 50% of dev_set is validation
        data_2=pd.read_csv(train_file+'/dev/'+arr[i],header=None,delim_whitespace=Tr
        data=data_2.to_numpy()
        real.append(c)
        ll_n=[]
        for i in range(len(validation_set)):
            ll=0
            for p in range(data.shape[0]):
                ll+= np.log(sum([weights[i][j]*multivariate_normal.pdf(data[p], mean
            ll_n.append(ll+np.log(prior_class[i]))
        ll_n=np.array(ll_n)
        predicted.append(np.argmax(ll_n))

print("accuracy on validation set using full covariance matrix and k="+str(k)+ " is
```

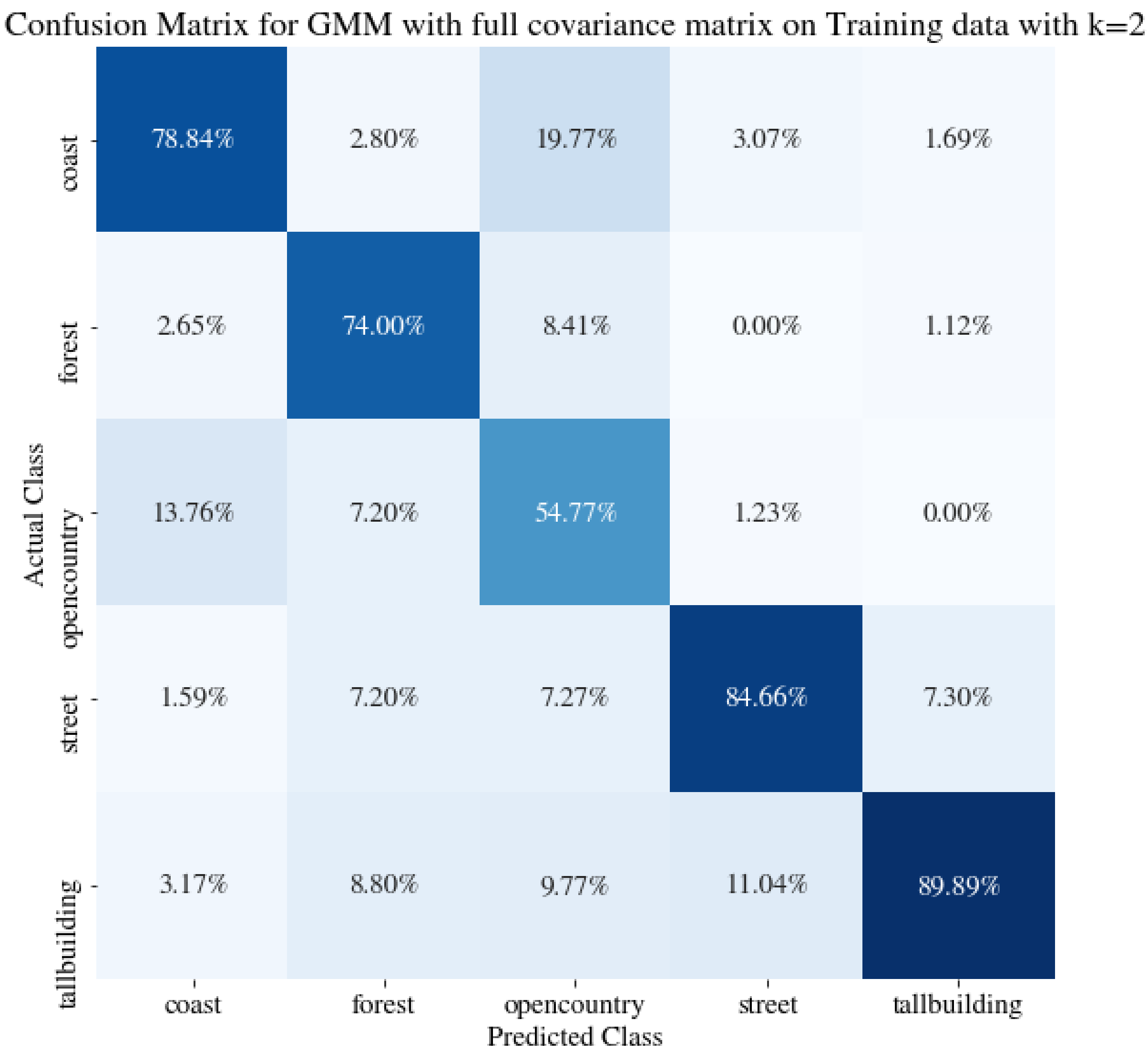
```
accuracy on validation set using full covariance matrix and k=2 is 71.26436781609196
```

In [10]:

```
import seaborn as sn
size=np.array(size)
prior_class=size/np.sum(size)

training_set=['coast','forest','opencountry','street','tallbuilding']
```

```python
real=[]
predicted=[]

for c, train_file in enumerate(training_set):
    arr = os.listdir('./'+train_file+'/train')

    for i in range(int(len(arr))):
        data_2=pd.read_csv(train_file+'/train/'+arr[i],header=None,delim_whitespace=
        data=data_2.to_numpy()
        real.append(c)
        ll_n=[]
        for i in range(len(training_set)):
            ll=0
            for p in range(data.shape[0]):
                ll+= np.log(sum([weights[i][j]*multivariate_normal.pdf(data[p], mean
            ll_n.append(ll+np.log(prior_class[i]))
        ll_n=np.array(ll_n)
        predicted.append(np.argmax(ll_n))

print("accuracy on Training set using full covariance matrix and k="+str(k)+ " is "

if k==2:
    confuse=confusion_matrix(real,predicted)

    sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
        fmt='.2%', cmap='Blues',cbar=False,xticklabels=training_set,yticklabels=trai
    plt.xlabel('Predicted Class')
    plt.ylabel("Actual Class")
    plt.title('Confusion Matrix for GMM with full covariance matrix on Training data
#     plt.savefig('Confusion_train_1.png')
    #plt.xaxis.set_ticklabels(validation_set);
    #ax.yaxis.set_ticklabels(validation_set[::-1]);
    plt.show()
```

```
<ipython-input-10-9b2622a9f48e>:22: RuntimeWarning: divide by zero encountered in lo
g
  ll+= np.log(sum([weights[i][j]*multivariate_normal.pdf(data[p], means[i][j], cov
[i][j],allow_singular=True)  for j in range(k)]))
accuracy on Training set using full covariance matrix and k=2 is 71.55737704918033
```

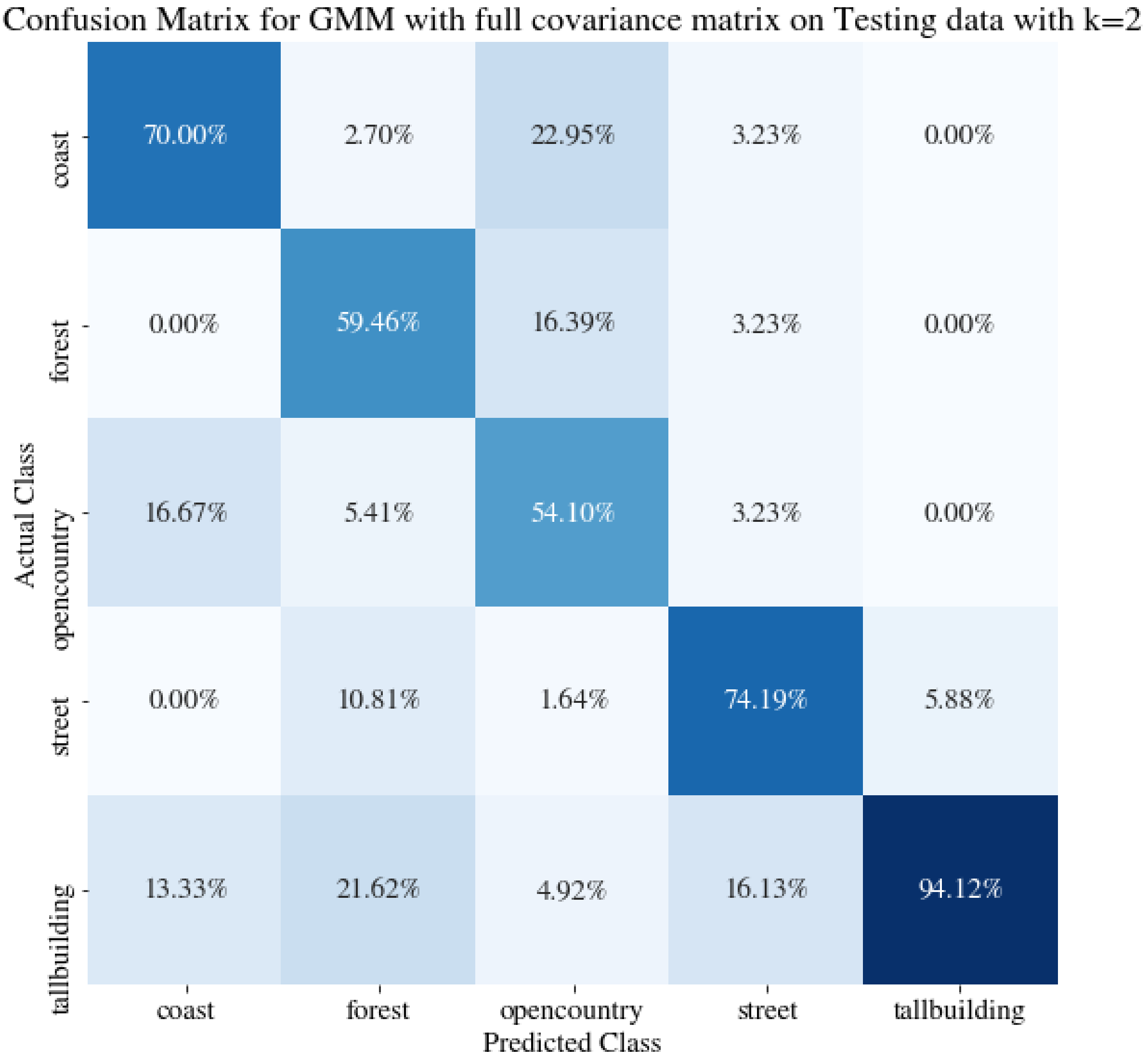


In [11]:

```
if k==2:
    size=np.array(size)
    prior_class=size/np.sum(size)

    test_set=['coast','forest','opencountry','street','tallbuilding']


    real=[]
    predicted=[]

    for c, train_file in enumerate(test_set):
        arr = os.listdir('./'+train_file+'/dev')
        #data=pd.DataFrame()

        for i in range(int(len(arr)/2),len(arr)):
            data_2=pd.read_csv(train_file+'/dev/'+arr[i],header=None,delim_whitespac
            data=data_2.to_numpy()
            real.append(c)
            ll_n=[]
            for i in range(len(test_set)):
                ll=0
                for p in range(data.shape[0]):
                    ll+= np.log(sum([weights[i][j]*multivariate_normal.pdf(data[p],
                ll_n.append(ll+np.log(prior_class[i]))
            ll_n=np.array(ll_n)
            predicted.append(np.argmax(ll_n))

    print("accuracy on test set using full covariance matrix and k="+str(k)+ " is "
```

```
    confuse=confusion_matrix(real,predicted)

    sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
        fmt='.2%', cmap='Blues',cbar=False,xticklabels=test_set,yticklabels=test_set
    plt.xlabel('Predicted Class')
    plt.ylabel("Actual Class")
    plt.title('Confusion Matrix for GMM with full covariance matrix on Testing data
#     plt.savefig('Confusion_test_1.png')
    #plt.xaxis.set_ticklabels(validation_set);
    #ax.yaxis.set_ticklabels(validation_set[::-1]);
    plt.show()
```

```
accuracy on test set using full covariance matrix and k=2 is 65.3409090909091
```



In [ ]:

In [ ]:

Dataset (B) Code for Bayes Classifier with a GMM for each class using diagonal covariance matrices

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn.cluster import KMeans
from scipy.stats import multivariate_normal
import os


plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.family'] = 'STIXGeneral'
plt.rcParams['font.size'] = 15
plt.rcParams["figure.figsize"] = (9,9)



size=[]
weights=[]
cov=[]
means=[]

def gauss(X, mean_vector, covariance_matrix):
    if (np.abs(np.linalg.det(covariance_matrix))==0):
        print("ERROR")
     # a= (2*np.pi)**(-len(X)/2)*np.abs(np.prod((np.linalg.eigvals(covariance_matrix
    b= (2*np.pi)**(-len(X)/2)*(np.linalg.det(covariance_matrix))**(-1/2)*np.exp(-np.
     # c= ((1/(((2*math.pi)**(X.shape[0]/2))*((np.linalg.det(covariance_matrix))**0.
     # return (2*np.pi)**(-len(X)/2)*np.linalg.det(covariance_matrix)**(-1/2)*np.exp

    return b

# The only hyperparameter is k ( no.of components for each class)
k=5
train=['coast','forest','opencountry','street','tallbuilding']

for c, train_file in enumerate(train):
    arr = os.listdir('./'+train_file+'/train')
    data=pd.DataFrame()

    for i in range(len(arr)):
        data_2=pd.read_csv(train_file+'/train/'+arr[i],header=None,delim_whitespace=
#         print(data.shape)
        #coast_train.concat(data)
        data=pd.concat([data,data_2],ignore_index=True)


    #data=pd.read_csv('dataset/'+train_file+'/train.csv')
    data=data.to_numpy()
    X=data
    size.append(len(X))
    print(f"\n\n\nClass {c}")
#     print(size)
    kmeans=KMeans(n_clusters=k,random_state=0).fit(X)
    # kmeans=KMeans(n_clusters=k).fit(X)
    means_old=kmeans.cluster_centers_
    labels=kmeans.labels_


    N=len(X)
```

```
r_old=np.zeros((len(X),k)) # form a Z ( indicator ) matrix

for i in range(len(X)):
    r_old[i,labels[i]]=1

Nq_old=np.sum(r_old,axis=0) # sum conatins the number of elements belonging
                             # to each cluster

print("\nOriginal effective number of elements in each cluster")
print(Nq_old)

# Initialization

#cov2 is a 3-d array containing the covariance matrix of each cluster
cov_old=np.zeros([k,X.shape[1],X.shape[1]])
Wq_old =np.zeros([k,1]) ## weight of each cluster

for i in range(k):
    Nq=Nq_old[i]
    Wq_old[i]= Nq/N
    tp=np.zeros([X.shape[1],X.shape[1]])

    for p in range(X.shape[0]):
        le=X[p,:]-means_old[i]
        le=np.reshape(le,[le.shape[0],1])
        tp=tp+r_old[p,i]*(np.dot(le,le.T))
    tp=tp/Nq

    d= np.diag(tp)
    tp=np.diag(d)
    cov_old[i,:,:]=tp.copy()

ll_old= 0.0
for n in range(len(X)):
    ll_old = ll_old + np.log(sum([Wq_old[j]*multivariate_normal.pdf(X[n], means_

print(f"\nInitial log-likehood = {ll_old}")

convergence=False
iter_convergence=0
run=0
runs=1000

while (convergence == False and run<runs):

    # ''' --------------------------   E - STEP   -------------------------- '''

    # Initiating the r matrix, every row contains the probabilities
    # for every cluster for this row

    r_new = np.zeros((len(X), k))  # responsibilty matrix

    # Calculating the r matrix
    for n in range(len(X)):
        for i in range(k):
            r_new[n][i] = Wq_old[i] * multivariate_normal.pdf(X[n], means_old[i]
            r_new[n][i] /= sum([Wq_old[j]*multivariate_normal.pdf(X[n], means_ol

    # Calculating the N effective elemts fro each component
    Nq_new = np.sum(r_new, axis=0)


    # ''' --------------------------   M - STEP   -------------------------- '''
```

```
        # Updating the weights list
        Wq_new =np.zeros([k,1]) ## weight of each cluster
        for i in range(k):
            Wq_new[i]= Nq_new[i]/ N


        # Initializing the mean vector as a zero vector
        means_new = np.zeros((k, len(X[0])))

        # Updating the mean vector
        for i in range(k):
            for n in range(len(X)):
                means_new[i] = means_new[i] + r_new[n][i] * X[n]
            means_new[i] = means_new [i]/Nq_new[i]



        # Initiating the list of the covariance matrixes
        cov_new =np.zeros([k,X.shape[1],X.shape[1]])

        # Updating the covariance matrices
        for i in range(k):
            Nq=Nq_new[i]
            tp=np.zeros([X.shape[1],X.shape[1]])

            for p in range(X.shape[0]):
                le=X[p,:]-means_new[i]
                le=np.reshape(le,[le.shape[0],1])
                tp=tp+r_new[p,i]*(np.dot(le,le.T))

            tp=tp/Nq
            d= np.diag(tp)
            tp=np.diag(d)
            cov_new[i,:,:]=tp.copy()


        # print(f"\nRun= {run}\n")
#         print(np.sum(Nq_new))
#         print("\nWeights\n")
#         print(np.sum(Wq_new))
#         print(Wq_new)
#         print(np.sum(r_new))
#         print("\n-----------------")

        # Calculating log-likelhood
        ll_new=0
        for n in range(len(X)):
            ll_new = ll_new + np.log(sum([Wq_new[j]*multivariate_normal.pdf(X[n], me

    #     print(ll_new)
        diff=ll_new-ll_old

    #     print(diff)

        #Convergence condition
        if diff < 1e-3:
            iter_convergence=run
            convergence=True
            break

        else:
            ll_old=ll_new.copy()
            Wq_old= Wq_new.copy()
            means_old=means_new.copy()
            cov_old=cov_new.copy()
```

```
        run= run +1

    if convergence==True and run!=runs:
        print("Iterations for convergence=",iter_convergence)
    else:
        print("Estimate has not converged yet, more runs needed")
    print(f"Final log-likehood = {ll_new}")

    print("\nEffective number of elements in each cluster is")
    print(Nq_new)
#     ass=np.sum(Nq_new)
#     print(ass)
    weights.append(Wq_new)
    means.append(means_new)
    cov.append(cov_new)

print("\n##############################################################################
```

```
Class 0

Original effective number of elements in each cluster
[1593. 3041. 1791.  971. 1640.]

Initial log-likehood = [531745.86007076]
Iterations for convergence= 5
Final log-likehood = [551273.51989332]

Effective number of elements in each cluster is
[2933.9508137  2195.66249795 1103.66668962  830.90287671 1971.81712202]



Class 1

Original effective number of elements in each cluster
[2609. 2096.  301. 1962. 1276.]

Initial log-likehood = [535274.36974155]
Iterations for convergence= 5
Final log-likehood = [541394.35873173]

Effective number of elements in each cluster is
[3194.20981524 1855.79715683  538.92401554 1156.71020167 1498.35881072]



Class 2

Original effective number of elements in each cluster
[2172. 1791.  899. 3073. 2397.]

Initial log-likehood = [630806.57373694]
Iterations for convergence= 5
Final log-likehood = [650463.26181028]

Effective number of elements in each cluster is
[1334.53701847 2675.04050027  970.76915767 2896.43478371 2455.21853988]



Class 3

Original effective number of elements in each cluster
[ 493. 1336. 2397. 1888. 1230.]
```

```
Initial log-likehood = [430028.26965779]
Iterations for convergence= 22
Final log-likehood = [437964.27573135]

Effective number of elements in each cluster is
[ 316.30348546 1505.89653762 1969.85095643 2265.57661141 1286.37240907]



Class 4

Original effective number of elements in each cluster
[1789. 1187. 2585. 1807. 1596.]

Initial log-likehood = [496476.62498885]
Iterations for convergence= 15
Final log-likehood = [513094.59156451]

Effective number of elements in each cluster is
[2555.06528165 1190.61025397 1227.6808494  1560.01291503 2430.63069995]

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

In [2]:

```python
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix


size=np.array(size)
prior_class=size/np.sum(size)

validation_set=['coast','forest','opencountry','street','tallbuilding']


real=[]
predicted=[]

for c, train_file in enumerate(validation_set):
    arr = os.listdir('./'+train_file+'/dev')

    for i in range(int(len(arr)/2)):  #only 50% of dev_set is validation
        data_2=pd.read_csv(train_file+'/dev/'+arr[i],header=None,delim_whitespace=Tr
        data=data_2.to_numpy()
        real.append(c)
        ll_n=[]
        for i in range(len(validation_set)):
            ll=0
            for p in range(data.shape[0]):
                ll+= np.log(sum([weights[i][j]*multivariate_normal.pdf(data[p], mean
            ll_n.append(ll+np.log(prior_class[i]))
        ll_n=np.array(ll_n)
        predicted.append(np.argmax(ll_n))

print("accuracy on validation set using diagional covariance matrix and k="+str(k)+
```

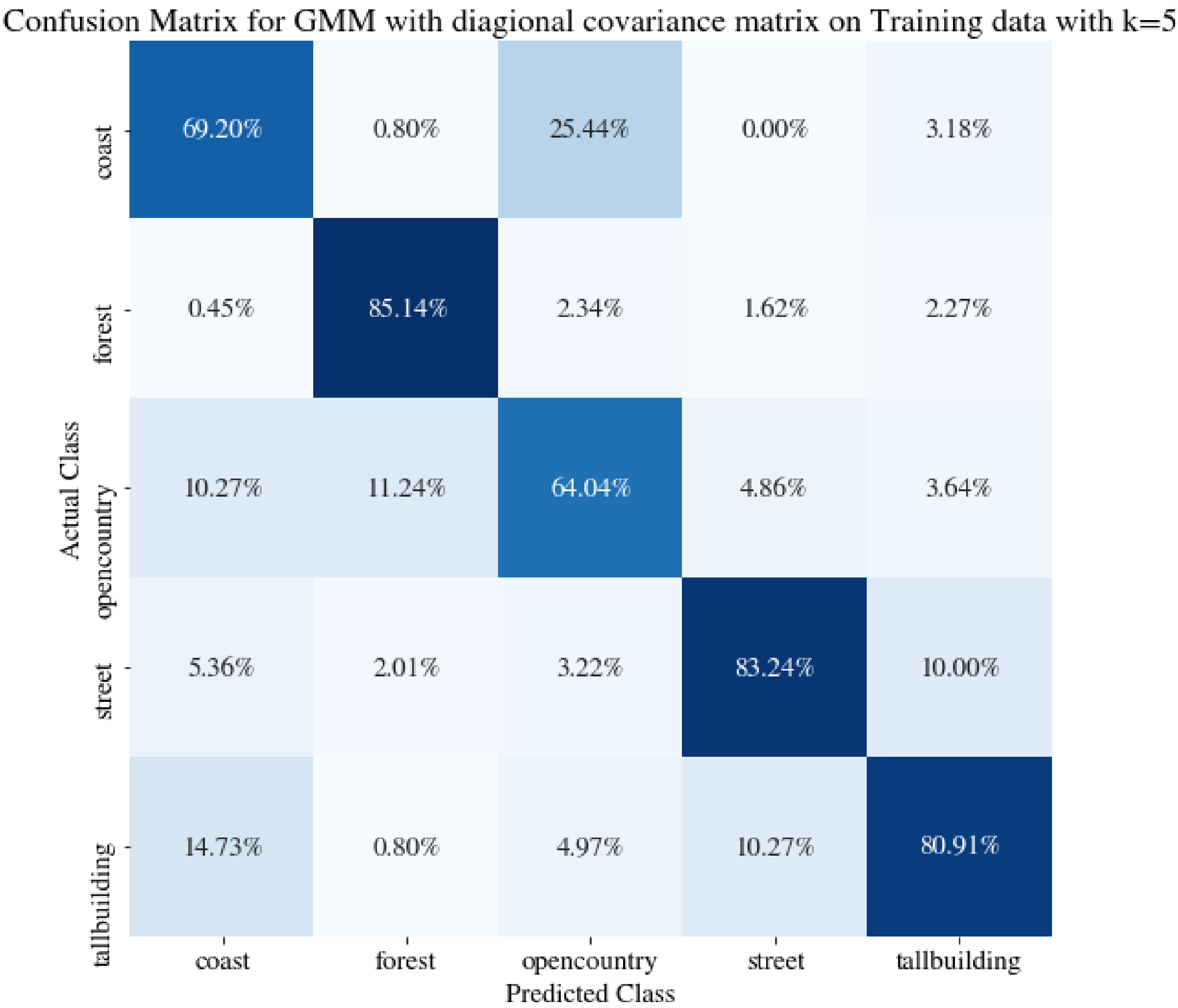
```
accuracy on validation set using diagional covariance matrix and k=5 is 78.160919540
22988
```

In [3]:

```python
import seaborn as sn
size=np.array(size)
prior_class=size/np.sum(size)

training_set=['coast','forest','opencountry','street','tallbuilding']
```

```
real=[]
predicted=[]

for c, train_file in enumerate(training_set):
    arr = os.listdir('./'+train_file+'/train')

    for i in range(int(len(arr))):
        data_2=pd.read_csv(train_file+'/train/'+arr[i],header=None,delim_whitespace=
        data=data_2.to_numpy()
        real.append(c)
        ll_n=[]
        for i in range(len(training_set)):
            ll=0
            for p in range(data.shape[0]):
                ll+= np.log(sum([weights[i][j]*multivariate_normal.pdf(data[p], mean
            ll_n.append(ll+np.log(prior_class[i]))
        ll_n=np.array(ll_n)
        predicted.append(np.argmax(ll_n))

print("accuracy on Training set using diagional covariance matrix and k="+str(k)+ "

if k==5:
    confuse=confusion_matrix(real,predicted)

    sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
        fmt='.2%', cmap='Blues',cbar=False,xticklabels=training_set,yticklabels=trai
    plt.xlabel('Predicted Class')
    plt.ylabel("Actual Class")
    plt.title('Confusion Matrix for GMM with diagional covariance matrix on Training
    plt.savefig('Confusion_train_2.png')
    #plt.xaxis.set_ticklabels(validation_set);
    #ax.yaxis.set_ticklabels(validation_set[::-1]);
    plt.show()
```

```
accuracy on Training set using diagional covariance matrix and k=5 is 75.24590163934
425
```

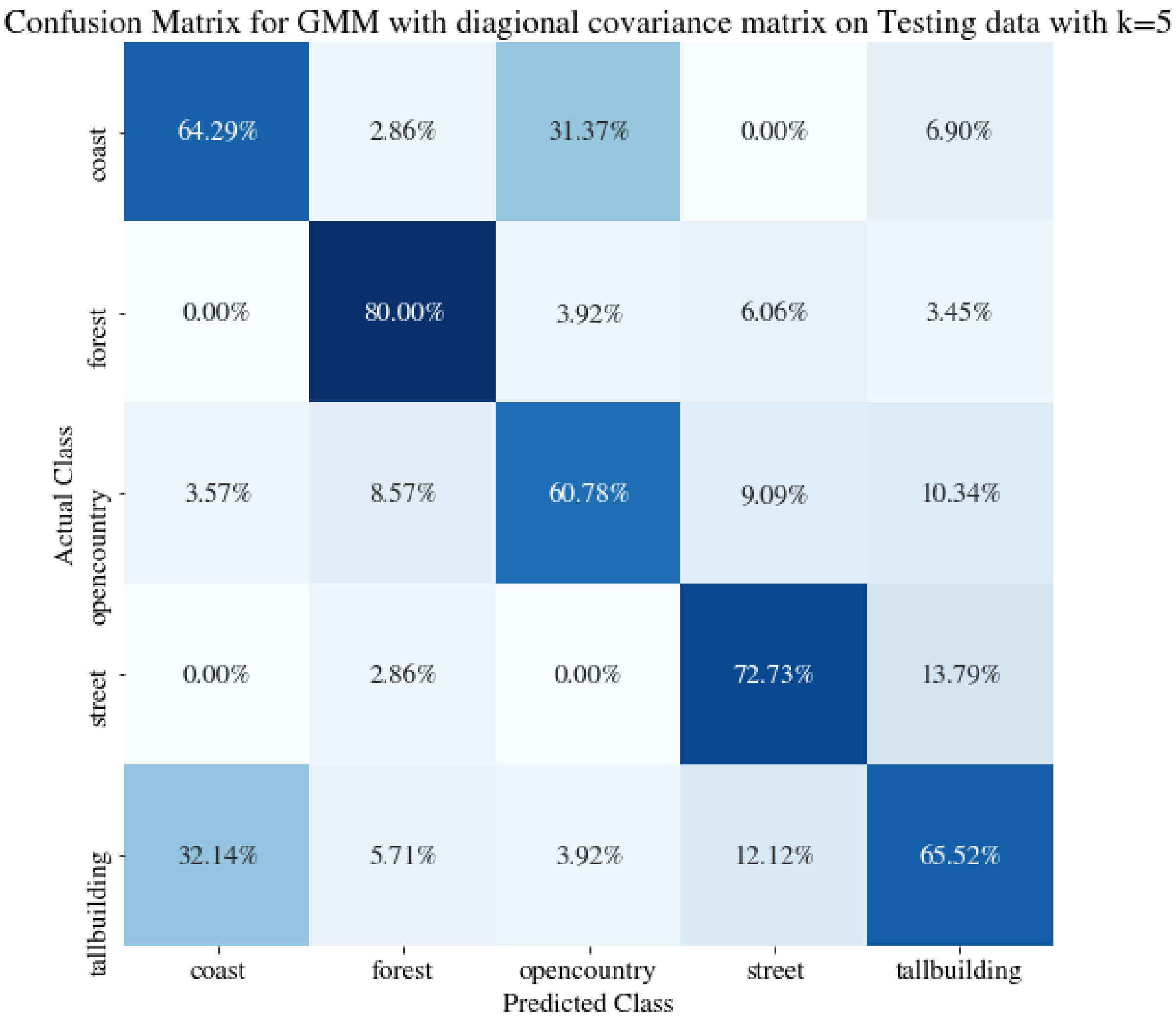


In [4]:

```
size=np.array(size)
prior_class=size/np.sum(size)

test_set=['coast','forest','opencountry','street','tallbuilding']


real=[]
predicted=[]

for c, train_file in enumerate(test_set):
    arr = os.listdir('./'+train_file+'/dev')
    #data=pd.DataFrame()

    for i in range(int(len(arr)/2),len(arr)):
        data_2=pd.read_csv(train_file+'/dev/'+arr[i],header=None,delim_whitespace=Tr
        data=data_2.to_numpy()
        real.append(c)
        ll_n=[]
        for i in range(len(test_set)):
            ll=0
            for p in range(data.shape[0]):
                ll+= np.log(sum([weights[i][j]*multivariate_normal.pdf(data[p], mean
            ll_n.append(ll+np.log(prior_class[i]))
        ll_n=np.array(ll_n)
        predicted.append(np.argmax(ll_n))

print("accuracy on test set using diagional covariance matrix and k="+str(k)+ " is "

if k==5:
```

```python
confuse=confusion_matrix(real,predicted)

sn.heatmap(confuse/np.sum(confuse,axis=0), annot=True,
    fmt='.2%', cmap='Blues',cbar=False,xticklabels=test_set,yticklabels=test_set
plt.xlabel('Predicted Class')
plt.ylabel("Actual Class")
plt.title('Confusion Matrix for GMM with diagional covariance matrix on Testing
plt.savefig('Confusion_test_2.png')
#plt.xaxis.set_ticklabels(validation_set);
#ax.yaxis.set_ticklabels(validation_set[::-1]);
plt.show()
```

```
accuracy on test set using diagional covariance matrix and k=5 is 68.18181818181817
```



In [ ]: