CPSC 552 - Assignment # 3

Problem #1: Program the Gaussian Mixture model for the multivariate Gaussian distribution, and test it on the Iris dataset. The Mixture of Gaussians is defined as:

$$p(\vec{x}) = \sum_{i=1}^{K} \phi_i \mathcal{N}(\vec{x} \mid \vec{\mu}_i, \Sigma_i)$$

$$\mathcal{N}(\vec{x} \mid \vec{\mu}_i, \Sigma_i) = \frac{1}{\sqrt{(2\pi)^K |\Sigma_i|}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu}_i)^{\mathrm{T}} \Sigma_i^{-1} (\vec{x} - \vec{\mu}_i)\right)$$

$$\sum_{i=1}^{K} \phi_i = 1$$

The GMM parameters can be computed using the Expectation Maximization algorithm as:

- Randomly assign samples without replacement from the dataset $X = \{x_1, ..., x_N\}$ to the component mean estimates $\hat{\mu}_1, ..., \hat{\mu}_K$. E.g. for $K = 3$ and $N = 100$, set $\hat{\mu}_1 = x_{45}, \hat{\mu}_2 = x_{32}, \hat{\mu}_3 = x_{10}$.
- Set all component variance estimates to the sample variance $\hat{\sigma}_1^2, ..., \hat{\sigma}_K^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2$, where $\bar{x}$ is the sample mean $\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$.
- Set all component distribution prior estimates to the uniform distribution $\hat{\phi}_1, ..., \hat{\phi}_K = \frac{1}{K}$.

Expectation (E) Step:

Calculate $\forall i, k$

$$\hat{\gamma}_{ik} = \frac{\hat{\phi}_k \mathcal{N}(x_i \mid \hat{\mu}_k, \hat{\sigma}_k)}{\sum_{j=1}^{K} \hat{\phi}_j \mathcal{N}(x_i \mid \hat{\mu}_j, \hat{\sigma}_j)},$$

where $\hat{\gamma}_{ik}$ is the probability that x_i is generated by component C_k. Thus, $\hat{\gamma}_{ik} = p(C_k | x_i, \hat{\phi}, \hat{\mu}, \hat{\sigma})$.

Maximization (M) Step:

Using the $\hat{\gamma}_{ik}$ calculated in the expectation step, calculate the following in that order $\forall k$:

- $\hat{\phi}_k = \sum_{i=1}^{N} \frac{\hat{\gamma}_{ik}}{N}$
- $\hat{\mu}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik} x_i}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}$
- $\hat{\sigma}_k^2 = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik} (x_i - \hat{\mu}_k)^2}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}$.

Solution:

```
#-----------GNN_ND.py---------------------
import sys
import pandas as pd
import math
import numpy as np
import random
from scipy.stats import multivariate_normal
import matplotlib.pyplot as plt
from scipy.stats import mode

def compute_gaussian_probab(x, mu, cov, d):  # d is the number of dimensions in data
   xmu = np.matrix(x - mu)
   exponent = np.exp(-0.5 * xmu*np.linalg.inv(cov)*xmu.T)
   denom = 1 / np.sqrt(((2*np.pi)**d) * np.linalg.det(cov))
   res = denom * exponent
   #distribution = multivariate_normal(mu, cov)
   #res2 = distribution.pdf(x)
   return res

def E_Step(xdata,mu,cov,d, kc, phi, gamma):
   #-----------E step-----------------
   N = xdata.shape[0]
   for i in range(0,N):
      denom = 0
      for j in range(0,kc):
         denom = denom + (phi[j] * compute_gaussian_probab(xdata[i,:],mu[j],cov[j],d))
      for k in range(0,kc):
         gamma[i,k] = (phi[k] * compute_gaussian_probab(xdata[i,:],mu[k],cov[k],d))/denom

def M_Step(xdata, mu, cov, d, kc, phi, gamma):
   #-----------M step-----------------
   N = xdata.shape[0]
   phi = np.mean(gamma,axis=0)
   sumgk = np.sum(gamma, axis=0)
   for k in range(0,kc):
      mu[k] = np.zeros((d))
      for i in range(0,N):
         mu[k] = mu[k] + (xdata[i,:] * gamma[i,k])
      mu[k] = mu[k]/sumgk[k]

   for k in range(0,kc):
      cov[k] = np.zeros((d,d))
      for i in range(0,N):
         xmu = np.matrix(xdata[i,:] - mu[k])
         cov[k] = cov[k] + ((xmu.T*xmu) * gamma[i,k])
      cov[k] = cov[k]/sumgk[k]

def plot_iris(xdata, clusters):
   plt.figure(figsize=(10, 6))
   plt.title('GMM Clusters')
   plt.scatter( # since we
     xdata[:,0],
     xdata[:,2],
      c=clusters,
```

```
        cmap=plt.cm.get_cmap('brg'),
        marker='.')
    plt.tight_layout()
    plt.show()

def main():
    #---------------Iris Dataset-------------
    df = pd.read_csv("d:/pythonam3/data/iris.csv")
    #---randomize data
    dfrandom = df #df.sample(frac=1, random_state=1119).reset_index(drop=True)
    # data read from a file is read as a string, so convert the first 4 cols to float
    df1 = dfrandom.iloc[:,0:4].astype(float)
    #---separate out the last column
    df2 = dfrandom.iloc[:,4]
    #---combine the 4 numerical columns and the last column that has the flower category
    #dfrandom = pd.concat([df1,df2],axis=1)

    xdata = df1.values
    xdata = xdata[:,0:4]
    print(xdata.shape)

    #--------------GMM N-D Algorithm-----------------
    kc = 3  # cluster count
    d = 4   # dimensionality of data
    N = xdata.shape[0]
    np.random.seed(42)
    mu = np.zeros((kc,d))
    cov = np.zeros((kc,d,d))

    #-----------initialization step------------
    phi = np.full(kc,1/kc)
    random_row = np.random.randint(low=0, high=150, size=kc)
    mu = np.array([xdata[row,:] for row in random_row ])
    mean_data = np.mean(xdata,axis=0) # mean of entire data (column wise)
    xmu = np.matrix(xdata-mean_data)
    cov = np.array([xmu.T*xmu/N for k in range(kc)])

    print(phi)
    print(mu)
    #print(cov)
    N = len(dfrandom)
    gamma = np.zeros((N,kc))
    print(gamma.shape)

    num_iterations = 20
    for n in range(0,num_iterations):
        E_Step(xdata, mu, cov, d, kc, phi, gamma)
        M_Step(xdata, mu, cov, d, kc, phi, gamma)
        print('----------------iteration =',n)

    #---------final result--------
    print(mu)
    print(gamma)
    print(phi)
```

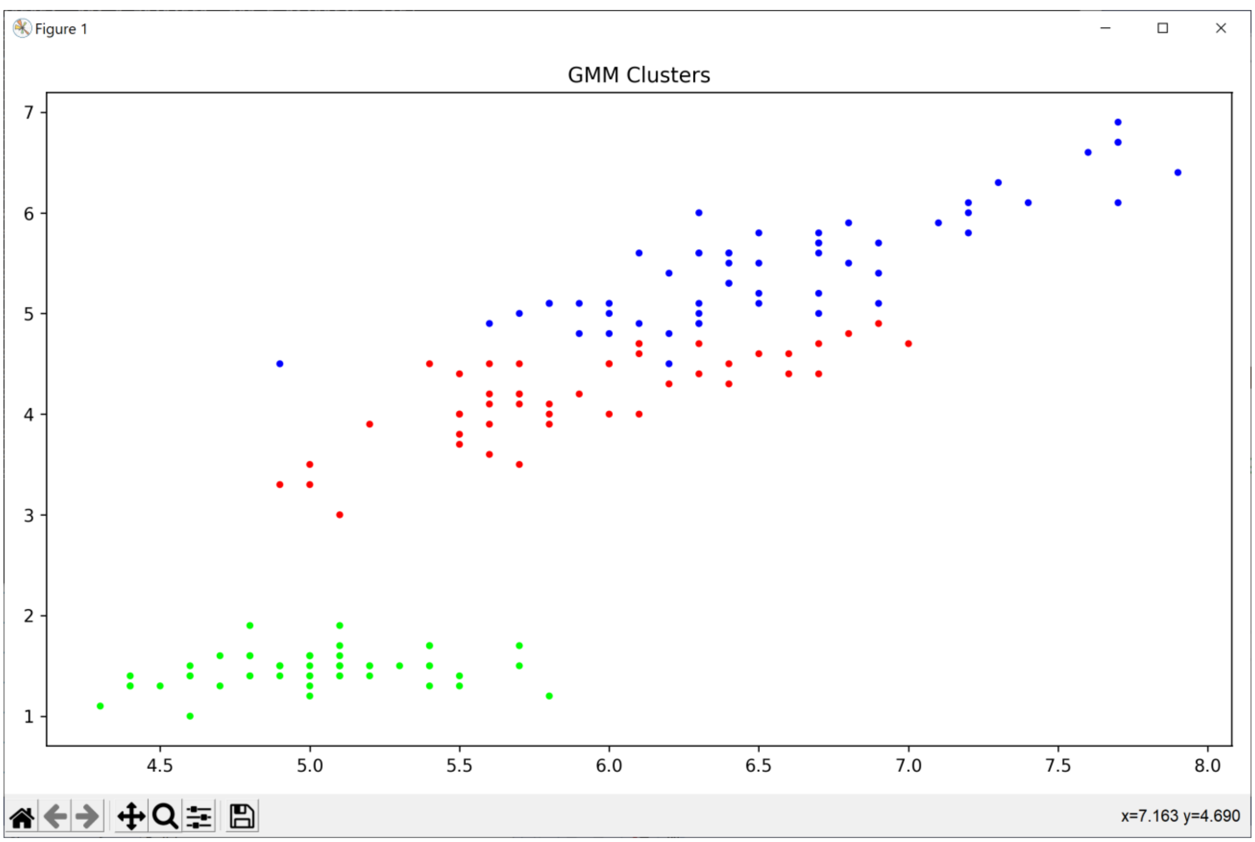
```
    #-----compute accuracy--------
    preds = []
    for i in range(0,N):
        pred = np.argmax(np.multiply(gamma[i,:],phi))
        preds.append(pred)
    print(preds)
    plot_iris(xdata, preds)

    cluster_assigned = []
    # since GMM is unsupervised, class assignments to clusters may vary on each run
    cluster_assigned = [mode(preds[0:50])[0], mode(preds[50:100])[0], mode(preds[100:150])[0]]
    acc = 0
    for i in range(0,N):
        if preds[i] == cluster_assigned[0] and i < 50: # first 50 members belong to class 0
            acc = acc + 1
        if preds[i] == cluster_assigned[1] and i >=50 and i < 100: # next 50 are class 1
            acc = acc + 1
        if preds[i] == cluster_assigned[2] and i >= 100 and i < 150: # last 50, class 2
            acc = acc + 1

    print('accuracy =',acc/N*100)

if __name__ == "__main__":
    sys.exit(int(main() or 0))
```

If run the program, the output appears as:



```
C:\Windows\system32\cmd.exe
 [9.99997493e-001 2.50702650e-006 3.02969417e-140]
 [1.00000000e+000 3.50337329e-010 8.14768783e-162]
 [9.97088085e-001 2.91191482e-003 7.28468582e-124]]
[0.33333333 0.33333333 0.33333333]
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
accuracy = 96.66666666666667
Press any key to continue . . .
```

Note that the GMM algorithm is implemented in the sklearn library. To see if your implementation produces the same results as the GMM in sklearn, add another file to the project with the code as shown below.

```python
#------------------GMM_SKL.py-----------------------
import sklearn
from sklearn.mixture import GaussianMixture
import numpy as np
from scipy import stats
import sys
import pandas as pd
from scipy.stats import mode

def main():
    #---------------Iris Dataset-------------
    df = pd.read_csv("d:/pythonam3/data/iris.csv")
    #---randomize data
    dfrandom = df #df.sample(frac=1, random_state=1119).reset_index(drop=True)
    # data read from a file is read as a string, so convert the first 4 cols to float
    df1 = dfrandom.iloc[:,0:4].astype(float)
    #---separate out the last column
    df2 = dfrandom.iloc[:,4]
    #---combine the 4 numerical columns and the last column that has the flower category
    dfrandom = pd.concat([df1,df2],axis=1)
    print(dfrandom)

    #--------------GMM-----------------
    xdata = df1.values[:,0:4]
    print(xdata.shape)
    N = xdata.shape[0]

    gm = GaussianMixture(n_components=3, max_iter=20)
    gm.fit(xdata)
    print(gm.means_)
    print(gm.weights_)
    print(gm.covariances_)
    z = gm.score(xdata)
    print(z)

    #-----compute accuracy--------
    preds = gm.predict(xdata)
    print(preds)
```

```
    cluster_assigned = []
    # since GMM is unsupervised, class assignments to clusters may vary on each run
    cluster_assigned = [mode(preds[0:50])[0], mode(preds[50:100])[0], mode(preds[100:150])[0]]
    acc = 0
    for i in range(0,N):
        if preds[i] == cluster_assigned[0] and i < 50: # first 50 members belong to class 0
            acc = acc + 1
        if preds[i] == cluster_assigned[1] and i >=50 and i < 100: # next 50 are class 1
            acc = acc + 1
        if preds[i] == cluster_assigned[2] and i >= 100 and i < 150: # last 50, class 2
            acc = acc + 1

    print('accuracy =',acc/N*100)

if __name__ == "__main__":
    sys.exit(int(main() or 0))
```

Set the GMM_SKL.py as the start file, and run the program. Your results should be similar in accuracy as your own code.

```
C:\Windows\system32\cmd.exe

 [[0.38744093 0.09223276 0.30244302 0.06087397]
  [0.09223276 0.11040914 0.08385112 0.05574334]
  [0.30244302 0.08385112 0.32589574 0.07276776]
  [0.06087397 0.05574334 0.07276776 0.08484505]]

 [[0.2755171  0.09662295 0.18547072 0.05478901]
  [0.09662295 0.09255152 0.09103431 0.04299899]
  [0.18547072 0.09103431 0.20235849 0.06171383]
  [0.05478901 0.04299899 0.06171383 0.03233775]]]
-1.20672096042722
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 2 1 2
 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1]
accuracy = 96.66666666666667
Press any key to continue . . .
```

Problem #2: Test the GMM algorithm (your code version) on the “Car Evaluation Dataset”. You can download the dataset from:

[UCI Machine Learning Repository: Car Evaluation Data Set]()

Note: You will need to convert the text column values to numerical values.