# Exercise 1

1. Expectation Maximization Clustering Write a script that implements the Expectation-Maximization (EM) algorithm for clustering (see Algorithm 13.3 in Chapter 13). Run the code on the iris.txt dataset. Use the first four attributes for clustering, and use the labels only for the purity-based clustering evaluation (see below). In your implementation, you should estimate the full covariance matrix for each cluster.

For EM initialization, use the first n/k points for cluster 1, the next n/k for cluster 2, and so on. For convergence testing, you can compare the sum of the euclidean distance between the old means and the new means over the k clusters. If this distance is less than $\epsilon$=0.001 you can stop the method.

Your program output should consist of the following information:

The final mean for each cluster The final covariance matrix for each cluster Number of iterations the EM algorithm took to converge. Final cluster assignment of all the points, where each point will be assigned to the cluster that yields the highest probability P(Ci|xj) Final size of each cluster Finally, you must compute the 'purity score' for your clustering, computed as follows: Assume that Ci denotes the set of points assigned to cluster i by the EM algorithm, and let Ti denote the true assignments of the points based on the last attribute. Purity score is defined as:

Purity=1n∑i=1kmaxkj=1{Ci∩Tj}

# Original implementation of the code was done by McDickenson available here - https://github.com/mcdickenson/em-gaussian (https://github.com/mcdickenson/em-gaussian)

considering two gaussian mixture model. This code modified the original work by extending to three gaussian mixture and shows a way of how to use the same code for n number of gaussian mixtures '''

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import norm
import random as rand
from sys import maxsize
from sklearn import datasets

print('Import Complete')
```

```
Import Complete
```

In [2]:
```
iris = datasets.load_iris()
X = iris.data[:, :4]  # we only take the first four features.
Y = iris.target
Y[:] = Y+1; #labelling the data in (1,2,3)

data = {'x': X[:,0], 'y': X[:,1], 'label' : Y}
df = pd.DataFrame(data=data)

#Y.tolist()

# inspect the data
df.head()
df.tail()

## Initialize three random index
k1 = rand.randrange(len(X));
k2 = rand.randrange(len(X));
k3 = rand.randrange(len(X))

## make intial guess using the three choosen random index
guess = { 'mu1': [X[k1,0],X[k1,1]],
          'sig1': [ [1, 0], [0, 1] ],
          'mu2': [X[k2,0],X[k2,1]],
          'sig2': [ [2, 0], [0, 1] ],
          'mu3': [X[k3,0],X[k3,1]],
          'sig3': [ [0.5, 0], [0, 1] ],
          'lambda': [0.3, 0.3, 0.4]
        }

#  lambda is the probablility that the point comes from that particular gaussian
# note that the covariance must be diagonal for this to work

# Probability of data point Val belonging to a cluster
def prob(val, mu, sig, lam):
    p = lam
    for i in range(len(val)):
        p *= norm.pdf(val[i], mu[i], sig[i][i])
    return p

# Expectation step - checking to which cluster the data point is expected to be cam
e from given the initial parameter setting
def expectation(dataFrame, parameters):
    for i in range(dataFrame.shape[0]):
        x = dataFrame['x'][i]
        y = dataFrame['y'][i]
    #assigning the probablilities of each cluster
        p_cluster1 = prob([x, y], list(parameters['mu1']), list(parameters['sig1
']), parameters['lambda'][0])
        p_cluster2 = prob([x, y], list(parameters['mu2']), list(parameters['sig2
']), parameters['lambda'][1])
        p_cluster3 = prob([x, y], list(parameters['mu3']), list(parameters['sig3
']), parameters['lambda'][2] )
    # Labelling each data according to the probabilities of cluster
        if (p_cluster1 >= p_cluster2) & (p_cluster1 >= p_cluster3):
            dataFrame['label'][i] = 1
        elif  (p_cluster2 >= p_cluster1) & (p_cluster2 >= p_cluster3):
            dataFrame['label'][i] = 2
        elif  (p_cluster3 >= p_cluster1) & (p_cluster3 >= p_cluster2):
            dataFrame['label'][i] = 3
        else: dataFrame['label'][i] = np.random.choice(3, len(df))+1

    return dataFrame
```

```
   mu1    sig1  mu2    sig2  mu3      sig3 lambda
0  6.5  [1, 0]  6.1  [2, 0]  5.2  [0.5, 0]    0.3
1  3.2  [0, 1]  2.6  [0, 1]  2.7    [0, 1]    0.3
2  NaN     NaN  NaN     NaN  NaN       NaN    0.4

C:\Users\jfrui\Anaconda3\lib\site-packages\ipykernel_launcher.py:55: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
C:\Users\jfrui\Anaconda3\lib\site-packages\ipykernel_launcher.py:51: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
```

In [16]:
```
from sklearn import mixture
import seaborn as sns
from sklearn.cluster import KMeans
```

In [11]:
```
iris = sns.load_dataset("iris")

iris.head()
```

Out[11]:

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | setosa |

In [12]:
```
kmeans = KMeans(n_clusters=3)
pred = kmeans.fit_predict(X)
```

In [13]:
```
kmeans_iris = KMeans(n_clusters=3)
pred_kmeans_iris = kmeans_iris.fit_predict(iris[['sepal_length','sepal_width','peta
l_length','petal_width']])
iris['kmeans_pred'] = pred_kmeans_iris
```

In [14]:
```
# Import adjusted rand score
from sklearn import metrics

# calculate adjusted rand score passing in the original labels and the kmeans predi
cted labels
iris_kmeans_score = metrics.adjusted_rand_score(iris['species'],iris['kmeans_pred
'])

# Print the score
iris_kmeans_score
```

Out[14]: 0.7302382722834697

In [17]:
```
# fitting the gmm to the Iris dataset
gmm_iris = mixture.GaussianMixture(n_components=3).fit(iris[['sepal_length','sepal_
width','petal_length','petal_width']])

#predicting the clustering labels from the dataset
pred_gmm_iris = gmm_iris.predict(iris[['sepal_length','sepal_width','petal_length
','petal_width']])

iris['gmm_pred']=pred_gmm_iris
```

In [18]:
```
iris['gmm_pred'] = pred_gmm_iris

# calculate adjusted rand score passing in the original
# labels and the GMM predicted labels iris['species']
iris_gmm_score = metrics.adjusted_rand_score(iris['species'],pred_gmm_iris)

# Print the score
iris_gmm_score
```

Out[18]:
```
0.9038742317748124
```

As you can see, using kmeans clustering results in a purity score of 73, while GMM results in a purity score of 90.

In [ ]:
```

```