Artificial Intelligence: `Lab Assignment-6`

Group Members-
Abhishek Katara
Ujwal Tewari
Nikhil Thota
Group-7

March 1, 2018

1 Question-1

Generate 100 data points coming from the mixture of four gaussians in R2 with weightages pi = [0.4, 0.3, 0.2, 0.1], means = [(0,0), (4,4), (0,3), (4,0)]. The covariance matrices corresponding to the gaussians are as follows: C1 = [1 0.7; 0.7], C2 = [1 0.25; 0.25 2.5], C3 = [0.5 0.1; 0.1 1] and C4 = [0.25 0; 0 0.35].

1.1 Pseudo Code-

The k-means problem is solved using Lloyd's algorithm. The average complexity is given by O(knT), were n is the number of samples and T is the number of iteration.
algorithm : "auto", "full" or "elkan", default="auto"
K-means algorithm to use. The classical EM-style algorithm is "full". The "elkan" variation is more efficient by using the triangle inequality, but currently doesn't support sparse data. "auto" chooses "elkan" for dense data and "full" for sparse data.

$\mathbf{n}_c lusters : int, optional, default : 8$
The number of clusters to form as well as the number of centroids to generate.

1.2 Code and Implementation

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import mixture

# Define simple gaussian
def gauss_function(x, amp, x0, sigma):
    return amp * np.exp(-(x - x0) ** 2. / (2. * sigma ** 2.))
```

```
mean1 = [0, 0]
cov1 = [[1, 0.7],[0.7, 1]]
mean2 = [4, 4]
cov2 = [[1, 0.25],[0.25, 0.5]]
mean3 = [0, 3]
cov3 = [[0.5, 0.1],[0.1, 1]]
mean4 = [4, 0]
cov4 = [[0.25, 0],[0,0.35]]

# Generate sample from three gaussian distributions
x1, y1 = np.random.multivariate_normal(mean1, cov1, 80).T
x2, y2 = np.random.multivariate_normal(mean2, cov2, 60).T
x3, y3 = np.random.multivariate_normal(mean3, cov3, 40).T
x4, y4 = np.random.multivariate_normal(mean4, cov4, 20).T

x=np.append(x1,x2,axis=0)
x=np.append(x3,x4,axis=0)
y=np.append(y1,y2,axis=0)
y=np.append(y3,y4,axis=0)
x=x.reshape(-1,1)
#print(x)
clf=mixture.GaussianMixture(n_components=6,covariance_type="full",ini
clf.fit(x,y)
x_0=[]
x_1=[]
x_2=[]
x_3=[]
x_4=[]
x_5=[]
y_0=[]
y_1=[]
y_2=[]
y_3=[]
y_4=[]
y_5=[]
print(clf.predict(x))
print(clf.predict(x)[0])
#print(len(clf.predict(x)))
i=0
while(i<60):
                print(i)
                if(clf.predict(x)[i]==0):
                        print(clf.predict(x)[i])
                        x_0.append(x[i])
                        print(x_0)

                        y_0.append(y[i])
                        print(y_0)
                elif(clf.predict(x)[i]==1):
```

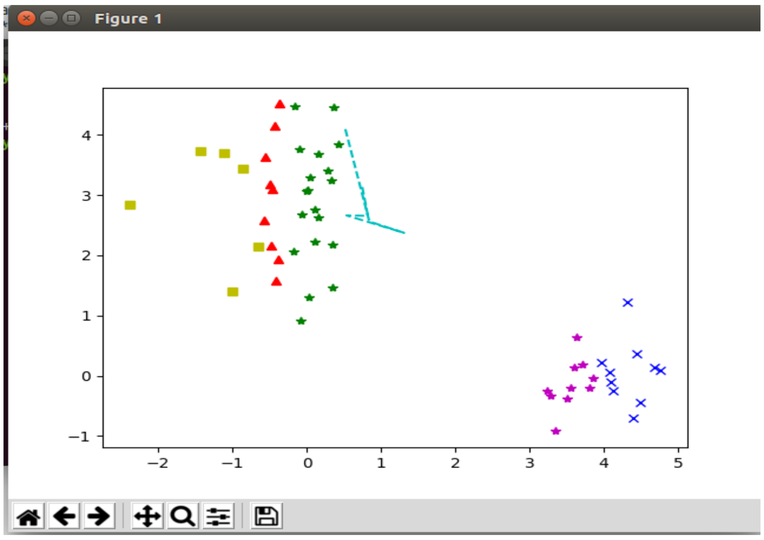

Figure 1: K-means clustering using weights

```
                        x_1.append(x[i])
                        y_1.append(y[i])
                elif(clf.predict(x)[i]==2):
                        x_2.append(x[i])
                        y_2.append(y[i])
                elif(clf.predict(x)[i]==3):
                        x_3.append(x[i])
                        y_3.append(y[i])
                elif(clf.predict(x)[i]==4):
                        x_4.append(x[i])
                        y_4.append(y[i])
                elif(clf.predict(x)[i]==5):
                        x_5.append(x[i])
                        y_5.append(y[i])
                i=i+1
print(x_0,y_0)
x_0=np.array(x_0)
y_0=np.array(y_0)

plt.plot(x_0,y_0,'r*')
plt.plot(x_1,y_1,'bx')
plt.plot(x_2,y_2,'gs')
plt.plot(x_3,y_3,'y^')
plt.plot(x_4,y_4,'m^')
plt.plot(x_5,y_5,'c^')


plt.show()
```

2 Question-2

Make a version of the K-means algorithm that models the data as a mixture of K arbitrary Gaussians, i.e., Gaussians that are not constrained to be axis-aligned. Used the developed algorithm to cluster the data points generated in part 1 with K = 1, 2, 3, 4, 5, 6.

2.1 Pseudo Code-

Gaussian Mixture model has been used to remove the hardcode centroids of K measn and transforming it into soft k means
The GaussianMixture object implements the expectation-maximization (EM) algorithm for fitting mixture-of-Gaussian models. It can also draw confidence ellipsoids for multivariate models, and compute the Bayesian Information Criterion to assess the number of clusters in the data. A GaussianMixture.fit method is provided that learns a Gaussian Mixture Model from train data. Given test data, it can assign to each sample the Gaussian it mostly probably belong to using the GaussianMixture.predict method.

The GaussianMixture object implements the expectation-maximization (EM) algorithm for fitting mixture-of-Gaussian models. It can also draw confidence ellipsoids for multivariate models, and compute the Bayesian Information Criterion to assess the number of clusters in the data. A GaussianMixture.fit method is provided that learns a Gaussian Mixture Model from train data. Given test data, it can assign to each sample the Gaussian it mostly probably belong to using the GaussianMixture.predict method.

The GaussianMixture comes with different options to constrain the covariance of the difference classes estimated: spherical, diagonal, tied or full covariance.

2.2 Code and Implementation

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import mixture

# Define simple gaussian
def gauss_function(x, amp, x0, sigma):
    return amp * np.exp(-(x - x0) ** 2. / (2. * sigma ** 2.))

mean1 = [0, 0]
cov1 = [[1, 0.7],[0.7, 1]]
mean2 = [4, 4]
cov2 = [[1, 0.25],[0.25, 0.5]]
mean3 = [0, 3]
cov3 = [[0.5, 0.1],[0.1, 1]]
```

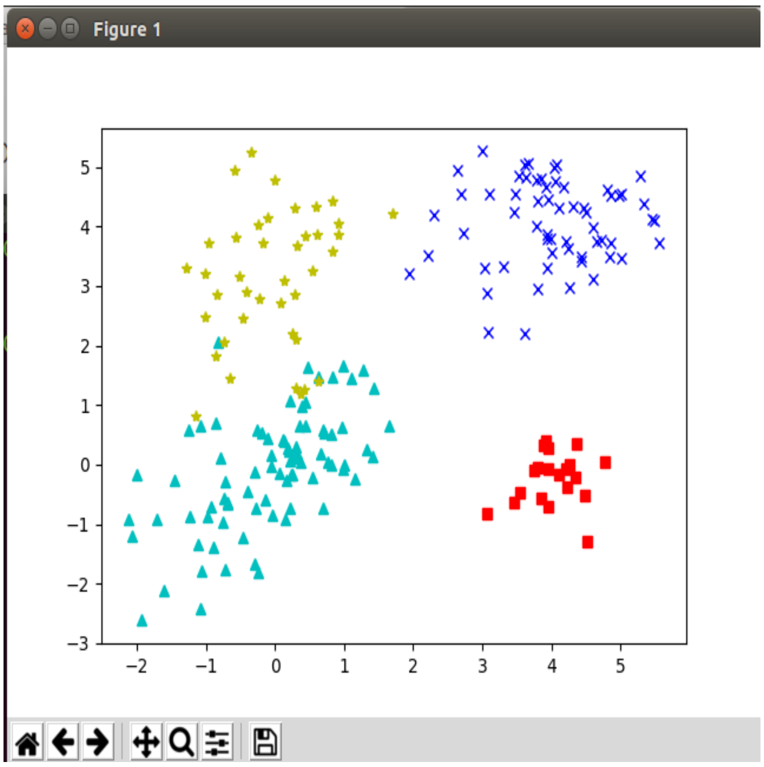

Figure 2: Gaussian mixture model

```
mean4 = [4, 0]
cov4 = [[0.25, 0],[0,0.35]]

# Generate sample from three gaussian distributions
x1, y1 = np.random.multivariate_normal(mean1, cov1, 80).T
x2, y2 = np.random.multivariate_normal(mean2, cov2, 60).T
x3, y3 = np.random.multivariate_normal(mean3, cov3, 40).T
x4, y4 = np.random.multivariate_normal(mean4, cov4, 20).T

x=np.append(x1,x2,axis=0)
x=np.append(x3,x4,axis=0)
y=np.append(y1,y2,axis=0)
y=np.append(y3,y4,axis=0)
plt.show()
```