Programming Assignment 2: Clustering

Instructor: Dr. Satish Kumar Thittamaranahalli
Group Member: Yu Hou; Haoteng Tang

In this assignment, we implemented the K-means algorithm and Expectation Maximization algorithm for clustering using a Gaussian Mixture Model (GMM) with Python and Matlab. The original data contains 150 2D points. We read these points in an array as [x, y]. In the first section, we will introduce the pseudocode of K-means algorithm and EM algorithm. In the second section, we will introduce the k-means method in Weka software and EM algorithm in Scikit learn with Python and other Clustering library in Matlab. In the third section, we will introduce you some interesting application using clustering.

1 pseudocode of clustering

(1) pseudocode
Clustering is unsupervised learning, we cannot get the label before we learn the knowledge for the data. Clustering helps us divided the data into different clusters. The first method is called K-means. The pseudocode of K-means is shown as follows:

Pseudocode:

K-means function:

```
Get the initial case. Using a random number to choose k centroids. (k is the number of clusters.)
Get the distances of different points to the centroids. The points will belong to the clusters which the distances of themselves to the centroids are the smallest separately.
Get the total distance of all points to their own centroids.

While distance is not the smallest:          //did not stop until finding the smallest.
    Get the mean value of different clusters.
    According to different mean, calculate the distance of every point to the mean again, put them into the new clusters.
    Calculate summation of every point to their own new centroid of their cluster.
    Get the smallest.
When the while loop stops. Every point gets into the appropriate clusters.
```

EM method (Gaussian function):

```
Get the initial case. Using random number to give each point a ric (i means the point, from 1 to N. c means the number of clusters, from 1 to k)
Call miu, get miu [miu1, miu2, miu3, ……, miuk]
Call sigma, get sigma [sigma1, sigma2, ……, sigmak]
Call pi, get pi [pi1, pi2, ……, pik]

While likelihood is the greatest:        //did not stop until finding the greatest likelihood
    Get the likelihood.
    Get the new ric array.
    Based on the new ric, get the new miu [miu1, miu2, miu3, ……, miuk]
```

Based on the new ric, get the new sigma [sigema1, sigema2, ……, sigemak]
Based on the new ric, get the new pi [pi1, pi2, ……, pik]
When the while loop stops, the likelihood is the greatest. And the now is the best clusters.

(2) The result.
Because we set the initial case randomly, so the results are also different based on the initial cases. As for the K-means method, the initial case can be shown in figure 1 and after the iterations, the results can be shown in the figure 2.

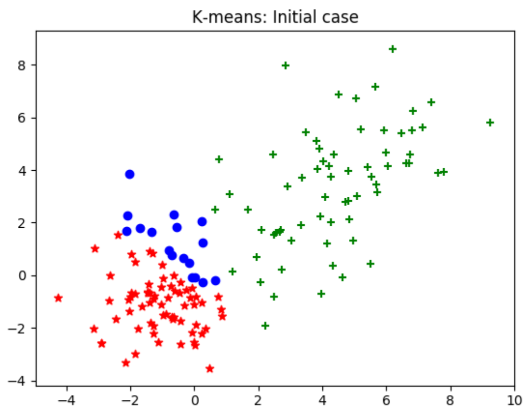


Figure 1 K-means: initial case

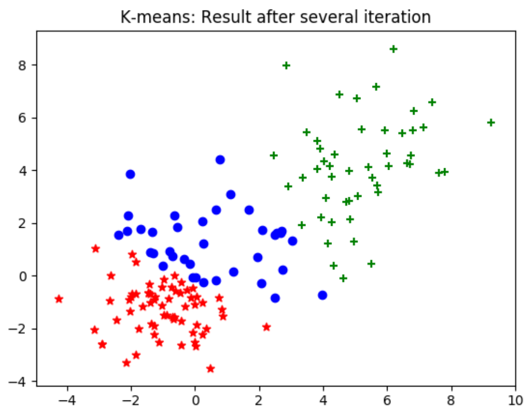


Figure 2 k-means: after several iterations

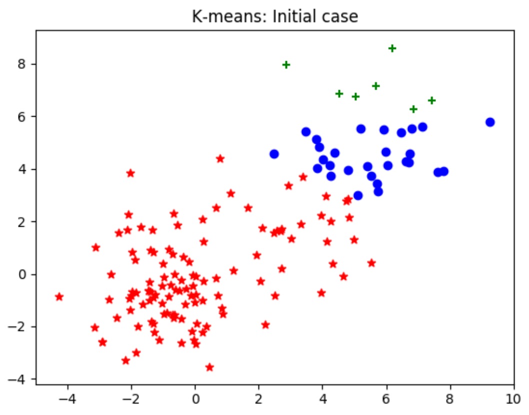


Figure 3 K-means: initial case II

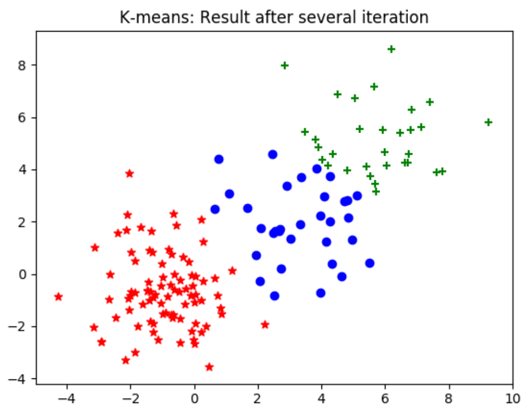


figure 4 K-means: after several iterations II

One of the results can be reported as follows.

The distance of this case 245.1996942792362
The centroid of each cluster
The centroid of cluster 1:
[5.738495346032257, 5.164838081096775]
The centroid of cluster 2:
[-0.9606529070232559, -0.6522184128604652]
The centroid of cluster 3:
[3.2888485605151514, 1.9326883657575762]

According to the figure1 and figure2, we can find that the green cluster was big but after several iterations, this cluster got smaller, which made the clusters evenly. But in the initial case, the green cluster not always was big. According to the figure3 and figure4, we could find that the green cluster was small, but after several iterations, this cluster is a little bit bigger. But we could find that the clustering result would depend on the initial case. In figure5, this figure will help you understand why we cannot get the same result.

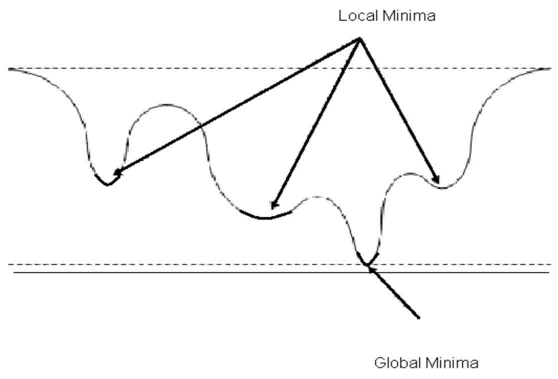


Figure5 local minimum and global minimum

Because this is a local minima issue, we can only get the initial case randomly, so that we just can get the local minima. One solution is to try another other initial case several times till we find the global minima case. As for the k-means method, we can run the program 100000 times. That means we will random find the initial case 100000 times so that we can find the minima case more precisely. When we running 100000 times, the result is as follows:

The minimal distance = 243.4417053864569
The minimal case's centroids:
[[-1.000687768928572, -0.6940532127261908],
[5.572892383999999, 4.903074262194447],
[2.9080951661999994, 1.6529236502666669]]

As for the EM algorithm using a Gaussian Mixture Model, the result can be shown as figure6 – figure9. Same as the k-means method, the result will depend on the initial case. After several running, we could find two typical results. Shown as figure7, this case usually will be gotten after 200 times iterations and the likelihood of this case usually is about 113.80. The figure9 showed another typical case. This case usually will be gotten after 545 times iterations and the likelihood of this case usually is about 114.40.

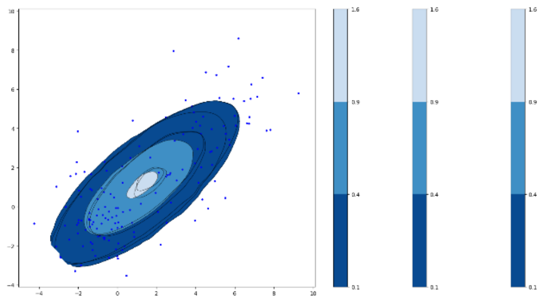


Figure 6 GMM: initial case I

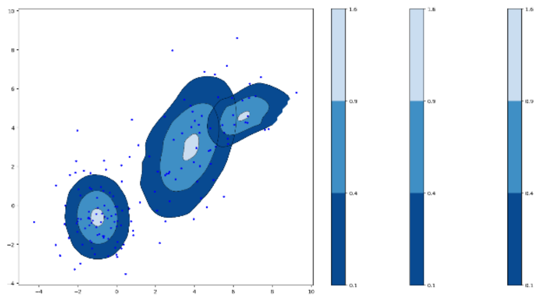


figure 7 GMM: after several iterations I

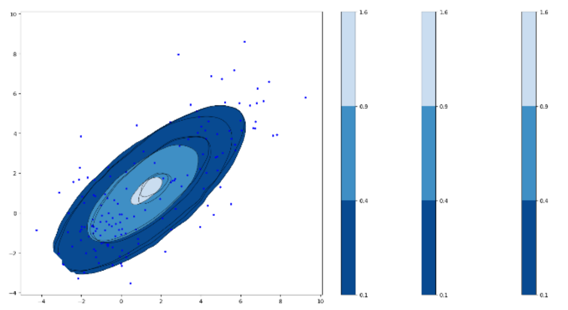


Figure 8 GMM: initial case II

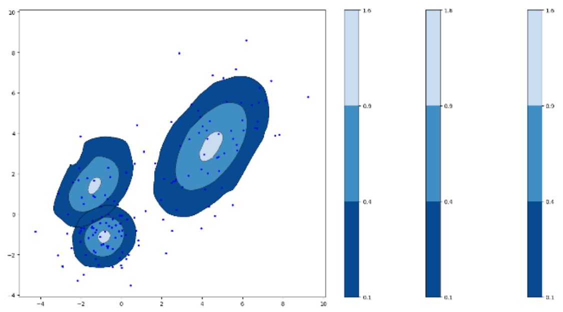


figure 9 GMM: after several iterations II

One of the GMM results can be shown as follows.

The likelihood of this case: 114.40249631075338
The mean of each Gaussian:
The first one: [4.4540446869505885, 3.354308978954069] The second one: [-0.84519845079216038, -1.1226482548646946] The third one: [-1.3448194730311778, 1.380236102503211]
The amplitude of each Gaussian:
The first one: 64.21838208 The second one: 66.327182893 The third one: 19.4544350271
The covariance matrix of each Gaussian:
The first one: [[3.44045143 2.29943619]
[2.29943619 5.17207772]]
The second one: [[1.05766816 0.1395411]
[0.1395411   1.02259829]]
The third one: [[1.60466632 0.74371824]
[0.74371824 1.74882223]]

Also, we could run this programming more times to get the result more precisely. As for the GMM case, we would run this programming 1000 times and get the mu, sigma, likelihood as follows.

The greatest likelihood = 115.70346557850243
The best case's mu = [Cluster 1 [-0.90489619964739842, -0.0499953319247944439],
Cluster 2 [4.2391403567677877, 3.2896031627755877],
Cluster 3 [-1.1407378512975259, -1.303868463195905]]

The best case's sigma = [Cluster1 ([[0.82882047, -0.67205081],
[-0.67205081, 2.29776972]]),
Cluster 2 ([[4.09561903, 2.45641688],
[2.45641688, 5.08475363]]),
Cluster 3 ([[1.45789339, 0.21867049],
[0.21867049, 0.80696156]])]

(3) data structures, challenges, and optimizations
In the k-means method, we used the list to store the 2D point. [x1, y1] represents the x value and y value of one record. we used a dictionary (key-value) to store data belonging to different clusters. Also, we store the centroid of each cluster into a list.
In GMM method, we used the list to store the 2D point and its contribute to each Gaussian cluster. For

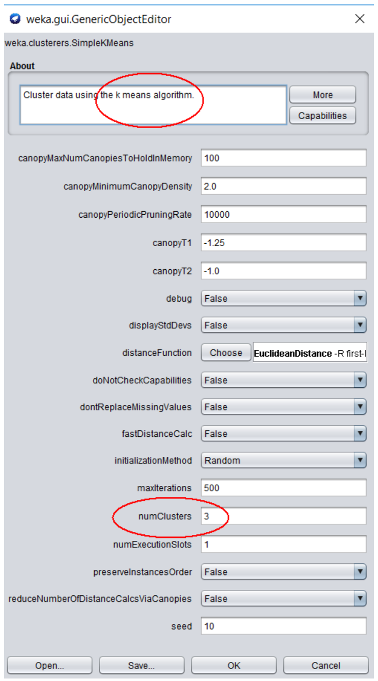
example, one record [X1, Y1, r11, r12, r13] represented that this was the first record, x value and y value were x1, y1 and r11 mean this point had r11 probability belonging to Gaussian cluster 1. The means, amplitude and covariance matrix of each Gaussian were stored into a different list.
One challenge we met was that sometimes the program would crash because of the invalid math calculation. We found that when we calculate the centroid of the initial case. If we chose the random number as kList = [13,119,119]. We had two 119, so that some points would go to the first 119, and the second 119 was empty. But in our code, we would put the number of points in a cluster as the denominator. This will crash the programming. So if we met such situation, we should star the initial case again.
Another challenge was that when we calculated the covariance matrix, we would be lost about some math equations. We tested our code many times to ensure the correctness.
As for the optimizations, because we tried to run our program more than one time. So when we tried to run the GMM method for 1000 times, it would cost us 15 hours. We found that we could store our result in our code and used the stored result to do the next step so that can reduce the time-consuming. For example, we could store the result of probability distribution function ahead of time. When we calculated the ric, we could directly use the result. After this optimizations, when we run the GMM for 1000 times, it cost us 10 hours.

(4) comparison of K-means and GMM
First of all, the most of the obvious difference is that the time consuming are different. Even though we run the k-means for 10000 times, it will finish in 10 minuses. However, we run the GMM for 1000 times, it will cost us 10 hours.
Secondly, the thoughts are different. The k-means is easier to understand. It uses the centroid thoughts. But the drawback of this method is it is very sensible to abnormal data. The GMM uses the Gaussian thoughts and the probabilities thoughts.
Finally, they have the same drawback is that we should set the number of clusters ahead of time.
Nowadays, we also can improve these two methods in many different ways.

<u>2 Software Familiarization</u>

In this case, we will introduce the software which is used to implement the k-means and GMM method. Such as Weka, Scikit learn, and library in Matlab.

(1) Weka for K-means and EM algorithm
Weka is a free software for machine learning. It is implemented by JAVA. Figure10 shows you how to set the information like the number of clusters and other settings. And the figure11 shows us the result of the simpleKmeans. Also, we can use the EM algorithm. Figure12 shows you how to set the information like the number of clusters and minimal likelihood and other settings. And the figure13 show you the result of EM algorithm.


weka.gui.GenericObjectEditor

weka.clusterers.SimpleKMeans

About

Cluster data using the k means algorithm.

canopyMaxNumCanopiesToHoldInMemory	100
canopyMinimumCanopyDensity	2.0
canopyPeriodicPruningRate	10000
canopyT1	-1.25
canopyT2	-1.0
debug	False
displayStdDevs	False
distanceFunction	EuclideanDistance -R first-l
doNotCheckCapabilities	False
dontReplaceMissingValues	False
fastDistanceCalc	False
initializationMethod	Random
maxIterations	500
numClusters	3
numExecutionSlots	1
preserveInstancesOrder	False
reduceNumberOfDistanceCalcsViaCanopies	False
seed	10

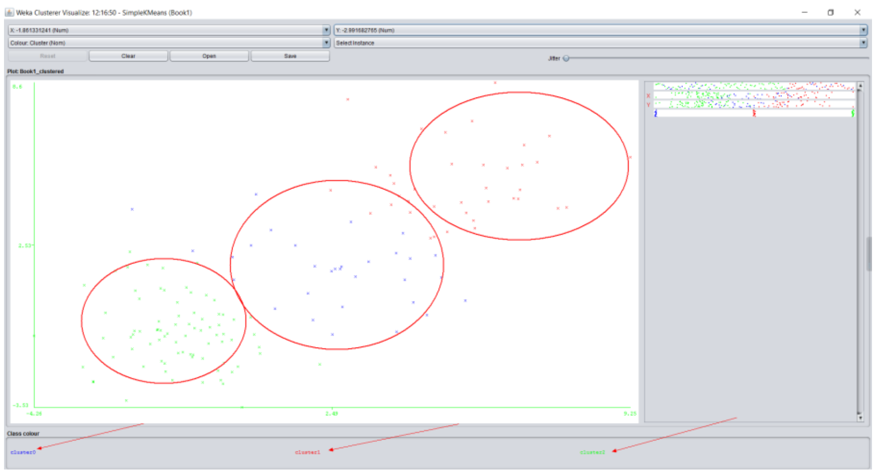
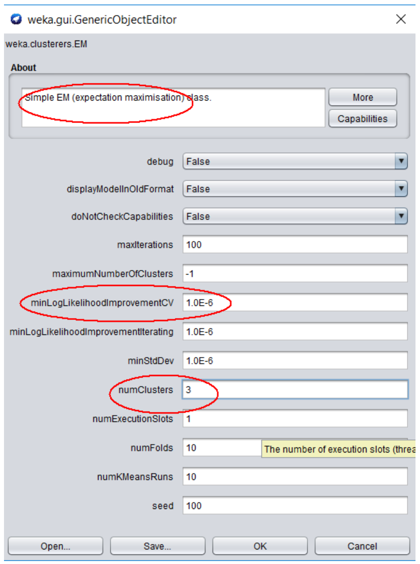


Figure 10 setting for K-means Figure 11 result of k-means


weka.gui.GenericObjectEditor

weka.clusterers.EM

About

Simple EM (expectation maximisation) class.

debug	False
displayModelInOldFormat	False
doNotCheckCapabilities	False
maxIterations	100
maximumNumberOfClusters	-1
minLogLikelihoodImprovementCV	1.0E-6
minLogLikelihoodImprovementIterating	1.0E-6
minStdDev	1.0E-6
numClusters	3
numExecutionSlots	1
numFolds	10
numKMeansRuns	10
seed	100

The number of execution slots (threa

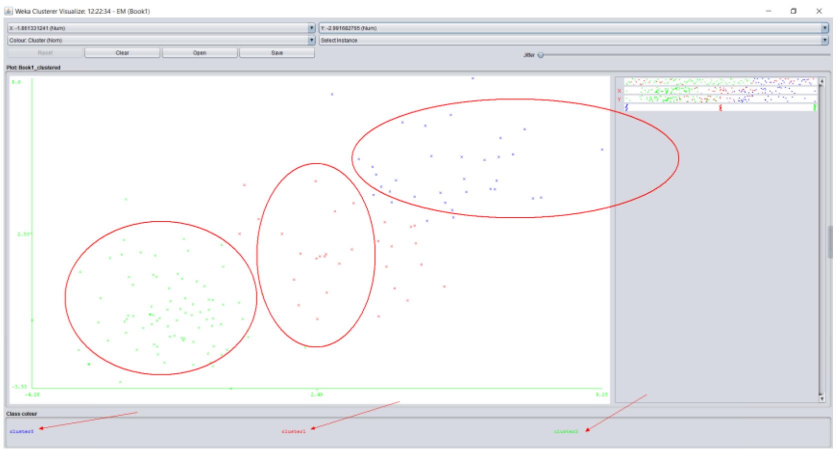


Figure 12 setting for EM Figure 13 result of EM

(2) Python: Sklearn for K-means and GMM

Scikit learn is a library for users to implement the machine learning algorithm. As for k-means methods, we can use the class – cluster.KMeans in this library. The statements look like follows.

from sklearn.cluster import KMeans

Kmeans = KMeans(n_clusters=3, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='auto', verbose=0, random_state=None, copy_x=True, n_jobs=1, algorithm='auto')

In this statement, n_clusters is the number of clusters. The max_iter is a maximum number of

iterations. Also, there are other settings.
As for GMM methods. We can use sklearn.mixture. This is a package which helps us implement the Gaussian Mixture Models. Here are it's statements.

GaussianMixture(n_components=1, covariance_type='full', tol=0.001, reg_covar=1e-06, max_iter=100, n_init=1, init_params='kmeans', weights_init=None, means_init=None, precisions_init=None, random_state=None, warm_start=False, verbose=0, verbose_interval=10)

In this statement. init_params is the method used to initialize the weights. You can choose "Kmeans" or "random". This setting is very useful.

(3) MATLAB: for K-means and EM algorithm
As for K-means algorithm, we can directly use:

'idx=kmeans(X,3);   %k mean cluster'to make a k-means cluster for data X.

And as for EM algorithm, we can directly use:

'GMModel=fitgmdist(X,3);
idx2=cluster(GMModel,X);' to make a EM algorithm .

So the K-means & the EM algorithm based on MATLAB library function can be showed as follows.

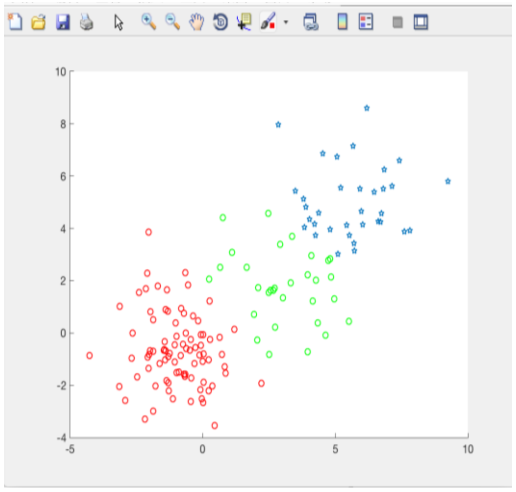


Figure 14 K-means result

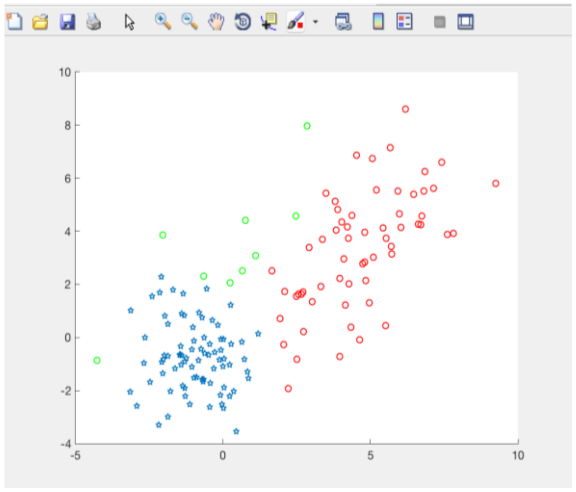


Figure 15 EM results

<u>3 Application</u>

Clustering algorithms are used in a lot of fields such as commerce, biology, IT, geology and insurance etc. One of the most popular applications that I want to state is that they are always used in the field of neuroscience. For instance, one of the research that I have dealt with is that analyzing the pattern of the abnormal white matter in the people's brain who has schizophrenia. The clustering algorithm was used in this research by clustering a set of DTI data from MRI associated with different patients. However, we did not know the exact cluster number before we cluster (actually, the exact number is what we want to know). So the first step, I set up the K-means clustering algorithm to cluster the data. The second step, I set three relative parameters to evaluate the result of each cluster under different cluster number. And the optimization result came out when the cluster number is two, which means that there are two different patterns of white matter in schizophrenia patient. And this result is identical to the result from the anatomy analysis.