Homework 2: K-Means and GMM

Zepei Zhao (5635405542)
Mingjun Liu (1321657359)

Part One

1. Algorithm

This homework contains two unsupervised algorithms, K-means and Gaussian Mixture Model (GMM). These two clustering model work on dividing n points to k clusters. Among them, K-means will randomly generate K initial centroids, and then allocate each point to the nearest initial centroid. Next, recalculate the centroids for k clusters. Repeat step 2 and 3 to find the best result. GMM can be regarded as the extension of K-means, considering the mean of data distribution and also covariance. In GMM, data is assumed as a combination of several random variables with Gaussian distribution and we need to estimate mean and covariance of each Gaussian distribution. Meanwhile, latent variable is introduced to calculate the probability that each sample belongs to each cluster. To get the parameter of GMM model, Expectation-Maximization algorithm is applied for find the best latent variable. There is two steps for EM. First, assign latent variable randomly. Then figure out mean, covariance. Next, recompute latent variable. We need to repeat these two step until reach iteration terminal condition.

2. Data Structure

In the algorithm, array and list are used to store data, so that we could utilize its index to access and modify data. Because we implement these two algorithms by using python. For the convenience of processing data, we applied numpy package. In K-means, we initially load data in list to visualize. Then we data is stored in array. To calculate the distance of each point and each cluster, or the step that assigns each point to closet centroid, we used [i] as index.

For GMM, when initialize parameter, we use numpy to generate initial array Pi,W, Mu and Var. Meanwhile, one-dimensional and two-dimensional array are employed.

3. Code Level Optimization

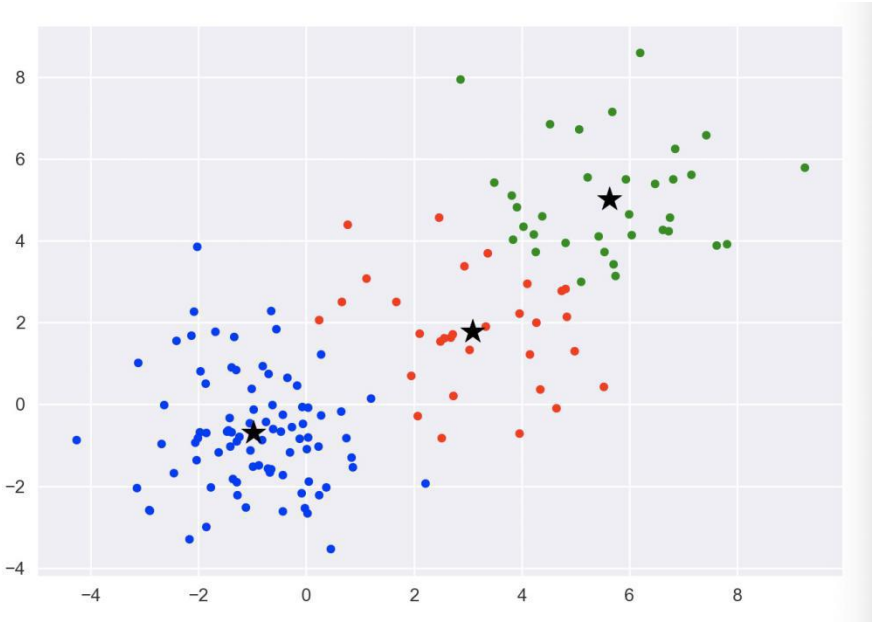
At begin, in the maximization step, we coded three functions to update Mu, Pi and Var. Then we realized it can be reduced and combined to one function, which is more concise. Besides, we use numpy package, which supports for dimensional array and matrix operations.

4. Challenge

First, we need to understand algorithm though of K-means, GMM and EM. The core of each step is required to learn. Second, the part of mathematics is also difficult for us, including turning basic math formula to code. Third, to get a better result, we have to set a small threshold, which causes the code runs many times to reach the threshold. That is quite cost a lot.

5. Result

K-means Result:



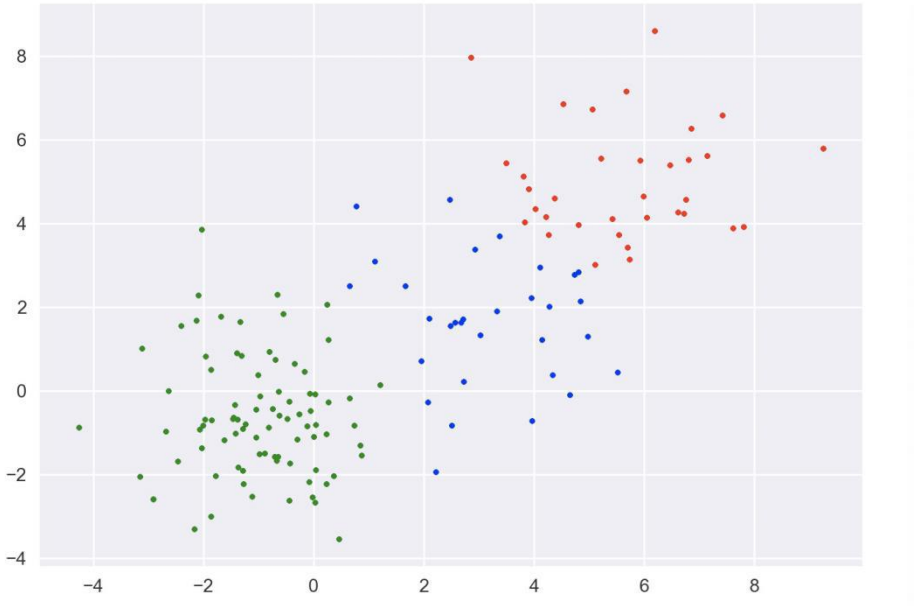
Centroids: [[3.0831826 1.7762138]

[5.620166 5.0262265]

[-0.9747657 -0.684193]]

Iteration times: 2

GMM Result:



Threshold: 1e-6

Iteration times: 34

Mu: [[5.58264914 4.95154469]

[-0.96209684 -0.63072733]

[3.24019432 1.84353861]]

Var: [[2.21578444 2.11330506]

[1.23363309 2.03449848]

[1.87553873 2.519102]]

Pi: [0.22148531 0.57118498 0.20732971]

Part 2: Software Familiarization

1. Library function

Python library Scikit-learn provides both KMeans and Gaussian Mixture Model (GMM) for data clustering. The KMeans module is called sklearn.cluster.KMeans(), which requires users to classify the number of clusters (n_clusters) and the data of size (number of sample * dimension). The model is able to assign the given data points to the different clusters (labels_), return the centers of the clusters (cluster_centers_), number of iterations run (n_iter_) and sum of squared distances of samples to their closet cluster center (inertia_). Also, the model is able to predict which cluster is a point belonging to. In this homework, we read the cluters.txt file as numpy array, set the maximum iteration to 100 (max_iter = 100), and let the program run 10 times the k-means algorithm (n_init = 10), then return the best output of trial.

The GMM module is called sklearn.mixture.GaussianMixture(). This module also requires users to input the number of clusters (n_clusters) and the data of size (number of samples * dimension). This program is able to return the weights of each mixture component (Pi in our program), mean of each mixture component (Mu in our program), covariance of each mixture component (Var in our program), number of iterations run and the lower bound value on the log-likelihood of the best fit of EM. Similar to KMeans, the number of clusters is set to 3 (n_clusters = 3), and the same data is applied to the program.

2. Usage

KMeans:

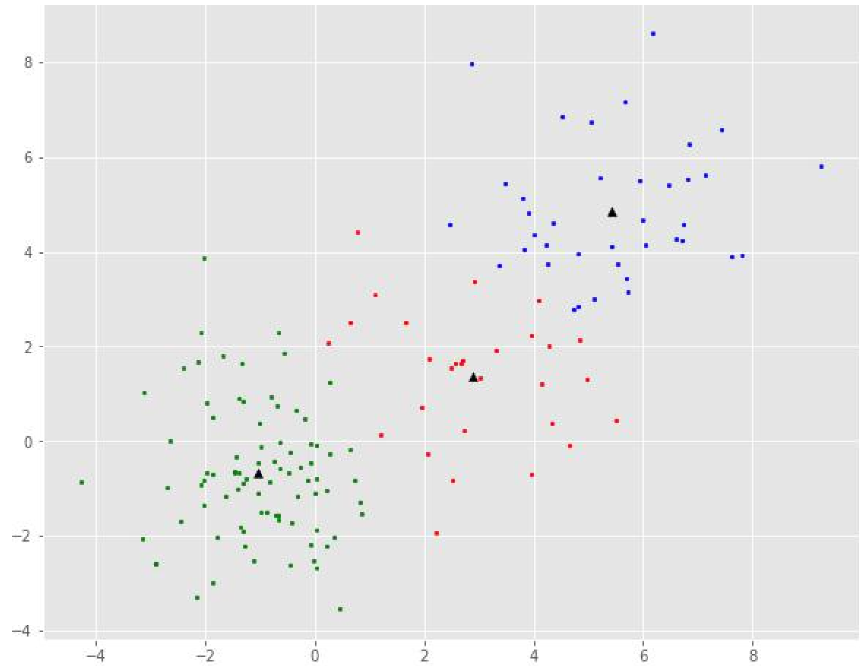
```
from sklearn.cluster import KMeans

km = KMeans(n_clusters = 3, n_init = 10, init = 'random', max_iter = 100)

y_km = km.fit_predict(data)
```

Results:

km.cluster_centers_ = [[-0.96065291, -0.65221841], [3.28884856, 1.93268837], [5.73849535, 5.16483808]]

km.labels_ = [0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 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, 0, 0, 0, 0, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 1, 1, 1, 2, 1, 1, 2, 2, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 1, 1, 2]

km.n_iter_ = 5



GMM:

from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components = 3)

y_gmm = gmm.fit_predict(data)

Results:

gmm.weights_ = [0.56497181, 0.26025477, 0.17477342]

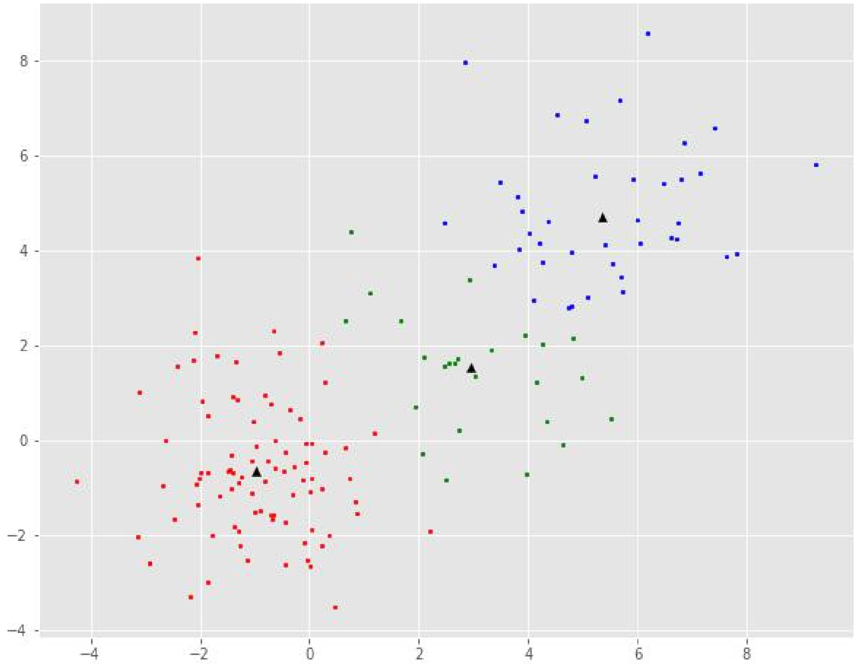
gmm.means_ = [[-0.97675717, -0.65978187], [5.35474763, 4.71166217], [2.95794243, 1.5172694]]

gmm.covariances_ = [[[1.21681206, -0.11996906], [-0.11996906, 1.97577318]], [[2.3640313 , 0.45069033], [0.45069033, 2.33636882]], [[2.05537151, -0.26590085], [-0.26590085,

2.08519189]]]

gmm.n_iter_ = 4

y_gmm = [0, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 2, 2, 2, 0, 2, 0, 0, 0, 2, 0, 0, 2, 0, 0, 2, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 2, 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, 0, 0, 0, 0, 0, 0, 0, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 2, 2, 1]



The result shows the same distribution on K-Means and GMMs. If we count the assignments on K-Means, we could find that the weights of each group are approximately equal between the two models. However, the GMM model has fewer iteration than K-Means.

3. Improvement

- Running time and iteration

Although the running time of both program for KMeans are near, the iteration of the library package is obviously smaller than the program we write. For GMMs, the iteration of the library package s only 4, but our program rises to 34. Also, the running time is therefore much larger. It takes 2.34931 seconds to show the result of our program, comparing to library packages which takes only 0.00999 seconds.

- No prediction automatically

The library packages come with prediction functions. However, if we need to predict other data points, we need to write the function ourselves.

- Parameter is preset

In our program, a lot of parameter is preset and unchangeable. The library packages are able to get the parameters of each functions by themselves.

Part 3: Applications

K-means:

K-means algorithm is usually applied to datasets with small and continuous dimensions and numerical values. It is used in many different scenes, such as document classifier, which divides documents into different categories based on tags, topics and document content. Besides, in business activity, K-means is applied for customer classification, helping marketers improve their customer base and further segment customer categories based on customer purchase history, interests or activity monitoring. Also, for telecommunication companies, the company collects information about users' calls, text messages and network activities, then use K-means to cluster customers' activities 24 hours a day to understand the usage of customers within hours.

GMM:

GMM is a basic and widely used model, which has application in image segmentation, object recognition, video analysis, etc. For example, it's a good algorithm to classify pulsar and calculate the likelihood of unidentified \textit{Fermi} point sources being pulsars and rank them accordingly.

Part 4: Individual contributes

Both Zepei and Mingjun worked for the K-Means algorithm and visualization. In the GMM model, Zepei is responsible for developing the basic algorithm, and Mingjun is responsible for the Gaussian probability and iterations.

Reference:

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html#sklearn.mixture.GaussianMixture

https://arxiv.org/abs/1205.6221