K-means and GMM Implementation and Applications

Chaoyu Li, chaoyuli@usc.edu, 6641732094

DSCI-552: Machine Learning for Data Science

Instructor: Satish Kumar Thittamaranahalli

Feb. 9, 2022

Table of Contents

1 Basic Algorithm and Code Functions Description

1.1 Implementation of K-means

1.1.1 Basic Algorithm of K-means

A. According to the required number of clusters K, I randomly set K initial centroids.

B. I calculated the distance for each data point and label the cluster, then calculated the new centroids.

C. I calculated the total distance for each new centroid with its own group of data points and compare it with the old distance.

D. This algorithm will be terminal if there is no improvement in the total distance. Otherwise, repeat B~D.

1.1.2 Code Functions of K-means

- **K-means(k, dimension, means_list, n)**

It calculates the position of the three centroids of clusters using the algorithm described in 1.1.1. It accepts the number of clusters K, the dimension of one data, here is [x, y], a list of initial random mean values, and the number of data N. It returns an array representing the final position of the three centroids of the clusters, a dictionary of which point belongs to which cluster, a list representing how many points each cluster contains

1.2 Implementation of GMM

1.2.1 Basic Algorithm of GMM

A. I randomly picked up K points from data points as the initial value of mean for each cluster. Assume the initial covariance matrix is [[1,0], [0,1]] for 2 dimensions' data set for each cluster. Assume the initial weight for each Gaussian distribution is 1/K.

B. I calculated the Probability under K Gaussian distribution for each data point.

C. I labeled the data point to the maximum probability of Gaussian distribution.

D. I calculated the new Gaussian distributions based on the new classification, then summarized the total probability of each Gaussian distribution.

E. This algorithm will be terminal if the likelihood has not a big change in one interaction. Otherwise, repeat B~D.

1.2.2 Code Functions of GMM

In all functions, "pointNumber" stands for the number of data of the dataset; "k" stands for the number of the clusters; "alpha" stands for the amplitude of one cluster; "mu" stands for the mean of one cluster; "sigma" stands for the covariance of one cluster.

- **normal(xI, muK, sigmaK, D)**

It stands for the formula: $N(x;\mu,\Sigma) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\Sigma|^{1/2}}e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$. Here xI stands for ith data point. muK stands for μ_k. sigmaK stands for σ_k. D stands for the dimension of data point.

- **maximizeL(pointNumber, k, alpha, points, mu, sigma)**

This function simulates Maximum Log-Likelihood Estimation Function to compare the new clusters with the old ones.

- **Estep(pointNumber, k, W, alpha, points, mu, sigma)**

This function simulates the Expectation-step in the EM-GMM algorithm. Here W refers to the weight, which is a hidden variable. This function will call normal() to calculate alpha[k]*normal(Xi, Uk, Sk) for each data point. Then, it summary them in all distributions.

- **Mstep(pointNumber, k, W, alpha, points, mu, sigma)**

This function simulates the Maximization-step in the EM-GMM algorithm. This function will calculate new Gaussian distributions to get mean, amplitude, and covariance matrix of each Gaussian distribution.

- **GMM(k, points, alpha, mu, sigma)**

It is the main function of GMM algorithm. It loops through all existing values to calculate likelihood, and stops if the new likelihood changes by less than my artificial threshold (5e-4) compared to the old likelihood.

2 Data Structure Used

2.1 Basic data structure

A. All means are stored in the data structure of List or NPArray. The format is:

[[centroid1X, centroid1Y], [centroid2X, centroid2Y],[centroid3X, centroid3Y]]

B. All data points are stored in the data structure of List or NPArry. The format is:

[[point1X, point1Y], [point2X, point2Y] …, [pointNX, pointNY]]

2.2 K-means data structure

A. Distance is stored in the data structure of NPArray. The format is:

[[Dis point1 to cluster1, Dis point1 to cluster2, Dis point1 to cluster3], [Dis point2 to cluster1, Dis point2 to cluster2, Dis point2 to cluster3], …, [Dis pointN to cluster1, Dis pointN to cluster2, Dis pointN to cluster3]]

B. Relationships between data points and clusters are stored in dictionaries. The format is:

{"Index of cluster1": [Index of data point, Index of data point…],
"Index of cluster2": [Index of data point, Index of data point…],
"Index of cluster3": [Index of data point, Index of data point…]}

2.3 GMM data structure

A. Amplitudes are stored in the data structure of the NPArray with length K. The format is:

[A/N, B/N, C/N](Here A,B,C represent the number of points of each cluster, N represent the number of data points in the dataset).

B. Means are stored in the data structure of the NPArray. The format is:

[[centroid1X, centroid1Y], [centroid2X, centroid2Y],[centroid3X, centroid3Y]]

C. Covariance matrix is stored in the data structure of the NPArray. The format is:

[[point1X, point1Y], [point2X, point2Y] …, [pointKX, pointKY]]

D. Posteriori probability of X_i for each cluster (W_{ik}) is stored in the data structure of the NPArray. The format is:

[[[W_1 in cluster 1], [W_2 in cluster 1],…, [W_n in cluster 1]],

[[W_1 in cluster 2], [W_2 in cluster 2],…, [W_n in cluster 2]],

…

[[W_1 in cluster K], [W_2 in cluster K],…, [W_n in cluster K]]]

3 Challenges Faced and Optimizations

3.1 Challenges

A. Getting an initialization centroid for each cluster randomly is not the best choice for K-means because I ran this algorithm many times the initialization centroid will significantly affect the result.

B. How to set the initial value of parameters of EM-GMM is a big challenge because a bad initialization will lead to a bad result.

3.2 Optimizations

A. I ran my K-means program many times and compare the result with the result getting from Sklearn-K-means to evaluate the quality of the result. Then, I chose the best one to be the initial value of parameters of EM-GMM algorithm to guaranteed to get better predictions.

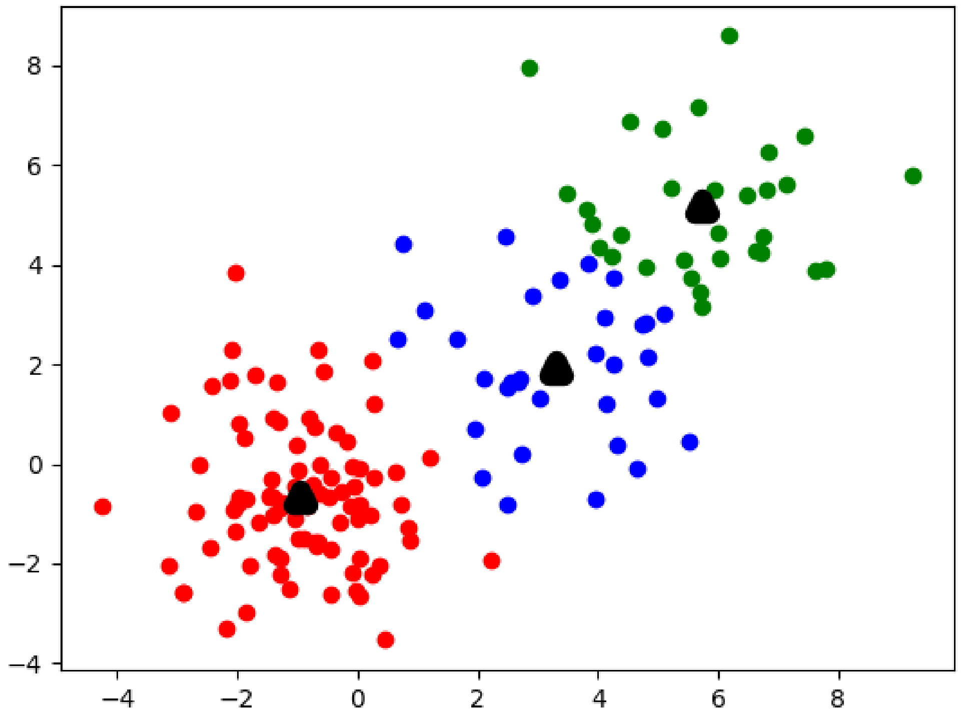
4 Execution of my algorithm and results

4.1 Execution of K-means program

- Print of centroids:

```
K-Means centroid is
 [[ 5.73849535  5.16483808]
 [ 3.28884856  1.93268837]
 [-0.96065291 -0.65221841]]
```

- Image of clusters:



4.2 Execution of GMM program

- Print of result:

```
The amplitudes are:
[0.22596436 0.2080321  0.56600354]
```

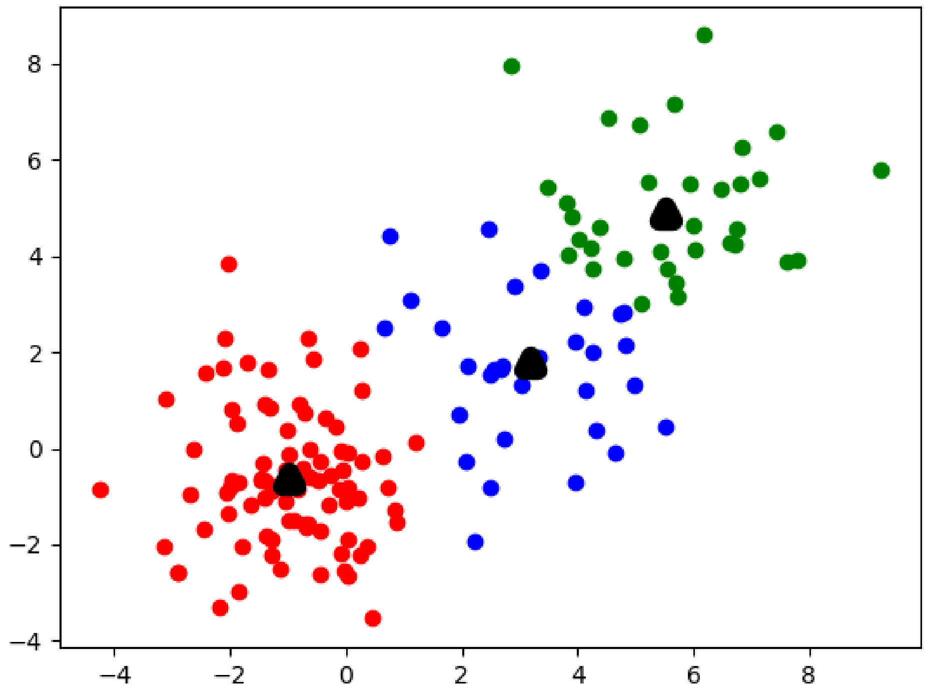
The means are:

```
[[ 5.52794237  4.89659356]
 [ 3.1917278   1.8040412 ]
 [-0.97944925 -0.64151781]]
```

The covariances are:

```
[[[ 2.31349825  0.28509797]
  [ 0.28509797  2.21584433]]
 [[ 2.02585542  0.18780605]
  [ 0.18780605  2.61775025]]
 [[ 1.20316601 -0.09844353]
  [-0.09844353  2.01675539]]]
```

- Image of clusters:



5 Comparison between my algorithm and scikit-learn

5.1 K-means Comparison

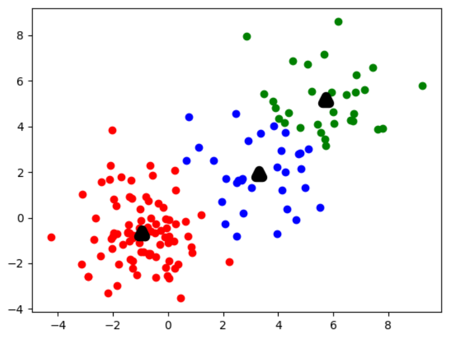
- The code of scikit-learn in K-means is:

```
from sklearn.cluster import K-means
trials = 10
clusters = 3
X = np.genfromtxt('clusters.txt', delimiter=',')
km = K-means(n_clusters=clusters, n_init=trials, algorithm ='full')
km.fit(X)
centroids = km.cluster_centers_
labels = km.labels_
colors = ["g.", "b.", "r."]
for i in range(len(X)):
    plt.plot(X[i][0], X[i][1], colors[labels[i]], markersize=12)
plt.scatter(centroids[:,0], centroids[:,1], marker='x', s=20, linewidths=10)
plt.savefig("Sklearn_K-means_cluster.png")
```

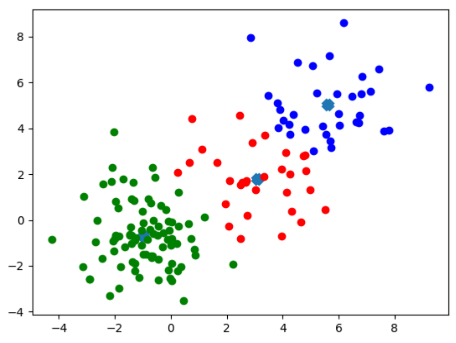
- The result is:

centroids= [[-0.97476572 -0.68419304]
[5.62016573 5.02622634]
[3.08318256 1.77621374]]

- The comparison of image between my program and scikit-learn is:



Mine



scikit-learn

5.2 GMM Comparison

- The code of scikit-learn in GMM is:

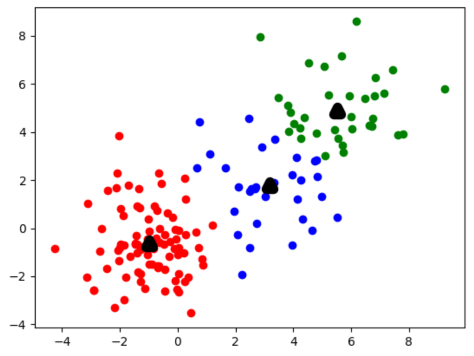
```
from sklearn import mixture
gmm = mixture.GaussianMixture(n_components=clusters, n_init=trials,
covariance_type="full")
```

```
gmm.fit(X)
labels = gmm.predict(X)
weights = gmm.weights_
means = gmm.means_
n_cov = gmm.covariances_
for i in range(len(X)):
    plt.plot(X[i][0], X[i][1], colors[labels[i]], markersize=12)
plt.scatter(means[:,0], means[:,1], marker='x', s=20, linewidths=10)
plt.savefig("Sklearn_GMM_cluster.png")
```

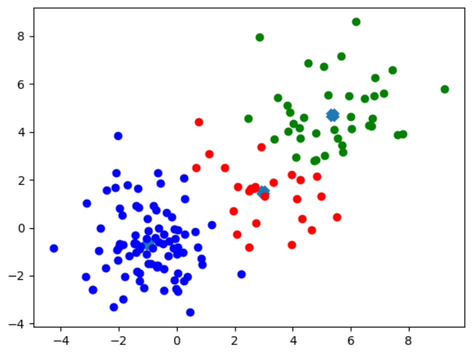
- The result is:

```
GMM weights: [0.26025477 0.56497181 0.17477342]
GMM means:
[[ 5.35474763  4.71166217]
 [-0.97675717 -0.65978187]
 [ 2.95794243  1.5172694 ]]
GMM covars: components:
[[[ 2.3640313   0.45069033]
  [ 0.45069033  2.33636882]]
 [[ 1.21681206 -0.11996906]
  [-0.11996906  1.97577318]]
 [[ 2.05537151 -0.26590085]
  [-0.26590085  2.08519189]]]
```

The comparison of results between my program and scikit-learn is:



Mine



scikit-learn

6 Application of K-means and GMM

6.1 Application of K-means

K-means can be applied to low-dimensional, numeric, and continuous data. Common applications are document clustering, identifying crime-prone areas, customer segmentation, insurance fraud detection, public transportation data analysis, IT alert clustering, etc.

6.2 Application of GMM

A common application of GMM is background subtraction in computer vision tasks. It can build a background model to detect intruding objects. In static images, though there are no moving objects, due to the influence from the external environment, it still generates some color changes. It will lead small change in the pixel value and is hard to detect by a simple model. Therefore, it would be a suitable way to use GMM to simulate color distribution in the background image.

7 Contributions

All works were completed by Chaoyu Li.