PRML Assignment 3

June 2, 2020

Introduction

In this assignment, we are going to manually construct an clustering data set, then using Gaussian Mixture Models to finish the unlabeled clustering task. After that, we will explore some method to improve the training speed and performance.

Dataset

Similar to assignment 1, we need to generate the data in gaussian distribution. We can use the numpy package, using the method **numpy.random.multivariate_normal(mean, cov, size)** to sample a batch of data that follow the Gaussian distribution with the parameter of mean and cov. You can run the following command to generate the data:

```
python source.py generate
```

the default setting is:

$$K = 3 \quad dim = 2 \quad size = [1000, 1000, 1000]$$

$$mean_0 = [1, 1] \quad cov_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$mean_1 = [5, 5] \quad cov_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$mean_2 = [9, 9] \quad cov_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

you can modify the setting by adding parameter in the command line, such as *–K, –dim*, You can check the code for detail.

Model

In this assignment, the data is unlabeled, so we can not use the method in previous assignment such as neural network, since these are supervised learning method. In this assignment, we must use unsupervised learning. There are some common unsupervised learning algorithms, such as auto-encoders and clustering. And the Gaussian Mixture Model we need to implement here is a kind of clustering algorithm, and here is a brief introduction.

Gaussian Mixture Model

A mixture model means that the probability model that consists K sub distribution. It can be represented as:

$$P(x|\theta) = \sum_{k=1}^{K} \pi_k \phi(x|\theta_k)$$

the π_k is the weight of the kth sub model in the main model. And it satisfy that $\sum \pi_i = 1$. The $\phi(x|\theta)$ is the probability distribution function, In this model we the ϕ is Gaussian distribution function, So we have:

$$\theta_k = (\mu_k, \sum)$$

$$\phi(x|\theta_k) = \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma_k|^{\frac{1}{2}}} exp(-\frac{(x-\mu_k)^T \Sigma_k^{-1} (x-\mu_k)}{2})$$

The $\sum$ means the covariance matrix of high dimensional Gaussian distribution.
Now we have the probability distribution function, then our goal is to get the best parameter that maximize the log likely hood function:

$$\begin{aligned} \hat{\theta} &= arg \max_{\theta} \sum_{i=1}^{N} logP(x_i|\theta) \\ &= arg \max_{\theta} \sum_{i=1}^{N} log(\sum_{k=1}^{K} \pi_k \phi(x|\theta_k)) \end{aligned}$$

in the assignment 1, we can directly calculate the partial derivative of the function, and let it be 0, we can get the extreme point of function. But in this case, the situation is totally different. It is because there are several distribution in the mixture

model, which means, there are some **hidden variable** (π_k) in the model, so when we calculate the partial derivative, we could not solve the extreme point directly. That's because the result formula include hidden variable, but we not know the value of hidden variable. Under this situation, we must change our method. The hidden variable is not know to us, but we can randomly initialize a parameter, in this way, we get the hidden variable. After that, we can calculate the extreme parameter of the model, then using the extreme point to update the parameter. This is the so-called **EM(Expectation maximization)** algorithm. It have two steps: first step is E-step: Calculating expectation values of each data point under the current parameter. and the M-step: Maximize the likely hood function according to the posterior probability calculated in the E-step. In this assignment, the iteration formula are as below:

$$\gamma_{jk} = \frac{\pi_k \phi(x_j|\theta_k)}{\sum_{k=1}^{K} \pi_k \phi(x_j|\theta_k)} \; j \in (1, N) \; k \in (1, K) \quad (1)$$

$$\mu_k = \frac{\sum_{j=1}^{N} \gamma_{jk} x_j}{\sum_{j=1}^{N} \gamma_{jk}} \quad (2)$$

$$\Sigma_k = \frac{\sum_{j=1}^{N} \gamma_{jk} (x_j - \mu_k)(x_j - \mu_k)^T}{\sum_{j=1}^{N} \gamma_{jk}} \quad (3)$$

$$\pi_k = \frac{\sum_{j=1}^{N} \gamma_{jk}}{N} \quad (4)$$

we calculate the equation (1) at E-step, and the equation (2) (3) (4) at the M-step. We can prove that the iteration process won't decrease the likelyhood function value. But we also need to know that the method won't necessarily get the global optimal solution, and the local optimal solution is likely to be obtained. It depends on the initialization parameter.

Experiments

In our experiment, we implement the GMM method, and visualize the result. You can run the GMM model by following command:

```
python source.py GMM
```

The default setting is $K = 3, dim = 2$(same to data generation setting), and the *iteration epochs* $= 50$. You can change the setting as you want. At first experiment, we init the parameter randomly. Here is a successful clustering result:

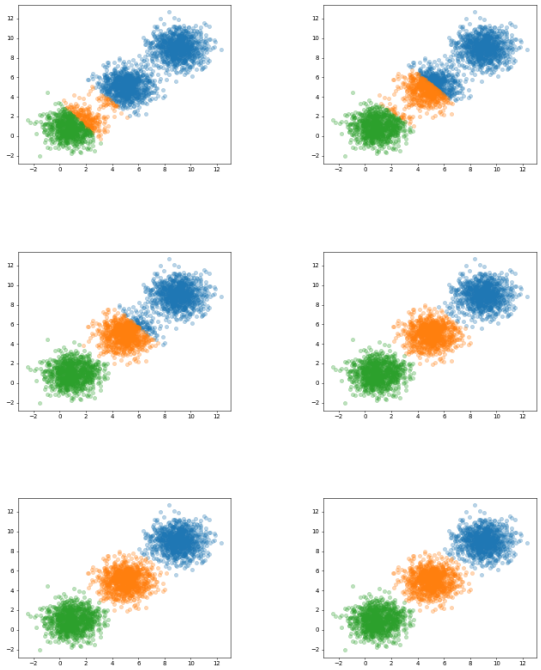

Figure 1: Successful clustering:
epoch 0 ~50 (left top to right bottom)
However, when the initialize parameter is not suitable, the result is dissatisfying:

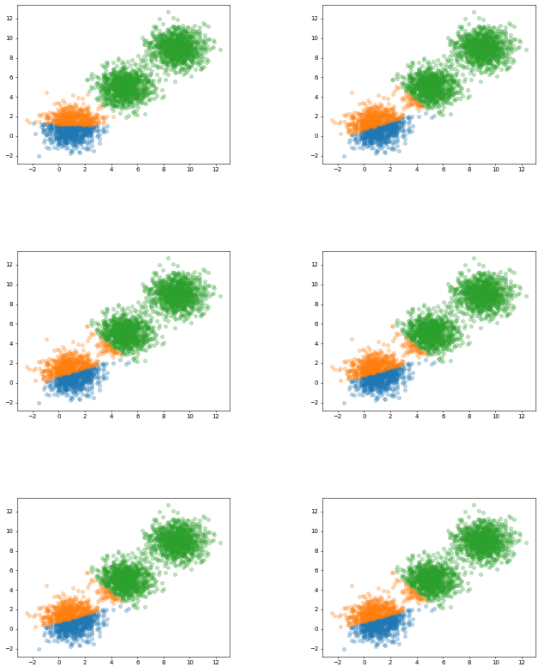

Figure 2: Unsuccessful clustering:
epoch 0~50 (left top to right bottom)
In the successful cases, the result is quite near to

the original distribution: The result of Figure1 is:

$$mean_0 = [0.9869, 0.9777] \quad cov_0 = \begin{bmatrix} 1.022 & -0.007 \\ -0.007 & 0.9.93 \end{bmatrix}$$

$$mean_1 = [5.001, 5.051] \quad cov_1 = \begin{bmatrix} 1.018 & -0.0008 \\ -0.0008 & 1.019 \end{bmatrix}$$

$$mean_2 = [8.992, 9.010] \quad cov_2 = \begin{bmatrix} 0.972 & 0.035 \\ 0.035 & 1.086 \end{bmatrix}$$

Refine Initialization

As we can see, there are two distribution initialized too close, causing the wrong boundary of the distribution. So if we want to make the GMM method more robust and stable, we must consider to refine our parameter initialize method. Here, we adopt the **K-means** algorithm to initialize the μ_k for each distribution. The algorithm is described below: we first randomly choose K points as the mean of the distribution. Then we cluster the point to the nearest mean point. In this way, the data is divided into K parts, then we recalculate the center of each part, then use the new center as the mean of the distribution. Iterating the above process, we will get a rough distribution, then we put it into GMM, the result will become more precise. The result is as below, after the initialization, the boundary is almost build.

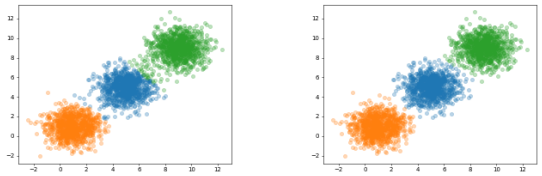

Figure 3: Successful clustering with K-means initialization: epoch 0 and epoch 10

Reorganize Data

We can change the distribution of the data, and see the performance of the method. The phenomenon is simple: when the data is clean, the method perform well, but when the data is dirty, this model can't do accurate classification. This situation is caused by the model itself: it is a clustering model, it won't learn some deep features of the data, just caring about the surface feature, so the data quality is very important. The detailed result are as below:

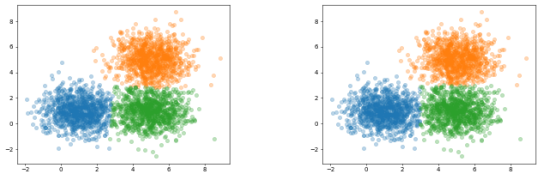

Figure 4: Good data, the model perform well: epoch 0 and epoch 10

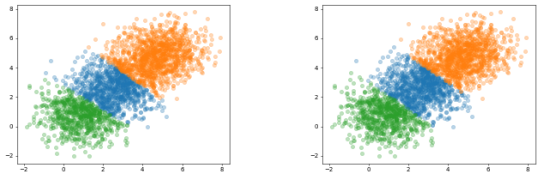

Figure 5: Bad data, Too many probability distributions overlap, the method can't do clustering properly: epoch 40 and epoch 50

Modify Hyperparameter

In the previous experiment, we have the same K for data and model. But in real-world life, we may know nothing about the distribution of the data, so we might mistaken the parameter. So we modify the K for model to see how its clustering different from the correct one. The result is shown in below figures:

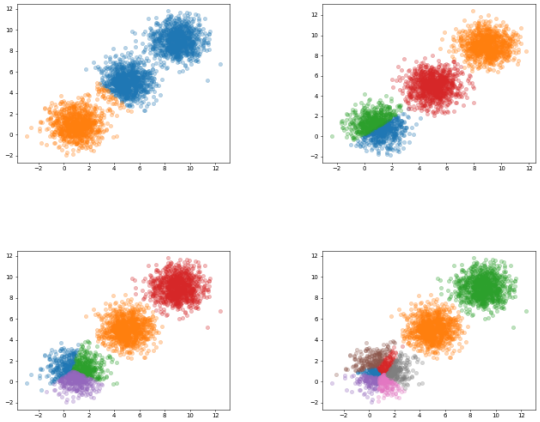

Figure 6: Mistaken model K to 2,4,5,8 (left top to right bottom)

As it's shown in the figure, when the model K is less than real K, the data in the middle is assigned to these two distributions. When the model K is greater than real K, the left bottom data is divided

into several part. That may caused by initialize strategy, the initialize values are often small, so the centers of these data are near the lower left corner.

Conclusion

Gaussian Mixture Model is a unsupervised learning method, it can do clustering to a large scale of data. and it's speed is fast. However, it also has some flaws. It is very dependent on the initial settings of the model, and has relatively high requirements for data quality. It can be used to do some data preprocessing and rough labeling work.

References

[1] https://numpy.org/

[2] https://nndl.github.io/