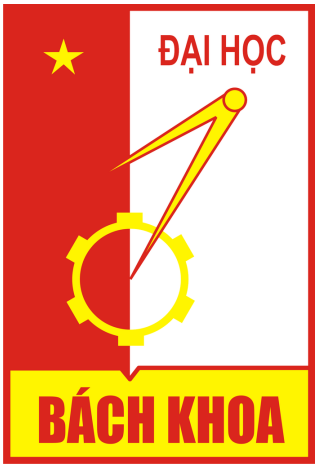
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

o0o



Báo cáo môn Project 1

Cài đặt thuật toán phân cụm KMeans và Gaussian Mixture Model

Giáo viên hướng dẫn: Nguyễn Thị Thu Hương

Sinh viên thực hiện : Nguyễn Hữu Hiệp - 20183528

Lớp : CTTN CNTT K63

Hà Nội - 2020

Mục lục

1 Giới thiệu về bài toán phân cụm

1.1 Giới thiệu

Bài toán phân cụm được xem là bài toán quan trọng nhất trong học không giám sát (dữ liệu đầu vào không được gán nhãn). Ta xem cụm là một tập hợp các dữ liệu có tính tương đồng và khác với các dữ liệu không thuộc tập hợp đó, phân cụm là việc ta chia toàn bộ dữ liệu thành các nhóm nhỏ dựa trên sự liên quan giữa các dữ liệu trong mỗi nhóm. Bài toán phân cụm có thể giúp ta tìm được cấu trúc, tính chất ẩn bên trong dữ liệu và nó cũng có thể được thực hiện như một công cụ độc lập giúp hiểu sâu về phân phối dữ liệu hoặc như là một bước tiền xử lí cho các thuật toán khác.
Như vậy, việc tìm ra được mối quan hệ ẩn giữa các điểm dữ liệu đã khiến cho bài toán phân cụm trở nên quan trọng. Ví dụ:

i) Trong marketing, khách hàng được phân thành các nhóm có tính tương đồng để thực hiện chiến lược một cách hiệu quả theo từng nhóm.

ii) Cho một tập các văn bản, ta cần tổ chức chúng theo sự tương đồng về mặt nội dung để tạo hệ thống phân cấp chủ đề.

1.2 Phân loại

Dựa trên cách thức phân cụm, bài toán được chia ra làm hai loại:

i) Hard clustering: mỗi điểm dữ liệu hoặc thuộc một cụm hoặc không.

ii) Soft clustering: thay vì gán mỗi điểm dữ liệu vào một cụm riêng biệt thì ta chỉ gán xác suất để điểm dữ liệu đó thuộc từng cụm.

Điển hình cho hai loại trên là hai thuật toán cơ bản: K-means và Gaussian Mixture Models.

2 Thuật toán K-means

2.1 Giả thiết

Thuật toán này được xây dựng dựa trên giả thiết đơn giản là:

i) Mỗi điểm dữ liệu thuộc một cụm duy nhất

ii) Các điểm thuộc cùng một cụm thì gần nhau về mặt khoảng cách

Một cách trực quan, mỗi cụm sẽ được kèm với một tâm cụm và với mỗi điểm dữ liệu, dựa trên khoảng cách đến từng tâm cụm, ta có thể phân điểm dữ liệu đó vào cụm mà có khoảng cách ngắn nhất.

2.2 Mô hình

Mô hình của bài toán bao gồm:

i) Đầu vào: tập dữ liệu $\{x_n\}$, tổng số cụm là K.

ii) Đầu ra:

(a) Tập các centroid ứng với mỗi cụm $\{\mu_k\}$

(b) Kết quả phân cụm ứng với mỗi điểm dữ liệu $\{Z_n^k\}$: $\sum_{k=1}^{K} Z_n^k = 1$, $Z_n^k \in \{0, 1\}$

2.3 Thuật toán

Dựa vào hai giả thiết trên, thuật toán hướng đến tối ưu tổng khoảng cách từ các điểm dữ liệu đến cụm mà điểm đó được gán. Hàm loss:

$$\sum_{n=1}^{N}\sum_{k=1}^{K} Z_n^k ||x_n - \mu_k||_2^2$$

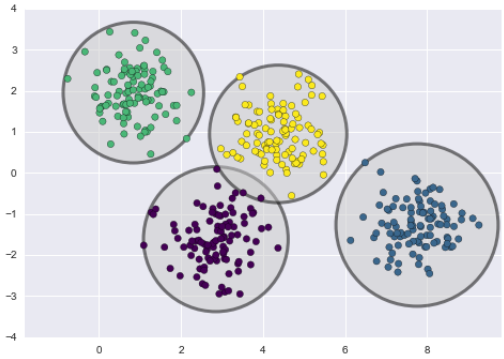
Để tối ưu hàm trên, ta sử dụng phương pháp Coordinate descent, ta lần lượt tối ưu theo Z_n^k và μ_k. Bằng cách tìm nghiệm của các phương trình đạo hàm riêng phần theo từng biến bằng 0, ta thu được kết quả và thuật toán sẽ lặp lại cho tới khi hội tụ các bước:

- Cập nhật tâm cụm: $\mu_k = \frac{\sum_{n=1}^{N} Z_n^k x_n}{\sum_{n=1}^{N} Z_n^k}$
- Cập nhật kết quả phân cụm: $Z_n^k = \arg\min_k ||x_n - \mu_k||$

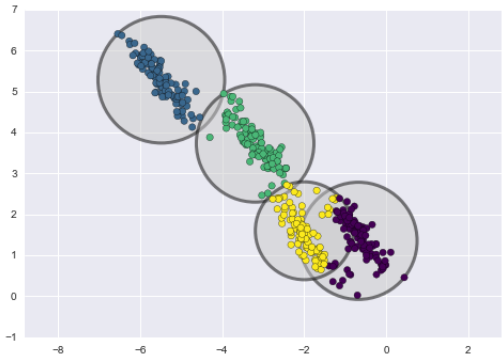
3 Gaussian Mixture Models

3.1 Hạn chế của thuật toán K-means

Thuật toán K-means là một thuật toán đơn giản, dễ hiểu, tuy nhiên lại có một số điểm hạn chế. Thứ nhất, K-means không xét đến hiệp phương sai (với hiệp phương sai, ta có thể xác định được hình dạng của phân phối). Một cách trực quan, mô hình này đặt một hình tròn (hoặc với dữ liệu nhiều chiều là siêu cầu) tại mỗi tâm cụm với bán kính là khoảng cách xa nhất đến một điểm dữ liệu thuộc cụm đó.



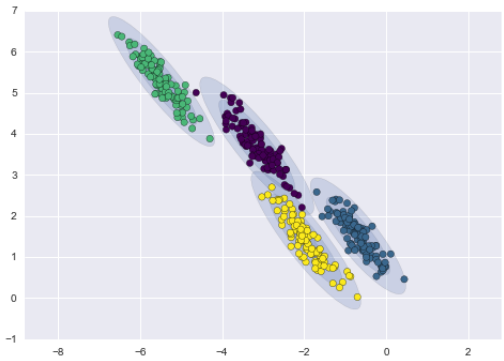
Điều này khiến thuật toán thực hiện tốt khi dữ liệu phân bố theo dạng hình tròn. Tuy nhiên khi dữ liệu có dạng khác, ta sẽ thu được kết quả kém hơn như:



Thứ hai, K-means là thuộc nhóm hard-clustering. Nói theo cách khác, thuật toán này cho ta biết dữ liệu thuộc cụm nào nhưng không cho biết xác suất để dữ liệu thuộc cụm đó.

3.2 Gaussian Mixture Model

Gaussian Mixture Model được đưa ra nhằm giải quyết hai vấn đề trên dựa trên hướng tiếp cận soft clustering và việc thêm hiệp phương sai vào mô hình. Với ví dụ là bộ dữ liệu được đề cập ở trên, thuật toán này cho ta kết quả tốt hơn rõ ràng so với K-means



3.2.1 Giả thiết

Thuật toán được xây dựng dựa trên hai giả thiết quan trọng:

i) Phân phối dữ liệu trong mỗi cụm là Multivariate Gaussian

ii) Phân phối của tập dữ liệu là Gaussian Mixture

3.2.2 Mô hình

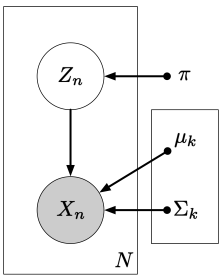
Mô hình của bài toán bao gồm:

i) Đầu vào: tập dữ liệu $\{x_n\}$, tổng số cụm là K.

ii) Đầu ra:

(a) Tập kì vọng và hiệp phương sai ứng với mỗi cụm $\{\mu_k, \Sigma_k\}$

(b) Xác suất thuộc từng cụm ứng với mỗi điểm dữ liệu $P(Z_n = k|x_n)$



Trong mô hình trên, X là biến quan sát, Z là biến ẩn; μ, Σ và π (xác suất tiên nghiệm của Z) là siêu tham số.

- $Z \sim Mult(\pi)$: $P(Z = k|\pi) = \pi_k$
- $x_n \sim \mathcal{N}(\mathrm{x}_n; \mu_{Zn}, \Sigma_{Zn})$

3.2.3 Thuật toán

Dựa vào mô hình trên, dễ thấy dữ liệu được quan sát một phần (chỉ biến X được quan sát) và biến ẩn Z là rời rạc. Do đó, thuật toán EM có thể được áp dụng để giải quyết bài toán này.
Thuật toán lặp các bước sau cho tới khi hội tụ:

E step: sử dụng tham số tại bước lặp hiện tại để tính expected sufficient statistic:

- Đặt $T_n^k = P(z_n = k|x_n, \pi, \mu, \Sigma) = \frac{\pi_k \mathcal{N}(x_n, \mu_k, \Sigma_k)}{\sum_{i=1}^{K} \pi_i \mathcal{N}(x_n, \mu_i, \Sigma_i)}$
- Coi bộ dữ liệu hiện tại gồm $\{Z_n = k, x_n\}$ với số lần xuất hiện là T_n^k

M step: cực đại hóa log likelihood theo π, μ, Σ dựa trên bộ dữ liệu được quan sát đầy đủ.

- Log likelihood:
  L= $\sum_{n=1}^{N} \sum_{k=1}^{K} T_n^k \log(P(z_n = k, x_n|\pi, \mu, \Sigma)$
  $= \sum_{n=1}^{N} \sum_{k=1}^{K} T_n^k (\log(P(z_n = k|\pi) + \log(P(x_n|z_n = k, \mu_k, \Sigma_k))$
  $= \sum_{n=1}^{N} \sum_{k=1}^{K} T_n^k (\log(\pi_k) + \log(\mathcal{N}(x_n; \mu_k, \Sigma_k))$
- Sử dụng Coordinate ascent và phương pháp nhân tử Lagrange ta thu được kết quả:
  - $\pi_k = \frac{\sum_{n=1}^{N} T_n^k}{n}$
  - $\mu_k = \frac{\sum_{n=1}^{N} T_n^k x_n}{\sum_{n=1}^{N} T_n^k}$
  - $\Sigma_k = \frac{\sum_{n=1}^{N} (x_n - \mu_k)^T T_n^k (x_n - \mu_k)}{\sum_{n=1}^{N} T_n^k}$

4 Cài đặt thực nghiệm

4.1 Các bộ thư viện được sử dụng

i) Thư viện numpy: giúp tính toán với ma trận và sinh dữ liệu theo phân phối Multivariate Gaussian.

ii) Thư viện scipy: giúp tính xác xuất theo phân phối Multivariate Gaussian khi cập nhật T_n^k ở E step.

iii) Thư viện matplotlib: giúp vẽ đồ thị giá trị hàm log likelihood theo từng bước lặp và thể hiện sự phân cụm đối với dữ liệu 2 chiều.

iv) Thư viện sklearn: giúp lấy bộ dữ liệu Iris Flower.

4.2 Bộ dữ liệu được sử dụng

i) Bộ dữ liệu Iris Flower: gồm 150 bản ghi về 3 loài hoa theo 4 đặc trưng.

```
106   X_iris = datasets.load_iris()['data']
107   KM2 = KMeans(n_clusters= 3, n_features= 4)
```

Hàm load_ris() trong thư viện sklearn trả về một dictionary. Value ứng với key 'data' là dữ liệu ta sử dụng (một mảng 2 chiều kích thước 150x4 ứng với số điểm dữ liệu và số đặc trưng).

ii) Bộ dữ liệu 2 chiều được sinh ngẫu nhiên gồm 3 cụm.

```python
def generate_data(n_samples= 3000):
    #randomly generate data
    x1 = np.random.multivariate_normal(mean=[0, 0], cov=[[10, 7], [7, 10]], size=n_samples // 3)
    x2 = np.random.multivariate_normal(mean=[5, -15], cov=[[5, 8], [8, 19]], size=n_samples // 3)
    x3 = np.random.multivariate_normal(mean=[17, 14], cov=[[27, 3], [3, 3]], size=n_samples // 3)
    x = np.concatenate((x1, x2, x3), axis=0)
    index = np.arange(0, n_samples // 3 * 3)
    np.random.shuffle(index)
    y = np.array([x[i] for i in index])  # shape= (n, d)
    return y

X_generate = generate_data(n_samples= 1000)
```

Hàm *generate_data*() giúp sinh dữ liệu 2 chiều theo 3 cụm với số điểm dữ liệu là tham số được truyền vào hàm và trả về bộ dữ liệu vừa được sinh. Mỗi cụm có số điểm dữ liệu như nhau được sinh dựa trên phân phối multivariate Gaussian bằng cách sử dụng hàm *np.random.multivariate_normal*() (với kì vọng và hiệp phương sai là tham số cho trước được truyền vào). Sau khi khởi tạo được các cụm, ta sử dụng hàm *np.concatenate*() giúp ghép các điểm dữ liệu và *np.shuffle*() để trộn bộ dữ liệu đó.

4.3 Cài đặt

4.3.1 Thuật toán K-means

Ta xây dựng một lớp KMeans với phương thức khởi tạo gồm các tham số truyền vào là số cụm ($n\_clusters$) và số đặc trưng ($n\_features$).

```python
class KMeans:
    def __init__(self, n_clusters, n_features):
        self.n_clusters = n_clusters
        self.n_features = n_features
        self.X = None
```

Hàm fit() thực hiện huấn luyện mô hình với tham số truyền vào là bộ dữ liệu (X), số epoch (n_epochs), sai số (epsi) và plot để xác định có đưa ra kết quả trực quan sau khi huấn luyện hay không.

```python
    def fit(self, X, n_epochs = 20, epsi = 1e-6, plot= True):
        self.X = X
        self.init_parameters()
        self.update_distance()
        epoch = 1
        while self.criterion(epoch= epoch, n_epochs= n_epochs, epsi= epsi, plot = plot):
            self.update_centroids()
            self.update_distance()
            self.compute_loss()
            epoch += 1
            print('Epoch {}:----------------------------'.format(epoch + 1))
```

Huấn luyện bắt đầu bằng việc khởi tạo các biến, tham số và lặp lại việc cật nhật tâm cụm, kết quả phân cụm cho tới khi điều kiện dừng thỏa mãn.
Trong hàm khởi tạo tham số (*init_parameters*()), *centroids*, *distances*, *predict* là các biến mảng với các phần tử tương ứng là tâm cụm, khoảng cách từ một điểm dữ liệu tới lần lượt từng cụm và kết quả phân cụm ứng với mỗi điểm dữ liệu.

```
    def init_parameters(self):
        #initialize p
        self.n_samples = self.X.shape[0]

        max_dims = np.array([np.max(self.X[:, i]) for i in range(self.n_features)])
        min_dims = np.array([np.min(self.X[:, i]) for i in range(self.n_features)])
        d = (max_dims - min_dims) * 1. / (self.n_clusters - 1)
        self.centroids = np.array([min_dims + d * i for i in range(self.n_clusters)])

        self.distances = np.zeros(shape= (self.n_samples, self.n_clusters))
        self.predict = np.zeros(shape=  (self.n_samples, 1))
        self.loss =  []
```

Với *distances* và *predict* ta khởi tạo là 0 tuy nhiên với tâm cụm, ta chọn một cách khởi tạo mặc định. Đó là việc ta chia miền giá trị của từng chiều trên bộ dữ liệu thành K khoảng bằng nhau (K là số cụm) và gán các tâm cụm tương ứng trên đó. Việc cập nhật khoảng cách và tâm cụm, ta thực hiện như thuật toán được trình bày ở trên.

```
    def update_distance(self):
        # update distance
        for i in range(self.n_clusters):
            self.distances[:, i] = [np.linalg.norm((self.X - self.centroids[i])[j]) for j in range(self.n_samples)]
        self.predict = np.argmin(self.distances, axis= 1)

    def update_centroids(self):
        #update centroids
        for i in range(self.n_clusters):
            cluster_point = (self.predict == i).reshape(-1, 1)
            self.centroids[i] = np.sum((self.X * cluster_point), axis= 0) / np.sum(cluster_point)

```

Với từng tâm cụm, ta kiểm tra các điểm dữ liệu có được gán vào cụm đó hay không và lưu vào biến *cluster_point* (0 nếu không thuộc, 1 nếu thuộc) rồi nhân element-wise với X (bộ dữ liệu) trước khi tính trung bình. Hàm *criterion*() kiểm tra điều kiện dừng (trả về false nếu hội tụ) với tham số truyền vào là số epochs (*n_epoch*), sai số (*epsi*), *plot* để kiểm tra có cần đưa ra kết quả huấn luyện không.

```
     def criterion(self,epoch, n_epochs, epsi, plot):
         #check stopping criterion
         if epoch <= 2:
             return True
         if (epoch < n_epochs) and ((self.loss[-2] - self.loss[-1]) > epsi):
             return True
         if plot:
             self.plot()
         return False

```

Nếu số lần lặp nhỏ hơn hoặc hiệu giữa hai giá trị hàm loss trong hai lần gần nhất lớn hơn *epsi* thì hàm trả về true (chưa hội tụ).

4.3.2 Gaussian Mixture Model

Thuật toán này được cài đăt tương tự như K-means, phương thức khởi tạo với hai tham số chuyền vào là số cụm và số đặc trưng.

```
class GaussianMixtureModel:
    def __init__(self, n_clusters, n_features):
        self.n_clusters = n_clusters
        self.n_features = n_features
```

Hàm $fit()$ cũng được xây dựng để thực hiện việc huấn luyện tương tự như thuật toán K-means, điểm khác biệt là hàm này lặp lại hai bước e_step và m_step cùng với cập nhật $log\_likelihood$.

```
    def fit(self, X, n_epochs= 20, epsi= 1e-3, plot= True):
        #training
        self.X = X
        self.init_parameters()
        epoch = 1
        while self.criterion(epoch= epoch, n_epochs= n_epochs, epsi= epsi, plot= plot):
            self.e_step()
            self.m_step()
            self.compute_log_likelihood()
            epoch += 1
            if (epoch / n_epochs * 100) % 10 == 0:
                print('Epoch {}----------------------------------------'.format(epoch))
    def init_parameters(self):
        #initialize parameters
        self.n_samples = self.X.shape[0]

        min_of_dims = np.array([np.min(self.X[:, i]) for i in range(self.n_features)])
        max_of_dims = np.array([np.max(self.X[:, i]) for i in range(self.n_features)])
        d = (max_of_dims - min_of_dims) * 1. / (self.n_clusters - 1)
        #initialize mean and covariance of each cluster
        self.mean = np.array([min_of_dims + d * i for i in range(self.n_clusters)])
        self.cov = np.zeros(shape= (self.n_clusters, self.n_features, self.n_features))
        for i in range(self.n_clusters):
            self.cov[i] += np.identity(self.n_features) * np.random.randint(5, 10)
        self.reg_cov = 1e-6 * np.identity(self.n_features)      #avoid that covariance equals to zero

        self.alpha = np.array([1. / self.n_clusters for i in range(self.n_clusters)])     #prior probability of latent var
        self.T = np.zeros(shape=(self.n_samples, self.n_clusters))      #posterior probability of latent variable
        self.log_likelihood = []
```

Trong hàm $init\_parameter()$, ta thực hiện khởi tạo các tham số:

- Tâm cụm ($mean$) khởi tạo tương tự như K-means.
- Hiệp phương sai (cov) bằng tích của ma trận đơn vị với một số ngẫu nhiên trong khoảng từ 5 đến 10 (nhằm khởi tạo ngẫu nhiên và đảm bảo ma trận xác định dương).
- reg_cov được gán bằng tích của một số vô cùng bé với ma trận đơn vị (được sử dụng để cộng thêm vào cov mỗi khi tính phân phối multivariate Gaussian giúp tránh hiện tượng $cov = 0$)
- $alpha$ là Xác suất tiên nghiệm cho biến ẩn Z_k và được khởi tạo bằng $\frac{1}{k}$ cho mỗi cụm
- T là xác suất hậu nghiệm ($T_n^k = P(z_n = k|x_n, \pi, \mu, \Sigma)$) được khởi tạo bằng 0
- $log\_likelihood$ là một list để lưu trữ kết quả log likelihood qua từng bước huấn luyện

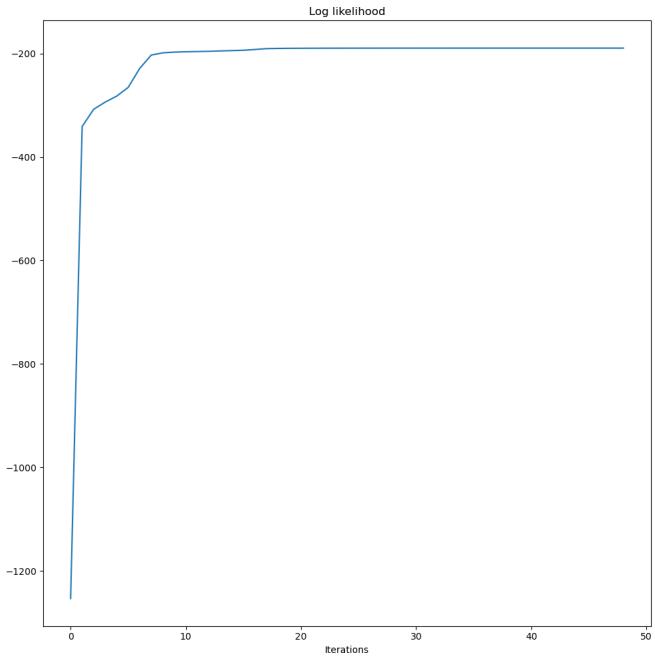
```
    def e_step(self):
        #update posterior
        self.pdf = np.array([multivariate_normal(mean= self.mean[i], cov= self.cov[i] + self.reg_cov).pdf(self.X)
                             for i in range(self.n_clusters)]).T     #probabil of that each observation belongs to each cl
        numerator = self.pdf * self.alpha
        denominator = np.sum(numerator, axis= 1).reshape(-1, 1)
        self.T = numerator / denominator

    def m_step(self):
        #update hyperparameters: alpha, mean, cov
        self.alpha = np.sum(self.T, axis= 0) / self.n_samples
        for cluster in range(self.n_clusters):
            t = np.reshape(self.T[:, cluster], (-1, 1))
            self.mean[cluster] = np.sum(self.X * t, axis= 0) / np.sum(t, axis= 0)
            self.cov[cluster]= ((self.X - self.mean[cluster]).T * self.T[:, cluster]).dot((self.X - self.mean[cluster])) /
```

Với các hàm cập nhật *e_step* và *m_step*, ta thực hiện theo các bước và công thức được mô tả trong thuật toán trên, với biến *pdf* lưu xác suất của từng điểm dữ liệu ứng với từng cụm ($\mathcal{N}(x_n, \mu_k, \Sigma_k)$).

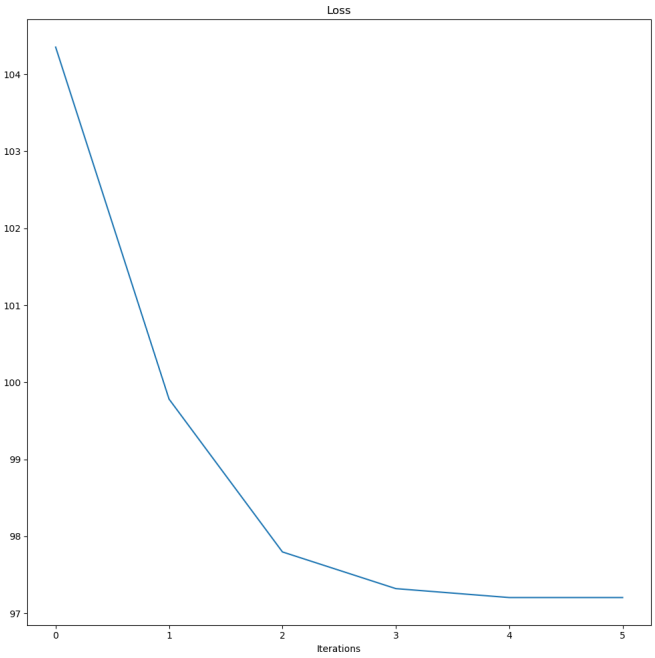
5 Kết quả thực nghiệm

5.1 Bộ dữ liệu Iris

5.1.1 K-Means

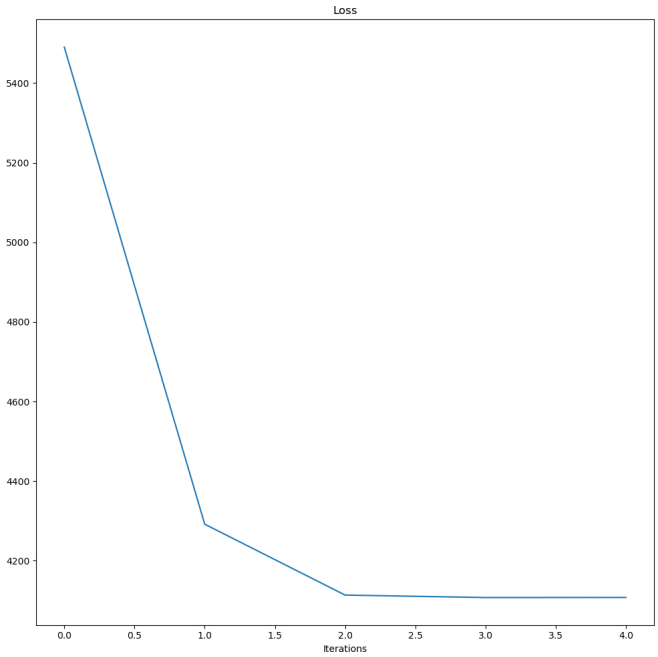


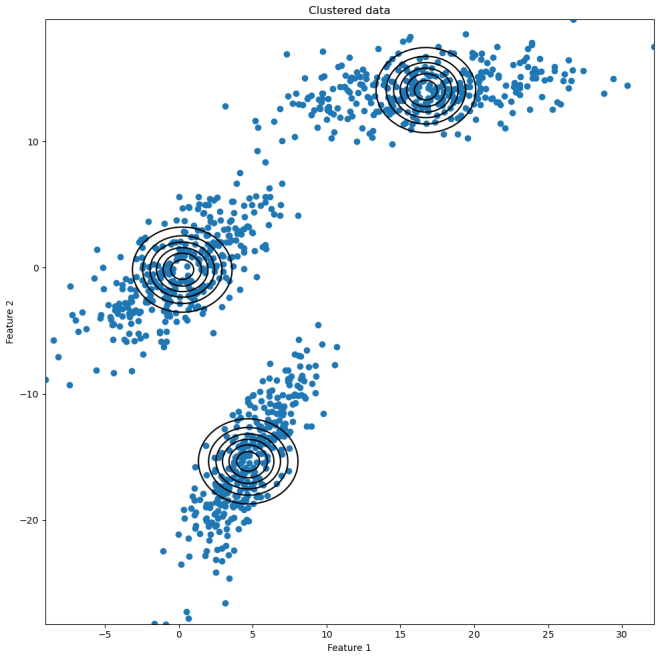
5.1.2 Gaussian Mixture Model



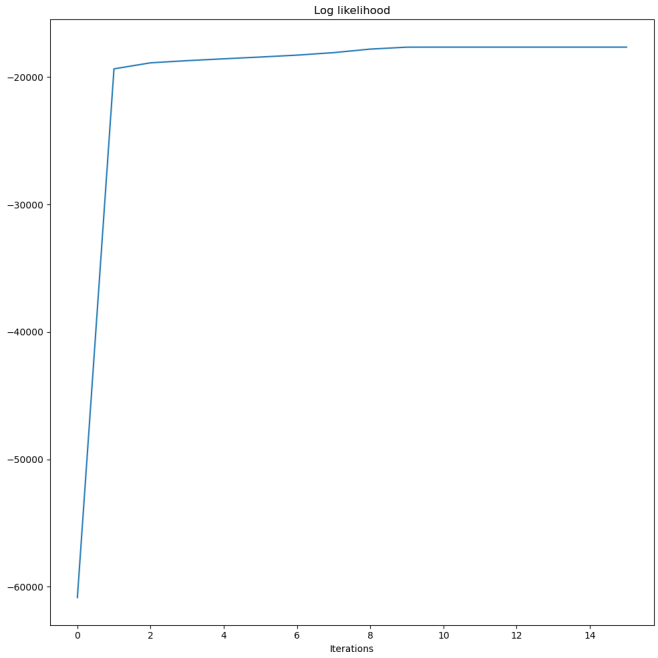
5.2 Bộ dữ liệu được sinh

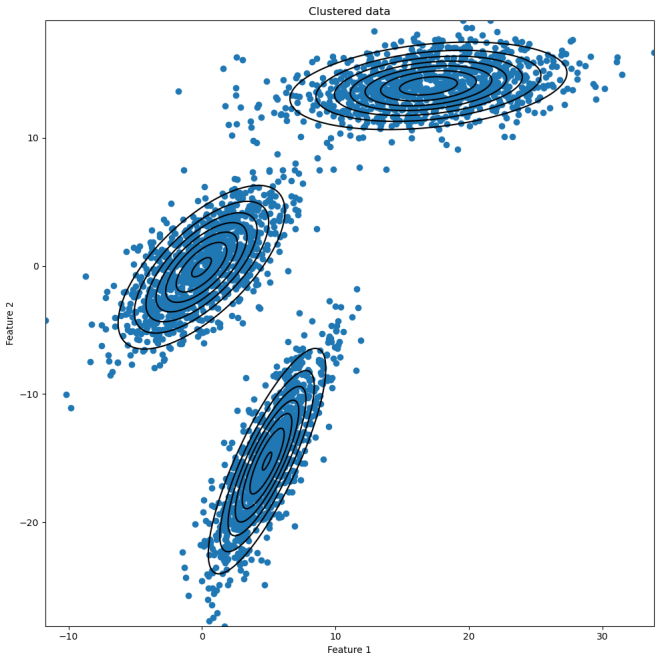
5.2.1 K-Means





5.2.2 Gaussian Mixture Model





6 Kết luận

Dựa trên lý thuyết và cả thực nghiệm, ta dễ thấy thuật toán Gaussian Mixture Model cho kết quả tốt hơn so với KMeans. Nguyên nhân có thể kể đến là do:

- Gaussian Mixture Model có thể biểu diễn được phương sai trên phân phối của từng cụm, phù hợp với các bài toán thực tế khi phân bố của các điểm dữ liệu trong không gian phức tạp (điều này không thể làm được với KMeans)
- Gaussian Mixture Model phân cụm dựa trên xác suất giúp thể hiện tính không chắc chắn, qua đó giúp việc tối ưu cũng hiệu quả hơn (do việc cập nhật từng cụm cũng có tính tương quan đến các cụm khác, trong khi KMeans khi cập nhật từng cụm thì không có được sự ảnh hưởng đó).

Hai thuật toán tuy còn có một số nhược điểm như kết quả phân cụm phụ thuộc vào khởi tạo, nhạy cảm với các điểm dữ liệu bất thường, ... nhưng có thể thấy đây là hai thuật toán đơn giản, dễ hiểu, dễ cài đặt, và là tiêu biểu cho bài toán phân cụm trong Machine Learning.

Tài liệu

[1] Daphne Koller, Nir Friedman (2009), *Probabilistic Graphical Models: Principles and Techniques*, The MIT Press (2009)