哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称：机器学习

课程类型：必修

实验题目：实现k-means聚类方法和混合高斯模型

学号：1190200523

姓名：石翔宇

1 实验目的

实现一个 k-means 算法和混合高斯模型，并且用 EM 算法估计模型中的参数。

2 实验要求及实验环境

2.1 实验要求

测试 用高斯分布产生 k 个高斯分布的数据（不同均值和方差），其中参数自己设定。

1. 用 k-means 聚类，测试效果；
2. 用混合高斯模型和你实现的 EM 算法估计参数，看看每次迭代后似然值变化情况，考察 EM 算法是否可以获得正确的结果（与你设定的结果比较）。

应用 可以 UCI 上找一个简单问题数据，用你实现的 GMM 进行聚类。

2.2 实验环境

Windows 11 + Python 3.7.8

3 设计思想

3.1 k-means 算法

给定 n 个样本的集合 $X=\{x_1,x_2,\dots,x_n\}$，$x_i\in\mathbb{R}^m$，k-means 聚类的目标是将 n 个样本分到 k 个不同的类或簇中（假设 $k<n$）。k 个类 $G_1,G_2,\dots,G_k$ 形成对集合 X 的划分，其中 $G_i\cap G_j=\emptyset$，$\bigcup\limits_{i=1}^{k}G_i=X$。用 C 表示划分，一个划分对应着一个聚类结果。

k-means 算法通过损失函数的最小化选取最优的划分 C^*。

首先，我们将样本之间的距离 $d(x_i,x_j)$ 定义为欧氏距离平方

$$\begin{aligned} d(x_i,x_j)=&\sum_{k=1}^{m}(x_{ik}-x_{jk})^2\\ =&\|x_i-x_j\|^2 \end{aligned}\tag{1}$$

我们定义损失函数 $W(C)$ 为

$$W(C)=\sum_{l=1}^{k}\sum_{C(i)=l}\|x_i-\bar{x}_l\|^2\tag{2}$$

其中，$\bar{x}_l=\frac{1}{n_l}\sum\limits_{C(i)=l}x_i$，$n_l=\sum\limits_{i=1}^{n}I(C(i)=l)$。

则 k-means 算法就是求解最优化问题

$$\begin{aligned} C^*=&arg\min_C W(C)\\ =&arg\min_C\sum_{l=1}^{k}\sum_{C(i)=l}\|x_i-\bar{x}_l\|^2 \end{aligned}\tag{3}$$

k-means 算法是一个迭代的过程。首先，对于给定的中心值 $(m_1, m_2, \ldots, m_k)$，将每个样本指派到与其最近的中心 m_l 的类 G_l 中，得到聚类结果，使得目标函数极小化

$$\min_{m_1,\ldots,m_k} = \sum_{l=1}^{k} \sum_{C(i)=l} \|x_i - m_l\|^2 \tag{4}$$

然后，对于每个包含 n_l 个样本的类 G_l，更新其均值 m_l

$$m_l = \frac{1}{n_l} \sum_{C(i)=l} x_i \tag{5}$$

其中，$l = 1, 2, \ldots, k$。

重复上述两个步骤，直到 $W(C)$ 结果小于阈值。

3.2 高斯混合模型（GMM）

高斯混合模型是指具有如下形式的概率分布模型：

$$P(y|\theta) = \sum_{k=1}^{K} \alpha_k \phi(y|\theta_k) \tag{6}$$

其中，$\alpha_k \geq 0$ 是系数，满足 $\sum\limits_{k=1}^{K} \alpha_k = 1$；$\phi(y|\theta_k)$ 是高斯分布密度，$\theta_k = (\mu_k, \sigma_k^2)$，$\phi(y|\theta_k) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp(-\frac{(y-\mu_k)^2}{2\sigma_k^2})$。

现有观测数据 $y = \{y_1, y_2, \ldots, y_N\}$ 由高斯混合模型生成，

$$P(y|\theta) = \sum_{k=1}^{K} \alpha_k \phi(y|\theta_k) \tag{7}$$

其中，$\theta = (\alpha_1, \alpha_2, \ldots, \alpha_K; \theta_1, \theta_2, \ldots, \theta_k)$。我们将用 EM 算法估计高斯混合概率模型的参数 θ。

我们定义隐变量 0-1 随机变量 γ_{jk} 为

$$\gamma_{jk} = \begin{cases} 1, & \text{第}j\text{ 个观测来自第}k\text{ 个分模型} \\ 0, & \text{否则} \end{cases} \tag{8}$$
$$j = 1, 2, \cdots, N; \quad k = 1, 2, \cdots, K$$

那么完全数据为

$$(y_j, \gamma_{j1}, \gamma_{j2}, \ldots, \gamma_{jK}), \quad j = 1, 2, \ldots, N \tag{9}$$

则似然函数为

$$\begin{aligned} P(y, \gamma|\theta) &= \prod_{j=1}^{N} P(y_j, \gamma_{j1}, \gamma_{j2}, \ldots, \gamma_{jK}|\theta) \\ &= \prod_{k=1}^{K} \prod_{j=1}^{N} [\alpha_k \phi(y_j|\theta_k)]^{\gamma_{jk}} \\ &= \prod_{k=1}^{K} \alpha_k^{n_k} \prod_{j=1}^{N} [\phi(y_j|\theta_k)]^{\gamma_{jk}} \\ &= \prod_{k=1}^{K} \alpha_k^{n_k} \prod_{j=1}^{N} [\frac{1}{\sqrt{2\pi}\sigma_k} \exp(-\frac{(y-\mu_k)^2}{2\sigma_k^2})]^{\gamma_{jk}} \end{aligned} \tag{10}$$

其中，$n_k=\sum\limits_{j=1}^{N}\gamma_{jk}$，$\sum\limits_{k=1}^{K}n_k=N$。

则对数似然函数为

$$\log P(y,\gamma|\theta)=\sum_{k=1}^{K}\{n_k\log\alpha_k+\sum_{j=1}^{N}\gamma_{jk}[\log(\frac{1}{\sqrt{2\pi}})-\log\sigma_k-\frac{1}{2\sigma_k^2}(y-\mu_k)^2]\} \tag{11}$$

EM 算法的 E 步要求我们确定 Q 函数

$$\begin{aligned}Q(\theta,\theta^{(i)})=&E[\log P(y,\gamma|\theta)]\\=&E\{\sum_{k=1}^{K}\{n_k\log\alpha_k+\sum_{j=1}^{N}\gamma_{jk}[\log(\frac{1}{\sqrt{2\pi}})-\log\sigma_k-\frac{1}{2\sigma_k^2}(y-\mu_k)^2]\}\}\\=&\sum_{k=1}^{K}\{\sum_{j=1}^{N}(E\gamma_{jk})\log\alpha_k+\sum_{j=1}^{N}(E\gamma_{jk})[\log(\frac{1}{\sqrt{2\pi}})-\log\sigma_k-\frac{1}{2\sigma_k^2}(y-\mu_k)^2]\}\end{aligned} \tag{12}$$

这里需要计算 $E(\gamma_{jk}|y,\theta)$，记为 $\hat{\gamma}_{jk}$：

$$\begin{aligned}\hat{\gamma}_{jk}=&E(\gamma_{jk}|y_j,\theta)=P(\gamma_{jk}=1|y_j,\theta)\\=&\frac{P(\gamma_{jk}=1,y_j|\theta)}{\sum\limits_{k=1}^{K}P(\gamma_{jk}=1,y_j|\theta)}\\=&\frac{P(y_j|\gamma_{jk}=1,\theta)P(\gamma_{jk}=1|\theta)}{\sum\limits_{k=1}^{K}P(y_j|\gamma_{jk}=1,\theta)P(\gamma_{jk}=1|\theta)}\\=&\frac{\alpha_k\phi(y_j|\theta_k)}{\sum\limits_{k=1}^{K}\alpha_k\phi(y_j|\theta_k)},\quad j=1,2,\ldots,N;\quad k=1,2,\ldots,K\end{aligned} \tag{13}$$

将 $\hat{\gamma}_{jk}=E\gamma_{jk}$ 和 $n_k=\sum\limits_{j=1}^{N}E\gamma_{jk}$ 代入式 13 得

$$Q(\theta,\theta^{(i)})=\sum_{k=1}^{K}\{n_k\log\alpha_k+\sum_{j=1}^{N}\hat{\gamma}_{jk}[\log(\frac{1}{\sqrt{2\pi}})-\log\sigma_k-\frac{1}{2\sigma_k^2}(y-\mu_k)^2]\} \tag{14}$$

EM 算法的 M 步是要求得函数 $Q(\theta,\theta^{(i)})$ 对 θ 的极大值，即

$$\theta^{(i+1)}=\arg\max_{\theta}Q(\theta,\theta^{(i)}) \tag{15}$$

将式 14 分别对 μ_k 和 σ_k^2 求偏导并令其为 0，可得

$$\begin{aligned}\frac{\partial Q}{\mu_k}=&\sum_{j=1}^{N}[\hat{\gamma}_{jk}\frac{1}{\sigma_k^2}(y-\mu_k)]=0\\\sum_{j=1}^{N}(\hat{\gamma}_{jk}y)=&\sum_{j=1}^{N}(\hat{\gamma}_{jk}\mu_k)\\\sum_{j=1}^{N}(\hat{\gamma}_{jk}y)=&\mu_k\sum_{j=1}^{N}\hat{\gamma}_{jk}\\\mu_k=&\frac{\sum_{j=1}^{N}(\hat{\gamma}_{jk}y)}{\sum_{j=1}^{N}\hat{\gamma}_{jk}},\quad k=1,2,\ldots,K\end{aligned} \tag{16}$$

$$
\begin{aligned}
\frac{\partial Q}{\sigma_k^2} =& \sum_{j=1}^{N}[\hat{\gamma}_{jk}(-\frac{1}{2\sigma_k^2}+\frac{1}{2\sigma_k^4}(y-\mu_k)^2)]=0 \\
\sum_{j=1}^{N}[\hat{\gamma}_{jk}\frac{1}{\sigma_k^2}(y-\mu_k)^2] =& \sum_{j=1}^{N}\hat{\gamma}_{jk} \\
\frac{1}{\sigma_k^2}\sum_{j=1}^{N}[\hat{\gamma}_{jk}(y-\mu_k)^2] =& \sum_{j=1}^{N}\hat{\gamma}_{jk} \\
\sigma_k^2 =& \frac{\sum_{j=1}^{N}[\hat{\gamma}_{jk}(y-\mu_k)^2]}{\sum_{j=1}^{N}\hat{\gamma}_{jk}}, \quad k=1,2,\ldots,K
\end{aligned} \tag{17}
$$

对于 $\hat{\alpha}_k$，需满足 $\sum\limits_{k=1}^{K}\alpha_k=1$，则构造拉格朗日多项式：

$$
Q'=\sum_{k=1}^{K}\{n_k\log\alpha_k+\sum_{j=1}^{N}\hat{\gamma}_{jk}[\log(\frac{1}{\sqrt{2\pi}})-\log\sigma_k-\frac{1}{2\sigma_k^2}(y-\mu_k)^2]\}+\lambda(\sum_{k=1}^{K}\alpha_k-1) \tag{18}
$$

将式 18 对 α_k 求导并令导数为 0 得

$$
\begin{aligned}
\frac{\partial Q'}{\partial\alpha_k} =& \frac{n_k}{\alpha_k}+\lambda=0 \\
n_k+\lambda\alpha_k =& 0 \\
\sum_{k=1}^{K}n_k+\sum_{k=1}^{K}\lambda\alpha_k =& 0 \\
N+\lambda =& 0 \\
\lambda =& -N \\
\alpha_k =& \frac{n_k}{N}=\frac{\sum_{j=1}^{N}\hat{\gamma}_{jk}}{N}, \quad k=1,2,\ldots,K
\end{aligned} \tag{19}
$$

重复以上计算，直到对数似然值不再有明显的变化为止。

4 实验结果分析

4.1 生成数据

我们将类别数 k 设置为 3，均值分别为 $(2,7)$、$(6,2)$ 和 $(8,7)$，协方差矩阵分别为 $[[1,0],[0,2]]$、$[[2,0],[0,1]]$ 和 $[[1,0],[0,1]]$，每个类别的数目设置为 150，生成数据，结果如图 **1**所示。

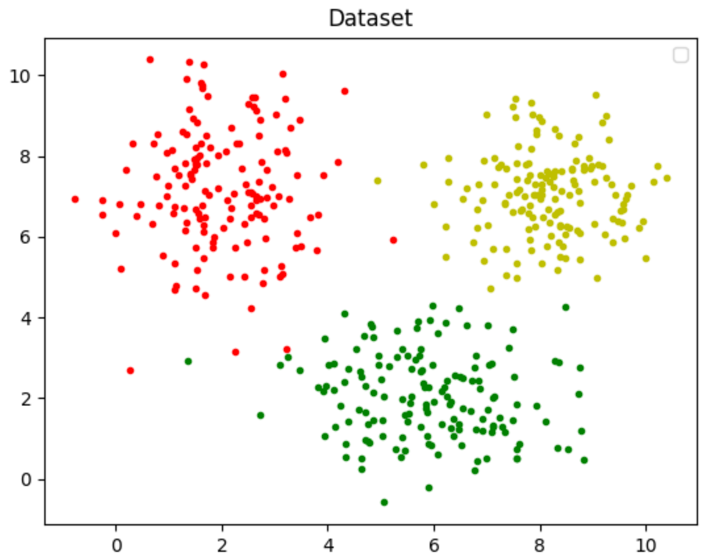

图 1

4.2 k-means 算法

我们数据的每个类别的数目设置为 200，其他参数不变，生成数据。我们将 k-means 算法的最大轮数设置为 100 轮，停止策略的系数设置为 10^{-7}，实验结果如图 2所示。

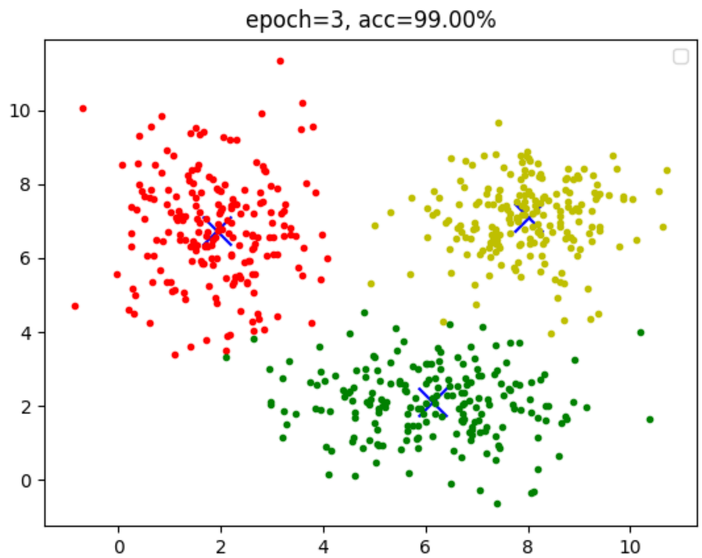

图 2

可以看到，在完成 3 轮迭代之后 k-means 算法收敛，得到最佳的中心点，预测准确率为 99.00%。k-means 算法收敛速度较快，准确率也比较高。

4.3 GMM 算法

我们数据的每个类别的数目设置为 200，其他参数不变，生成数据。我们将 GMM 算法的最大轮数设置为 100 轮，停止策略的系数设置为 10^{-7}，实验结果如图 3所示。

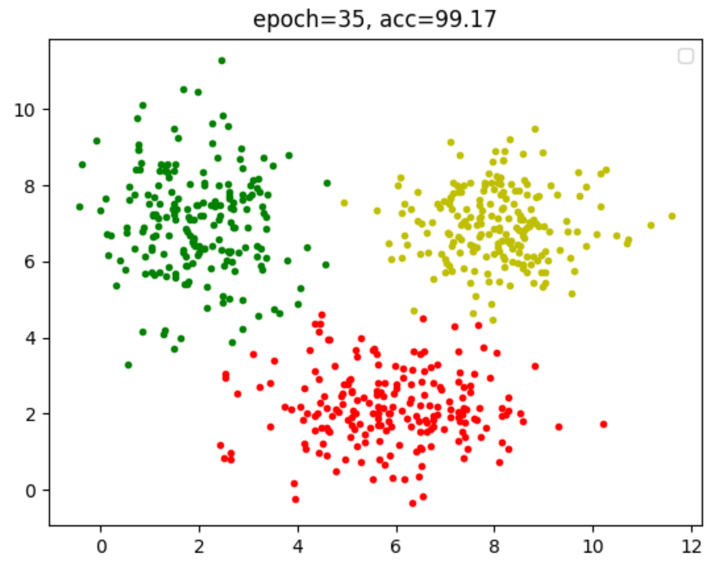

图 3

可以看到，在完成 35 轮迭代之后 GMM 算法收敛，得到最佳的中心点，预测准确率为 99.17%。GMM 算法收敛速度较快，准确率也比较高。

4.4 UCI 数据集

我们选用 UCI 数据集 Iris，该数据集共有 150 个数据，类别数为 3，分别选用 k-means 和 GMM 算法进行实验，实验结果表1所示。

算法	迭代次数	准确率
k-means	6	89.33%
GMM	61	96.67%

表 1

我们可以看到，在 UCI 数据集上，GMM 算法比 k-means 算法效果更好，但收敛速度较慢。

5 结论

k-means 算法较易理解与实现，在简单数据集上效果很好并且收敛较快；GMM 的实现复杂，推导繁琐，在各种数据集上都能取得良好的效果，收敛速度较 k-means 缓慢。

A 源代码（带注释）

A.1 生成数据

Listing 1: data.py

```python
import numpy as np
import matplotlib.pyplot as plt


def data_generator(k, size, loc, cov):
    # 生成数据
    X = np.zeros((k * size, loc.shape[1]))
    Y = np.zeros((k * size), dtype=np. int)
    for i in  range(k):
        X[i * size:(i + 1) * size] = np.random.multivariate_normal(loc[i], cov[i], size=(size))
        Y[i * size:(i + 1) * size] = i
    shuffle_ix = np.random.permutation(X.shape[0])
    X, Y = X[shuffle_ix], Y[shuffle_ix]
    return X, Y

def get_data_option(k=3, size=150):
    loc = np.array([[2, 7], [6, 2], [8, 7]])
    cov = np.array([[[1, 0], [0, 2]], [[2, 0], [0, 1]], [[1, 0], [0, 1]]])
    return k, size, loc, cov

# 画图
def show(dataset):
    X, Y = dataset
    k =  max(y for y in Y) + 1
    print(k)
    colors = ['r', 'g', 'y', 'b']
    for i in  range(k):
        ix = np.where(Y == i)
        plt.plot(X[ix, 0], X[ix, 1], '.', color=colors[i])
    plt.title('Dataset')
    plt.legend()
    plt.show()

if __name__ == '__main__':
    k, size, loc, cov = get_data_option()
    dataset = data_generator(k, size, loc, cov)
    show(dataset)
```

A.2 k-means 算法

Listing 2: kmeans.py

```python
import numpy as np
from data import *
from utils import *
from ucidata import *
import collections


def kmeans(X, k, epochs=100, epsilon=1e-7):
    c = X[np.random.choice( range(X.shape[0]), k, replace=False)]
    for epoch in  range(epochs):
        tags = np.zeros((X.shape[0]), dtype=np. int)
        for i in  range(X.shape[0]):
            _distances = [np.linalg.norm(x - X[i], 2) for x in c]
            tags[i] = _distances.index( min(_distances)) # 找到最近的簇
        new_c = np.zeros((k, X.shape[1]))
        for i in  range(X.shape[0]):
            new_c[tags[i]] += X[i]
        counter = collections.Counter(tags)
        new_c /= np.array([counter[tag] for tag in  range(k)]).reshape((k, 1)) # 均值
        if np.linalg.norm(new_c - c, 2) < epsilon:
            c = new_c
            break
        c = new_c
    return c, tags, epoch

# 画图
def show(dataset, c, tags, epoch):
    X, Y = dataset
    k =  max(y for y in Y) + 1
    colors = ['r', 'g', 'y', 'b']
    for i in  range(k):
        ix = np.where(Y == i)
        plt.plot(X[ix, 0], X[ix, 1], '.', color=colors[i])
    plt.scatter(c[:, 0], c[:, 1], c='b', marker="x", s=250)
    plt.title('epoch={}, acc={:.2f}%'. format(epoch, get_acc(dataset[1], tags) * 100))
    plt.legend()
    plt.show()

if __name__ == '__main__':
    k, size, loc, cov = get_data_option(size=200)
    dataset = data_generator(k, size, loc, cov)
    c, tags, epoch = kmeans(dataset[0], k)
    show(dataset, c, tags, epoch)

    file_path = './bezdekIris.data'
    dataset = uci_data_iris(file_path)
    c, tags, epoch = kmeans(dataset[0], k)
```

```
    print('epoch={}, acc={:.2f}%'. format(epoch, get_acc(dataset[1], tags) * 100))
```

A.3 GMM 算法

Listing 3: GMM.py

```
import numpy as np
from data import *
from utils import *
from ucidata import *

class GMM:
    def __init__(self, x, K, epochs=100, delta=1e-7):
        # x: data
        # K: number of clusters
        # epochs: 最大迭代次数
        # delta: 停止迭代的阈值
        self.delta = delta
        self.x = x
        self.K = K
        self.epochs = epochs
        self._sigma = np.array([0.1 * np.eye(self.x.shape[1])] * K)
        self._mu = x[np.random.choice( range(x.shape[0]), k, replace=False)]
        self._alpha = np.ones(k) * (1.0 / k)

    def E_step(self):
        self._gamma = np.zeros((self.x.shape[0], self.K))
        for j in  range(self.x.shape[0]):
            now_sum = 0
            for k in  range(self.K):
                self._gamma[j][k] = self._alpha[k] * phi_func(self.x[j], self._mu[k], self.
                    _sigma[k])
                now_sum += self._gamma[j][k]
            self._gamma[j] /= now_sum

    def M_step(self):
        _mu, _sigma, _alpha = np.zeros((self.K, self.x.shape[1])), np.array([0.1 * np.eye(self.
            x.shape[1])] * self.K), np.zeros((self.K))
        for k in  range(self.K):
            sum_gamma, sum_gamma_x, sum_gamma_x_mu = 0, 0, np.zeros((self.x.shape[1], self.x.
                shape[1]))
            for j in  range(self.x.shape[0]):
                sum_gamma += self._gamma[j][k]
                sum_gamma_x += self._gamma[j][k] * self.x[j]
                v = (self.x[j] - self._mu[k]).reshape(-1, 1)
                sum_gamma_x_mu += self._gamma[j][k] * np.dot(v, v.T)
            _mu[k] = sum_gamma_x / sum_gamma
```

```python
            _sigma[k] = sum_gamma_x_mu / sum_gamma
            _alpha[k] = sum_gamma  / self.x.shape[0]
        self._mu, self._sigma, self._alpha = _mu, _sigma, _alpha

    def GMM(self):
        likelihood = 0
        for epoch in  range(self.epochs):
            self.E_step()
            self.M_step()
            new_likelihood = log_likelihood(self.x, self._alpha, self._mu, self._sigma, self._gamma)
            if  abs(new_likelihood - likelihood) < self.delta:
                break
            likelihood = new_likelihood
        return np.argmax(self._gamma, axis=1), epoch

# 画图
def show(dataset, y_pred, epoch):
    X, Y = dataset
    k =  max(y for y in Y) + 1
    colors = ['r', 'g', 'y', 'b']
    for i in  range(k):
        ix = np.where(y_pred == i)
        plt.plot(X[ix, 0], X[ix, 1], '.', color=colors[i])
    plt.title('epoch={}, acc={:.2f}'. format(epoch, get_acc(dataset[1], y_pred) * 100))
    plt.legend()
    plt.show()

if __name__ == '__main__':
    k, size, loc, cov = get_data_option(size=200)
    dataset = data_generator(k, size, loc, cov)
    y_pred, epoch = GMM(dataset[0], k).GMM()
    show(dataset, y_pred, epoch)

    file_path = './bezdekIris.data'
    dataset = uci_data_iris(file_path)
    y_pred, epoch = GMM(dataset[0], k).GMM()
    print('epoch={}, acc={:.2f}%'. format(epoch, get_acc(dataset[1], y_pred) * 100))
```

A.4 解析 UCI 数据集

Listing 4: ucidata.py

```python
import numpy as np
import matplotlib.pyplot as plt

def uci_data_iris(file_path):
```

```python
    with  open(file_path, 'r') as f:
        label_map, label_map_counter = {}, 0
        lines = [l[:-1].split(',') for l in f.readlines()]
        for line in lines:
            x = line[-1]
            if x not in label_map.keys():
                label_map_counter += 1
                label_map[x] = label_map_counter
        np.random.shuffle(lines)
    X = np.array([[ float(l[i]) for i in  range( len(l) - 1)] for l in lines])
    Y = np.array([label_map[l[i]] - 1 for l in lines for i in  range( len(l) - 1,  len(l))])
    return X, Y

if __name__ == '__main__':
    file_path = './bezdekIris.data'
    dataset = uci_data_iris(file_path)
```

A.5 辅助代码

Listing 5: utils.py

```python
import numpy as np
from scipy.stats import multivariate_normal
import itertools

def phi_func(x, _mu, _sigma, delta=-1e7):
    res = multivariate_normal.pdf(x, mean=_mu, cov=_sigma)
    if  abs(res) < delta:
        res += delta
    return res

def log_likelihood(x, _alpha, _mu, _sigma, _gamma):
    now_sum = 0
    for k in  range(_mu.shape[0]):
        for j in  range(x.shape[0]):
            now_sum += _gamma[j][k] * np.log(_alpha[k])
            now_sum += _gamma[j][k] * np.log(phi_func(x[j], _mu[k], _sigma[k]))
    return now_sum

def get_acc(ys, y_preds):
    # 获取准确率
    ks =  list( set([x for x in ys]))
    news =  list(itertools.permutations(ks))
    acc_best = 0
    for new in news:
        acc = 0
        for i in  range( len(ys)):
```

```
            if ys[i] == new[y_preds[i]]:
                acc += 1
        acc /=  len(ys)
        acc_best =  max(acc_best, acc)
    return acc_best
```