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： K-means 和 GMM 模型

学号：
姓名：

# 一、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。

# 二、实验要求及实验环境

## 2.1 要求：

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）。

## 2.2 验证：

(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。

## 2.3 实验环境：

PyCharm 2020.3.2 x64
Python 3.8　numpy1.20.3　matplotlib 3.4.2

# 三、设计思想（本程序中的用到的主要算法及数据结构）

## 3.1 K-means 理论推导：

假设数据集为$\{x_1,\ldots,x_N\}$, 且维度为 D，我们的目标是将数据集划分为K个类，假设K已给定。

直观上，认为一组数据点构成的一个聚类中，聚类内部的点应小于数据点与外部的点之间的距离，假设$\mu_k$为聚类的中心。

定义目标函数，该函数表示每个数据点和它被分到的类的中心$\mu_k$的距离的平方和：

$$\mathrm{J}=\sum_{n=1}^{N}\sum_{k=1}^{K}r_{nk}\|x_n-\mu_k\|^2 \tag{1}$$

当且仅当$x_n$被分到类别k的时候，$r_{nk}=1.$

使用迭代的方式，使目标函数J达到最小值，分别对应于$r_{nk}$和$\mu_k$的最优化。在第一阶段中，固定$\mu_k$，关于$r_{nk}$最小化J；第二阶段中，固定$r_{nk}$，关于$\mu_k$最小化J。不断重复上述两个过程，直到收敛。这两步分别对应于EM算法中的E(期望)和M(最大化)的步骤。

第一阶段：

由于不同的与n相关的项是独立的，因此可以分别对n项进行最优化，如下式：

$$r_{nk}=\begin{cases}1 & 若k=\arg min_j\left\|x_n-\mu_j\right\|^2\\0 & 其他情况\end{cases} \tag{2}$$

第二阶段：

当$r_{nk}$固定时，关于$\mu_k$的最优化。由于J是关于$\mu_k$的二次函数，令它关于$\mu_k$的导数为0，即可达到最小值，因此对公式(1)关于$\mu_k$求导，可得：

$$\frac{\partial}{\partial\mu_k}\mathrm{J}=2\sum_{n=1}^{N}r_{nk}(x_n-\mu_k) \tag{3}$$

令公式(3)为0，可得：

$$2\sum_{n=1}^{N}r_{nk}(x_n-\mu_k)=0 \tag{4}$$

化简可得：

$$\mu_k=\frac{\sum_{n=1}^{N}r_{nk}x_n}{\sum_{n=1}^{N}r_{nk}} \tag{5}$$

公式(5)中，分母等于聚类k中数据点的数量，因此公式(5)的含义为，$\mu_k$等于类别k的所有数据点的均值。

重复为数据分类，重新计算聚类均值的步骤重复进行，直到聚类的分配不改变或直到迭代次数超过某个最大值。

$\mathrm{K-means}$算法的收敛性：每个阶段都减小了目标函数J的值。

## 3.2 高斯混合模型与EM算法理论推导：

每个高斯分布对应$\mathrm{N}(\mu_k,\Sigma_k)$，因此高斯混合概率分布可以写成K个多维高斯分布的线性叠加的形式，即：

$$\mathrm{P(x)}=\sum_{k=1}^{K}\pi_k\mathrm{N}(\mu_k,\Sigma_k) \tag{6}$$

且有

$$0\le\pi_k\le1 \tag{7}$$

$$\sum_{k=1}^{K}\pi_k=1 \tag{8}$$

引入K维二值隐变量z，且满足以下条件：

$$\mathrm{P}(z_k=1)=\pi_k \tag{9}$$

$$0\le\pi_k\le1 \tag{10}$$

因此概率分布也可以写为

$$\mathrm{P(z)}=\prod_{k=1}^{K}\pi_k^{z_k} \tag{11}$$

此时x所对应z的后验概率为：

$$\mathrm{P}(\mathrm{x}|\mathrm{z}) = \prod_{k=1}^{K} \mathrm{N}(\mathrm{x}|\mu_k, \Sigma_k)^{z_k} \tag{12}$$

当给定$z_k$特定的值时，x的条件概率分布是一个高斯分布：

$$\mathrm{P}(\mathrm{x}|z_k = 1) = \mathrm{N}(\mathrm{x}|\mu_k, \Sigma_k) \tag{13}$$

因此可以从x的联合概率分布求得x的联合概率分布，即

$$\mathrm{P}(\mathrm{x}) = \sum_{z} P(z)P(x|z) \tag{14}$$

$$\mathrm{P}(\mathrm{x}) = \sum_{k=1}^{k} \pi_k \mathrm{N}(x|\mu_k, \Sigma_k) \tag{15}$$

此时得出的公式(15)与公式(6)完全一致。

假设数据集为$\{x_1, \dots, x_N\}$，对于每一个观测数据点$x_n$都对应一个$z_n$，并且样本$x_i$为d维向量，且由d维高斯分布生成。

因此有

$$\mathrm{N}(x_i|\mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)} \tag{16}$$

在给定观测值$x_i$的时候，将$\gamma(z_k)$看作$x_i$对应的后验概率，它的值可以通过贝叶斯定理得出：

$$\gamma(z_k) = \mathrm{p}(z_k = 1|\mathrm{x}) = \frac{P(z_k = 1)P(x|z_k = 1)}{\sum_{j=1}^{K} P(z_j = 1)P(x|z_j = 1)} \tag{17}$$

$$= \frac{\pi_k \mathrm{N}(\mathrm{x}|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathrm{N}(\mathrm{x}|\mu_j, \Sigma_k)} \tag{18}$$

因此对于一个样本点$x_i$，若要将其归为$1, \dots, \mathrm{K}$簇中的一个，需要找到令公式(18)最大的k，即所归的簇的序号，可以表示为：

$$\mathrm{k} = \arg max_k\, \mathrm{p}(z_k = 1|x_i) \tag{19}$$

$$\mathrm{k} = \arg max_k \frac{\pi_k \mathrm{N}(\mathrm{x}|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathrm{N}(\mathrm{x}|\mu_j, \Sigma_k)} \tag{20}$$

因此，找到参数$\pi_k$、$\mu_k$、$\Sigma_k$是GMM聚类的关键。

将数据集表示为一个$\mathrm{N} * \mathrm{d}$的矩阵X，其中第n行为$x_n^T$，类似的对应的隐含变量会被表示为一个$\mathrm{N} * \mathrm{K}$的矩阵Z，它的行为$z_n^T$。

由公式(6)可得，对数似然函数为：

$$\mathrm{L}(\mathrm{X}) = \ln P(\boldsymbol{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{21}$$

$$= \ln \prod_{n=1}^{N} \left( \sum_{k=1}^{K} \pi_k \mathrm{N}(\boldsymbol{x_n}|\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k}) \right) \tag{22}$$

$$= \sum_{n=1}^{N} \ln\left(\sum_{k=1}^{K} \pi_k \mathrm{N}(\boldsymbol{x_n}|\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})\right) \tag{23}$$

公式(23)对$\boldsymbol{\mu_k}$求导：

$$\frac{\partial}{\partial \boldsymbol{\mu_k}} \mathrm{L(X)} = \sum_{n=1}^{N} \frac{\frac{\partial}{\partial \mu_k} \sum_{k=1}^{K} \pi_k \mathrm{N}(\boldsymbol{x_n}|\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})}{\sum_{k=1}^{K} \pi_k \mathrm{N}(\boldsymbol{x_n}|\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})} \tag{24}$$

$$= \sum_{n=1}^{N} \frac{\frac{\partial}{\partial \mu_k}\left(\pi_1 \mathrm{N}(\boldsymbol{x_n}|\boldsymbol{\mu_1}, \boldsymbol{\Sigma_1}) + \cdots + \pi_k \mathrm{N}(\boldsymbol{x_n}|\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k}) + \cdots + \pi_k \mathrm{N}(\boldsymbol{x_n}|\boldsymbol{\mu_K}, \boldsymbol{\Sigma_K})\right)}{\sum_{k=1}^{K} \pi_k \mathrm{N}(\boldsymbol{x_n}|\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})} \tag{25}$$

$$= \sum_{n=1}^{N} \frac{\frac{\partial}{\partial \mu_k} \pi_k \mathrm{N}(\boldsymbol{x_n}|\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})}{\sum_{k=1}^{K} \pi_k \mathrm{N}(\boldsymbol{x_n}|\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})} \tag{26}$$

多维高斯分布对均值μ求导：

$$\mathrm{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{n}{2}}} \frac{1}{|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})} \tag{27}$$

$$\frac{\partial \mathrm{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})}{\partial \mu} = \frac{\partial}{\partial \mu}\left(\frac{1}{(2\pi)^{\frac{n}{2}}} \frac{1}{|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}\right) \tag{28}$$

$$= \frac{\partial}{\partial \mu}\left(\frac{1}{(2\pi)^{\frac{n}{2}}} \frac{1}{|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}\left(x^T \Sigma^{-1} x - \mu^T \Sigma^{-1} x - x^T \Sigma^{-1} \mu + \mu^T \Sigma^{-1} \mu\right)}\right) \tag{29}$$

$$= \frac{1}{(2\pi)^{\frac{n}{2}}} \frac{1}{|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})} * \left(-\frac{1}{2}\right)\left(-(x^T \Sigma^{-1})^T - \Sigma^{-1} x + \left(\Sigma^{-1} + {\Sigma^{-1}}^T\right)\mu\right) \tag{30}$$

$$= \frac{1}{(2\pi)^{\frac{n}{2}}} \frac{1}{|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})} * \left(-\frac{1}{2}\right)\left(-\left({\Sigma^{-1}}^T + \Sigma^{-1}\right) + \left(\Sigma^{-1} + {\Sigma^{-1}}^T\right)\mu\right) \tag{31}$$

由于各个维度相关系数为 0，因此有$\Sigma$为对称矩阵，故有：

$$\frac{\partial \mathrm{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})}{\partial \mu} = \mathrm{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \Sigma^{-1}(x - \mu) \tag{32}$$

将结果代入公式(26)，因此我们有：

$$\frac{\partial}{\partial \boldsymbol{\mu_k}} \mathrm{L(X)} = \sum_{n=1}^{N} \frac{\pi_k \mathrm{N}(\boldsymbol{x_n}|\boldsymbol{\mu}, \boldsymbol{\Sigma})}{\sum_{j=1}^{K} \pi_j \mathrm{N}\left(\boldsymbol{x_n}\middle|\boldsymbol{\mu_j}, \boldsymbol{\Sigma_j}\right)} \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x_n} - \boldsymbol{\mu_k}) \tag{33}$$

由公式(18)，我们有

$$\frac{\partial}{\partial \boldsymbol{\mu_k}} \mathrm{L(X)} = \sum_{n=1}^{N} \gamma(z_k) \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x_n} - \boldsymbol{\mu_k}) \tag{34}$$

令公式(34)为 0，可得

$$\sum_{n=1}^{N} \gamma(z_k) \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x_n} - \boldsymbol{\mu_k}) = 0 \tag{35}$$

因此，我们有

$$\boldsymbol{\mu_k} = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) \boldsymbol{x_n}}{\sum_{n=1}^{N} \gamma(z_{nk})} \tag{36}$$

令公式(23)对$\Sigma_k$并令其为 0 可得：

$$\frac{\partial}{\partial \boldsymbol{\Sigma}} \mathrm{L}(\mathrm{X}) = 0 \tag{37}$$

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) (\boldsymbol{x_n} - \boldsymbol{\mu_k})(\boldsymbol{x_n} - \boldsymbol{\mu_k})^{\boldsymbol{T}} \tag{38}$$

由公式(8)可得，当前优化为有约束的优化，因此使用拉格朗日乘数法:

$$\mathrm{L}(\mathrm{X}) = \sum_{n=1}^{N} \ln \left( \sum_{k=1}^{K} \pi_k \mathrm{N}(\boldsymbol{x_n} | \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k}) \right) + \lambda \left( \sum_{k=1}^{K} \pi_k - 1 \right) \tag{39}$$

使用公式(39)对$\pi_k$求导可得，

$$\frac{\partial}{\partial \boldsymbol{\Sigma}} \mathrm{L}(\mathrm{X}) = \sum_{n=1}^{N} \frac{\mathrm{N}(\boldsymbol{x_n} | \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})}{\sum_{k=1}^{K} \pi_k \mathrm{N}(\boldsymbol{x_n} | \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})} + \lambda \tag{40}$$

令公式(40)为 0，有：

$$\sum_{n=1}^{N} \frac{\mathrm{N}(\boldsymbol{x_n} | \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})}{\sum_{k=1}^{K} \pi_k \mathrm{N}(\boldsymbol{x_n} | \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})} + \lambda = 0 \tag{41}$$

对公式(42)同乘$\pi_k$，可得

$$\pi_k \sum_{n=1}^{N} \frac{\mathrm{N}(\boldsymbol{x_n} | \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})}{\sum_{k=1}^{K} \pi_k \mathrm{N}(\boldsymbol{x_n} | \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})} + \pi_k \lambda = 0 \tag{42}$$

将公式(18)代入得

$$\sum_{n=1}^{N} \gamma(z_{nk}) + \pi_k \lambda = 0 \tag{43}$$

将K个等式相加，可得

$$\sum_{k=1}^{K} \left( \pi_k \sum_{n=1}^{N} \frac{\mathrm{N}(\boldsymbol{x_n} | \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})}{\sum_{k=1}^{K} \pi_k \mathrm{N}(\boldsymbol{x_n} | \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})} \right) + \sum_{k=1}^{K} \pi_k \lambda = 0 \tag{44}$$

$$\sum_{k=1}^{K} \left( \sum_{n=1}^{N} \frac{\pi_k \mathrm{N}(\boldsymbol{x_n} | \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})}{\sum_{k=1}^{K} \pi_k \mathrm{N}(\boldsymbol{x_n} | \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})} \right) + \sum_{k=1}^{K} \pi_k \lambda = 0 \tag{45}$$

将累加符号调换位置

$$\sum_{k=1}^{N} \left( \frac{\sum_{k=1}^{K} \pi_k \mathrm{N}(\boldsymbol{x_n} | \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})}{\sum_{k=1}^{K} \pi_k \mathrm{N}(\boldsymbol{x_n} | \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})} \right) + \sum_{k=1}^{K} \pi_k \lambda = 0 \tag{46}$$

可得

$$\lambda = -\mathrm{N} \tag{47}$$

因此有

$$\pi_k = \frac{1}{\mathrm{N}}\sum_{n=1}^{N}\gamma(z_{nk}) \tag{48}$$

EM算法的完整描述：

E步骤：使用当前的参数计算每个样本$x_i$的$\gamma(z_{ik})$。

$$\gamma(z_{ik}) = \frac{\pi_k \mathrm{N}(x_i|\mu_k,\Sigma_k)}{\sum_{j=1}^{K}\pi_j \mathrm{N}\left(x_i\middle|\mu_j,\Sigma_k\right)} \tag{18}$$

M步骤：使用当前的$\gamma(z_{ik})$重新估计参数。

$$\mu_k^{新} = \frac{\sum_{n=1}^{N}\gamma(z_{nk})\boldsymbol{x_n}}{\sum_{n=1}^{N}\gamma(z_{nk})} \tag{49}$$

$$\Sigma_k^{新} = \frac{\sum_{n=1}^{N}\gamma(z_{nk})\left(\boldsymbol{x_n}-\mu_k^{新}\right)\left(\boldsymbol{x_n}-\mu_k^{新}\right)^{\boldsymbol{T}}}{\sum_{n=1}^{N}\gamma(z_{nk})} \tag{50}$$

$$\pi_k^{新} = \frac{1}{\mathrm{N}}\sum_{n=1}^{N}\gamma(z_{nk}) \tag{51}$$

计算对数似然函数

$$\ln P(\boldsymbol{X}|\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma}) = \sum_{n=1}^{N}\ln\left(\sum_{k=1}^{K}\pi_k \mathrm{N}(\boldsymbol{x_n}|\boldsymbol{\mu_k},\boldsymbol{\Sigma_k})\right) \tag{52}$$

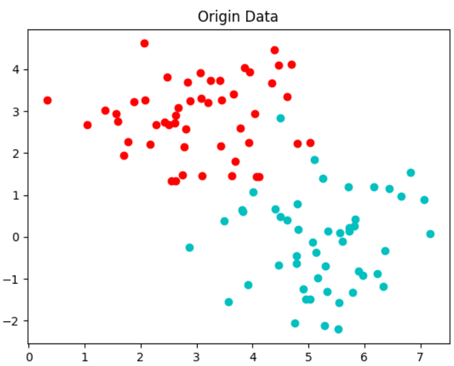
若对数函数没有满足收敛的准则，则继续重复E、M步骤。

# 四、实验结果与分析

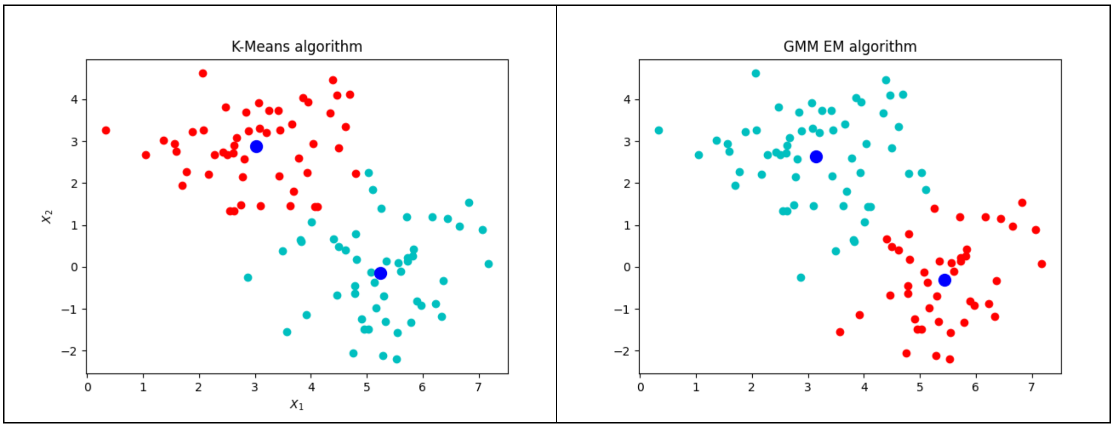
## 4.1 生成数据实验：

实验设定生成的数据符合二维高斯分布，二维方差均为 0.1。

(1)生成的数据的$\mathrm{K}=2$时。

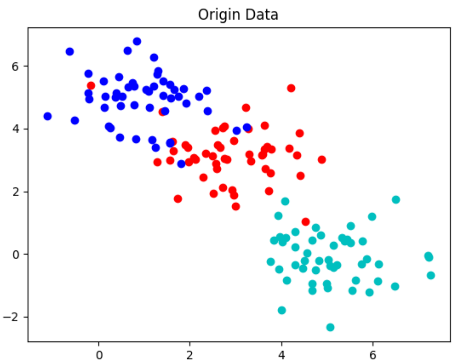



如图为生成的原始数据，两堆数据分别符合两个二维高斯分布。

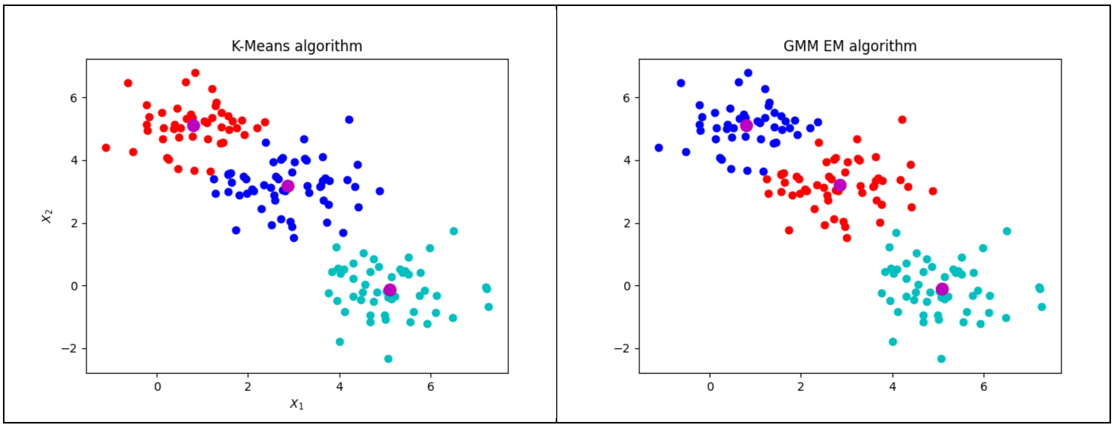



左图为使用$K-Means$算法后分类的结果，右图为使用$GMM\ EM$算法后分类的结果。可以看到当$K=2$的时候，两种算法都有较好的分类效果。

(2)生成的数据的$K=3$时。




如图为生成的原始数据，三堆数据分别符合三个二维高斯分布。




左图为使用$K-Means$算法后分类的结果，右图为使用$GMM\ EM$算法后分类的结果。可以看到当$K=3$的时候，两种算法都有较好的分类效果。和原图相比，两个分类结果都有比较好的展示，没有出现过拟合的情况。

## 4.2 UCI 数据实验：

本阶段采用 UCI 网站的鸢尾花数据集。

由于 K-Means 和 GMM EM 为无监督机器学习，因此需要先定义好簇和其所属类别的映射关系。

对簇的类别进行排列组合，由于我们已经有$K$的值，因此可以很清晰的定义排列组合的次数为$K!$次。

我们假设$K = 3$，因此此时有$[[0,1,2],[0,2,1],[1,2,0],[1,0,2],[2,0,1],[2,1,0]]$六种排列组合。

接下来定义一个字典的列表，将上述组合中的每一个分别用第i个元素和i对应起来，就得到了簇和Y中的种类的关系的映射。

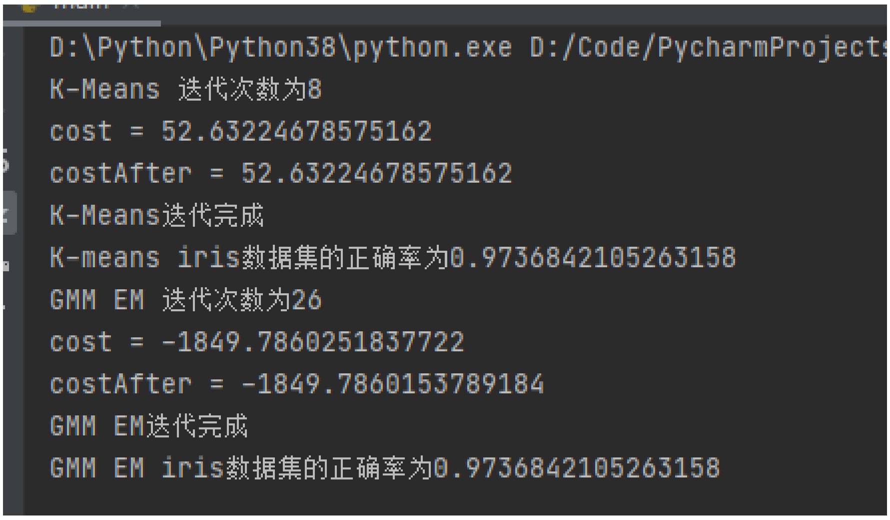
以$[0,2,1]$为例，所得到的字典为

$$dict = \{0{:}\,1, 2{:}\,1, 1{:}\,2\}$$

通过这样的映射关系，我们遍历训练集中所有得到的簇的序号到数据集中的预测数据之间的关系，并进行比较，记录下所有匹配的个数。

将所有字典循环后，我们即可以得到使训练好的模型和原数据集最匹配的映射关系。选取这样的映射关系，我们对测试集进行计算，当测试集预测出的簇的序号映射到的Y中的序号与其本来Y的序号对应，则视为预测正确，否则视为预测错误。

这样我们就有了对数据集正确率的说明。



```
D:\Python\Python38\python.exe D:/Code/PycharmProjects
K-Means 迭代次数为8
cost = 52.63224678575162
costAfter = 52.63224678575162
K-Means迭代完成
K-means iris数据集的正确率为0.9736842105263158
GMM EM 迭代次数为26
cost = -1849.7860251837722
costAfter = -1849.7860153789184
GMM EM迭代完成
GMM EM iris数据集的正确率为0.9736842105263158
```

实际实验中，K − Means和GMM EM算法都能够较好的对鸢尾花数据集分类。K − Means迭代8次，GMM EM算法迭代26次。测试集共38个数据，K − Means和GMM EM的正确分类37个数据，正确率为97.368%，有较好的预测表现。

## 五、结论

K-Means 和 GMM EM 都有较好的表现，可以快速收敛，并且二者的分类性能都较好。但是 K-Means 有可能找到的是局部最优的聚类，而不是全局最优的聚类。并且在样本维度过高时，可能耗费大量的计算。

高斯混合模型不仅可以给出一个样本属于某类的概率，还可以用于概率密度的估计，用于生成新的样本点。

# 六、参考文献

[1] Pattern Recognition and Machine Learning
[2] 机器学习 周志华著 北京：清华大学出版社，2016 年 1 月.

# 七、附录：源代码（带注释）

main.py

```python
from KMeans import *
from generateData import *
from GMM import *
from readFromFile import *

if __name__ == '__main__':
    # # 类别个数
    # K = 3
    # # 迭代结束次数
    # iter = 200
    # # 生成数据
    # X1, X2 = generateData(K)
    # KMeansClusterData = makeDataForKmeans(X1, X2)
    # GMMClusterData = makeDataForGMM(X1, X2)
    # # KMeans 算法
    # DataCenter, centers = kMeans(K, iter, KMeansClusterData)
    # drawKmeansClassify(KMeansClusterData, DataCenter, centers, K)
    # # GMM EM 算法
    # Gamma, mus,sigmas,alpha = GMMem(K, iter, GMMClusterData)
    # drawGMM(GMMClusterData, Gamma, mus)



    iterFile = 200
    # 从文件中读取数据
    X, Y, K = readFromFile("./DataSet/iris.csv")
    KMeansTrain, KMeansTest, KMeansYTrain, KMeansYtest = makeFileDataForKmeans(X, Y)
    GMMTrain, GMMTest, GMMYTrain, GMMYTest = makeFileDataForGMM(X, Y)

    # KMeans 算法
    DataCenterFile, centersFile = kMeans(K, iterFile, KMeansTrain)
    TestKMeansAccuracy(DataCenterFile,centersFile,KMeansYTrain,KMeansYtest,Y,KMeansTest,K)
```

```


    # GMM EM 算法
    GammaFile, mus,sigmas,alpha = GMMem(K, iterFile, GMMTrain)
    centersTrain = np.zeros(GammaFile.shape[0])
    for i in range(GammaFile.shape[0]):
        centersTrain[i] = np.array(GammaFile[i]).argmax()
    TestGMMAccuracy(centersTrain,mus,sigmas,alpha,GMMYTrain,GMMYTest,Y,GMMTest,K)
```

generateData.py

```
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.model_selection import train_test_split

color = ['c', 'r', 'b', 'm', 'y', 'k', 'g', 'w']
muList = [[5, 0], [3, 3], [1, 5], [4, 2], [1, 3], [2, 3]]


def generate2DimensionalData(Mu1, Mu2, cov11, cov12, cov21, cov22, Num, noiseSigma1, noiseSigma2):
    """
    生成二维高斯分布数据
    :param Mu1: 维度 1 的平均值
    :param Mu2: 维度 2 的平均值
    :param cov11: 维度 1 的方差
    :param cov12: 维度 12 的协方差
    :param cov21: 维度 21 的协方差
    :param cov22: 维度 2 的方差
    :param Num: 生成样本点的个数
    :param noiseSigma1: 维度 1 噪声的方差
    :param noiseSigma2: 维度 2 噪声的方差
    :return: 分别返回两个维度的样本点
    """
    mean = np.array([Mu1, Mu2])
    cov = np.array([[cov11, cov12], [cov21, cov22]])
    Data = np.random.multivariate_normal(mean, cov, Num)
    X1 = Data[:, 0]
    X2 = Data[:, 1]
    # 添加高斯噪声
    GuassNoise1 = np.random.normal(0, scale=noiseSigma1, size=Num)
    GuassNoise2 = np.random.normal(0, scale=noiseSigma2, size=Num)
```

```python
    X1 = X1 + GuassNoise1
    X2 = X2 + GuassNoise2
    return X1, X2


def generateData(K):
    """
    生成 K 堆二维高斯分布数据
    :param K: 生成样本点的堆数
    :return: X1、X2 的列表
    """
    X1 = np.empty(0)
    X2 = np.empty(0)
    plt.title("Origin Data")
    number = 50
    # Y = np.empty(0)
    for i in range(K):
        # X1gen, X2gen = generate2DimensionalData(random.randint(0, 5), random.randint(0, 5), 1, 0, 0, 1, number, 0.1, 0.1)
        X1gen, X2gen = generate2DimensionalData(muList[i][0], muList[i][1], 1, 0, 0, 1, number, 0.1, 0.1)
        # Y = np.hstack((Y,np.ones(number)*i))
        X1 = np.hstack((X1, X1gen))
        X2 = np.hstack((X2, X2gen))
        plt.scatter(X1gen, X2gen, c=color[i])
    plt.show()
    return X1, X2


def makeDataForKmeans(X1, X2):
    """
    将数据处理，适合 KMeans 算法
    :param X1: 一行数据
    :param X2: 一行数据
    :return: 两行数据
    """
    X = np.vstack((X1, X2))
    return X


def KmeansTrainTestSplit(ClusterData, Y):
    clusterData = np.transpose(ClusterData)
    X_train, X_test, Y_train, Y_test = train_test_split(clusterData, Y)
    X_train = np.transpose(X_train)
```

```
    X_test = np.transpose(X_test)
    return X_train, X_test, Y_train, Y_test


def makeDataForGMM(X1, X2):
    X1 = np.reshape(X1, (-1, 1))
    X2 = np.reshape(X2, (-1, 1))
    X = np.column_stack((X1, X2))
    return X
```

**KMeans.py**

```
import numpy as np
import matplotlib.pyplot as plt
import math
from generateData import color
import itertools

def initCenters(KMeansData, K):
    """
    初始化中心点
    :param KMeansData:所有数据点
    :param K: 分成类别的个数
    :return:
    """
    centerX1 = np.random.choice(KMeansData[0], K)
    centerX2 = np.random.choice(KMeansData[1], K)
    center = np.vstack((centerX1, centerX2))
    return center


def kMeansCost(KMeansData, centers, DataCenter):
    """
    计算当前所有 KMeansData 中的所有点与 centers 所有点的代价函数的值
    :param KMeansData: 所有 KMeansData 的数据点
    :param centers: 中心点
    :param DataCenter: 每个点对应的中心点
    :return:
    """
    cost = 0
    for i in range(KMeansData.shape[1]):
        X1 = KMeansData[0][i]
        X2 = KMeansData[1][i]
        centerX1 = centers[0][int(DataCenter[i])]
```

```python
        centerX2 = centers[1][int(DataCenter[i])]
        cost = cost + math.sqrt((X1 - centerX1) * (X1 - centerX1) + (X2 - centerX2) * (X2 - centerX2))
    return cost


def calculateDistanceCenter(X1, X2, center):
    """
    计算当前的点与中心点的距离，选择最近的类
    :param X1: 点的坐标
    :param X2: 点的坐标
    :param center: 中心点数组
    :return: 点的类
    """
    cost = float("inf")
    minCost = cost
    Xclass = 0
    for i in range(center.shape[1]):
        cost = 0
        cost = cost + math.sqrt((X1 - center[0][i]) * (X1 - center[0][i]))
        cost = cost + math.sqrt((X2 - center[1][i]) * (X2 - center[1][i]))
        if cost < minCost:
            minCost = cost
            Xclass = i
    return Xclass


def assignClass(KMeansData, Centers):
    """
    为 KMeansData 每个数据分配中信
    :param KMeansData: 数据
    :param Centers: 中心点
    :return: 为每个点分配离它最近的中心点的序号
    """
    center = np.zeros(KMeansData.shape[1])
    for i in range(KMeansData.shape[1]):
        center[i] = calculateDistanceCenter(KMeansData[0][i], KMeansData[1][i], Centers)
    return center


def updateCenters(KMeansData, centers, K):
    """
    更新中心点 K 个中心点
```

```python
    :param KMeansData:数据
    :param centers:每个数据的中心点
    :param K:类别个数
    :return:更新新的中心点
    """
    UpdateCenters = np.zeros([2, K])
    list1 = []
    list2 = []
    for i in range(K):
        for j in range(KMeansData.shape[1]):
            if centers[j] == i:
                list1.append(KMeansData[0][j])
                list2.append(KMeansData[1][j])
        x1 = 0
        x2 = 0
        for k in range(len(list1)):
            x1 = x1 + list1[k]
            x2 = x2 + list2[k]

        if x1 == 0 and x2 == 0:
            print("此时无点离他最近")
            UpdateCenters[0][i] = 0
            UpdateCenters[1][i] = 0
            # randomNum = random.randint(0, KMeansData.shape[1]-1)
            # UpdateCenters[0][i] = KMeansData[0][randomNum]
            # UpdateCenters[1][i] = KMeansData[1][randomNum]
        else:
            x1 = x1 / len(list1)
            x2 = x2 / len(list2)
            UpdateCenters[0][i] = x1
            UpdateCenters[1][i] = x2
        list1 = []
        list2 = []
    return UpdateCenters


def kMeans(K, iter, ClusterData):
    """
    调用 Kmeans 算法
    :param K: 类别个数
    :param iter: 迭代次数
    :param ClusterData: 实验数据
    :return:
    """
```

```
    # 初始化开始时候的 K 个中心点
    centers = initCenters(ClusterData, K)
    costAfter = 0
    DataCenter = np.zeros(ClusterData.shape[1])
    for i in range(iter):
        cost = costAfter
        # 为每个点分配中心
        DataCenter = assignClass(ClusterData, centers)
        # 更新中心点
        centers = updateCenters(ClusterData, DataCenter, K)
        costAfter = kMeansCost(ClusterData, centers, DataCenter)
        if (costAfter == cost):
            print("K-Means 迭代次数为{}".format(i))
            print("cost = {}".format(cost))
            print("costAfter = {}".format(costAfter))
            print("K-Means 迭代完成")
            # drawKmeansClassify(ClusterData, DataCenter, centers,K)
            break
    return DataCenter, centers


def drawKmeansClassify(KMeansData, DataCenter, centers, K):
    """
    为 K-Means 算法生成的结果画图
    :param KMeansData:KMeans 的数据
    :param DataCenter:每个点对应的中心
    :param K:类别的个数
    """
    plt.title("K-Means algorithm")
    for i in range(K):
        listX1 = []
        listX2 = []
        for j in range(KMeansData.shape[1]):
            if DataCenter[j] == i:
                listX1.append(KMeansData[0][j])
                listX2.append(KMeansData[1][j])
        plt.scatter(listX1, listX2, c=color[i])
    plt.scatter(centers[0], centers[1], c=color[K], s=100)
    plt.xlabel("$X_{1}$")
    plt.ylabel("$X_{2}$")
    plt.show()

```

```python
def TestKMeansAccuracy(centersTrain, centers, YTrain, YTest,Y,KMeansTest, K):
    # 簇号
    Ytest = np.zeros(KMeansTest.shape[1])
    for i in range(KMeansTest.shape[1]):
        Ytest[i] = calculateDistanceCenter(KMeansTest[0][i], KMeansTest[1][i], centers)
    # Dict 簇到原来 y 的映射
    Dict = KMeansClusterMap(K, YTrain,centersTrain,Y)
    sum = 0
    for i in range(YTest.shape[0]):
        Ytest[i] = Dict[Ytest[i]]
    for i in range(YTest.shape[0]):
        sum = sum +1
    Accuracy = sum/Ytest.shape[0]
    print("K-means iris 数据集的正确率为{}".format(Accuracy))

def KMeansClusterMap(K,YTrain, dataCenters,Y):
    Set = list(set(Y))
    List = []
    for i in range(K):
        List.append(i)
    listArray = list(itertools.permutations(List,K))
    MapAccuracyDict = np.zeros(len(listArray))
    listDict = []
    for j in range(len(listArray)):
        # 簇号
        Dict = {}
        for k in range(len(listArray[j])):
            Dict[listArray[j][k]] = Set[k]
        listDict.append(Dict)
        MapAccuracyDict[j] = KMeansMapAccuracy(Dict,YTrain,dataCenters)
    return listDict[np.array(MapAccuracyDict).argmax()]

def KMeansMapAccuracy(Dict,YTrain,dataCenters):
    sum = 0
    for i in range(YTrain.shape[0]):
        if Dict[dataCenters[i]] == YTrain[i]:
            sum = sum+1
    return sum/YTrain.shape[0]
```

**GMM.py**

```python
import numpy as np
import random
import matplotlib.pyplot as plt
import math
from scipy.stats import multivariate_normal
from generateData import color
import itertools
def GMMem(K, iter, ClusterData):
    mus, sigmas, alpha = initParams(ClusterData, K)
    costAfter = 0
    gamma = 0
    for i in range(iter):
        cost = costAfter
        gamma = getExpectation(ClusterData,mus,sigmas,alpha,K)
        mus, sigmas, alpha = maximize(ClusterData,gamma,K)
        costAfter = calculateMLE(ClusterData,mus, sigmas, alpha ,K)
        if abs(costAfter - cost)<=1e-5:
            print("GMM EM 迭代次数为{}".format(i))
            print("cost = {}".format(cost))
            print("costAfter = {}".format(costAfter))
            print("GMM EM 迭代完成")
            break
    return gamma,mus,sigmas,alpha


def calculateMLE(ClusterData,mus, sigmas, alpha, K):
    """
    计算似然函数
    :param ClusterData:所有数据点
    :param mus:多维高斯分布均值矩阵
    :param sigmas:多维高斯分布的协方差矩阵
    :param alpha:每个簇的权重
    :param K:簇的个数
    :return:MLE 的值
    """
    iterN = 0
    for i in range(ClusterData.shape[0]):
        iterK = 0
        for k in range(K):
            iterK = iterK+alpha[k]*multivariate_normal.pdf(ClusterData[i], mus[k], sigmas[k])
        iterN = iterN+  math.log(iterK,math.e)
```

```
    return iterN


def getExpectation(ClusterData, mus, sigmas, alpha, K):
    """
    返回 gamma 矩阵
    :param ClusterData:所有数据点
    :param mus:多维高斯分布均值矩阵
    :param sigmas:多维高斯分布的协方差矩阵
    :param alpha:每个簇的权重
    :param K:簇的个数
    :return:gamma 矩阵
    """
    gamma = np.zeros((ClusterData.shape[0], K))
    for i in range(ClusterData.shape[0]):
        gamma_sum = 0
        for j in range(K):
            gamma[i][j] = alpha[j] * multivariate_normal.pdf(ClusterData[i], mus[j], sigmas[j])
            gamma_sum = gamma_sum + gamma[i][j]
        for j in range(K):
            gamma[i][j] = gamma[i][j] / gamma_sum
    return gamma



def maximize(ClusterData, gamma, K):
    """
    M Step
    :param ClusterData:所有数据点
    :param gamma:gamma 矩阵
    :param K:所有的类别
    :return:m step 后的参数
    """
    mus = np.zeros((K,ClusterData.shape[1]))
    for i in range(K):
        iterNumerator = np.zeros(ClusterData.shape[1])
        iterDenominator = 0
        for j in range(ClusterData.shape[0]):
            iterNumerator = iterNumerator + gamma[j][i]*ClusterData[j]
            iterDenominator = iterDenominator+gamma[j][i]
        for k in range(ClusterData.shape[1]):
            # 存在除 0 可能
            mus[i][k] = iterNumerator[k] / iterDenominator
```

```python

    sigmas = np.zeros((K, ClusterData.shape[1], ClusterData.shape[1]))
    for k in range(K):
        iterNumerator = np.zeros((ClusterData.shape[1],ClusterData.shape[1]))
        iterDenominator = 0
        for i in range(ClusterData.shape[0]):
            iterNumerator = iterNumerator + gamma[i][k] * np.dot(np.transpose(ClusterData[i] - mus[k]),(ClusterData[i] - mus[k]))
            iterDenominator = iterDenominator +gamma[i][k]
        sigma = iterNumerator / iterDenominator
        for j in range(ClusterData.shape[1]):
            sigma[j][j] = sigma[j][j]*1.01
        sigmas[k] = sigma

    alpha = np.zeros(K)
    for i in range(K):
        iterNumerator = 0
        for j in range(ClusterData.shape[0]):
            iterNumerator=iterNumerator+gamma[j][i]
        alpha[i] = iterNumerator/ClusterData.shape[0]
    return mus, sigmas, alpha


def initParams(ClusterData, K):
    """
    初始化 GMM 参数
    :param ClusterData:所有数据点
    :param K: 要分类的类别
    :return: GMM 的各个参数
    """
    d = ClusterData.shape[1]
    # 高斯分布模型均值
    mus = np.random.rand(K, d)
    # 初始化协方差矩阵
    sigmas = np.array([np.eye(d)] * K)
    # 假设起始时各个类别概率相同
    alpha = np.ones(K) * (1 / K)
    return mus, sigmas, alpha



def drawGMM(ClusterData,gamma,mus):
    """
```

```
    画出 GMM 簇的图像
    :param ClusterData:所有数据点
    :param gamma: gamma 矩阵
    """
    center = np.zeros(ClusterData.shape[0])
    for i in range(ClusterData.shape[0]):
        center[i] = gamma[i].argmax()
    plt.title("GMM EM algorithm")
    for i in range(gamma.shape[1]):
        listX1 = []
        listX2 = []
        for j in range(ClusterData.shape[0]):
            if int(center[j]) == i:
                listX1.append(ClusterData[j][0])
                listX2.append(ClusterData[j][1])
        plt.scatter(listX1, listX2,c=color[i])
    mu = np.transpose(mus)
    plt.scatter(mu[0],mu[1],c=color[gamma.shape[1]],s=100)
    plt.show()


def TestGMMAccuracy(centersTrain, mus,sigmas,alpha, YTrain, YTest,Y,Test, K):
    # 簇号
    Ytest = np.zeros(Test.shape[0])
    for i in range(Test.shape[0]):
        gamma= getExpectation(Test, mus, sigmas, alpha, K)
        Ytest[i] = np.array(gamma[i]).argmax()
    # Dict 簇到原来 y 的映射
    Dict = ClusterMap(K, YTrain,centersTrain,Y)
    sum = 0
    for i in range(YTest.shape[0]):
        Ytest[i] = Dict[Ytest[i]]
    for i in range(YTest.shape[0]):
        sum = sum +1
    Accuracy = sum/Ytest.shape[0]
    print("GMM EM iris 数据集的正确率为{}".format(Accuracy))


def ClusterMap(K,YTrain, dataCenters,Y):
    Set = list(set(Y))
    List = []
    for i in range(K):
        List.append(i)
```

```python
    listArray = list(itertools.permutations(List,K))
    MapAccuracyDict = np.zeros(len(listArray))
    listDict = []
    for j in range(len(listArray)):
        # 簇号
        Dict = {}
        for k in range(len(listArray[j])):
            Dict[listArray[j][k]] = Set[k]
        listDict.append(Dict)
        MapAccuracyDict[j] = MapAccuracy(Dict,YTrain,dataCenters)
    return listDict[np.array(MapAccuracyDict).argmax()]



def MapAccuracy(Dict,YTrain,dataCenters):
    sum = 0
    for i in range(YTrain.shape[0]):
        if Dict[dataCenters[i]] == YTrain[i]:
            sum = sum+1
    return sum/YTrain.shape[0]
```

**readFromFile.py**

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split


def readFromFile(DocumentName):
    """
    从文件种读取数据
    :param DocumentName: 文件的名字
    :return: 数据
    """
    df = pd.read_csv(DocumentName)
    df.rename(columns={df.columns.array[df.columns.shape[0] - 1]: 'Predict'}, inplace=True)
    df['Predict'] = LabelEncoder().fit_transform(df['Predict'])
    Y = df['Predict']
    X = df.drop('Predict', axis=1)
    X = np.array(X.values)
    Y = np.array(Y.values)
    K = len(set(Y))
```

```python
    return X, Y, K


def makeFileDataForKmeans(X, Y):
    """
    使数据适合写好的模型
    :param X: 数据点
    :return: 修改后的数据
    """
    Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y)
    Xtrain = np.transpose(Xtrain)
    Xtest = np.transpose(Xtest)
    Ytrain = np.transpose(Ytrain)
    Ytest = np.transpose(Ytest)
    return Xtrain, Xtest, Ytrain, Ytest


def makeFileDataForGMM(X, Y):
    """
    使数据适合写好的模型
    :param X: 数据点
    :return: 修改后的数据
    """
    Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y)
    return Xtrain, Xtest, Ytrain, Ytest
```