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聚类方法和混合高斯模型

学号：1190202105

姓名：傅浩东

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

实验测试：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）。

实验应用：可以UCI上找一个简单问题数据，用你实现的GMM进行聚类。

实验环境：**Windows 11; Visual Studio Code; python 3.9.7**

三. 设计实验（主要算法和数据结构）

1. 数据生成与基本操作

(1) 数据的随机生成

`numpy.random.multivariate_normal()` 函数是从多元正态分布中随机抽取样本的函数。多元正态分布、多重正态分布或高斯分布它是一维正态分布向更高维度的推广。这种分布由其均值和协方差矩阵来表示，在本次实验中，我随机生成的是二维正太函数，其中均值`mean` 是一个包含两个元素的矩阵分别表示生成数据的 `x` 方向的均值和 `y` 方向的均值，然后是协方差 `cov` 如下所示，若满足朴素贝叶斯假设则 `b` 与 `c` 为0，即 `x` 与 `y` 独立，为对角矩阵；否则的话 `b` 与 `c` 不为0，那么就不满足朴素贝叶斯条件。

$$cov = \begin{Bmatrix} a, b \\ c, d \end{Bmatrix}$$

(2) 计算准确度

因为是无监督学习，对于给定标签的数据集（例如自己生成的数据）可以计算分类是否符合预期，在这里考虑到有三个以及以上的类别，但是初步选择类别点是随机或者距离最远的，他们标签与实际上类的标签无法准确相符，所以对于给定标签进行排列组合，分别计算准确度，最后选取最高的一个，就是分类的准确度。实际上这个过程就是寻找最符合实际的标签排列。

2. K-Means（K均值算法）

(1) 算法描述

k-means聚类是的方法的矢量量化，最初来自信号处理，其目的是将 N 个观测分为 K 簇，其中每个观测属于最近的均值的簇（聚类中心或群集的质心），作为原型的集群。这导致将数据空间划分为 Voronoi 单元。

给定样本集 $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$，其中每个样本是一个 d 维实向量，“k 均值”(k-means) 算法针对聚类所得簇划分 $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ 最小化平方误差：

$$E = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in C_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|_2^2$$

即：

$$\arg\min_{\mathbf{C}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \left\|\mathbf{x}-\boldsymbol{\mu}_i\right\|^2 = \arg\min_{\mathbf{C}} \sum_{i=1}^{k} |C_i| \operatorname{Var} C_i$$

其中 $\boldsymbol{\mu}_i = \frac{1}{|C_i|} \sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}$ 是簇 C_i 的均值向量。直观来看，上式在一定程度上刻画了簇内样本围绕簇均值向量的紧密程度，E 值越小则簇内样本相似度越高。因为总方差是恒定的，这相当于最大化不同集群中点之间的平方偏差总和，遵循总方差定律。

常用的**初始化方法**有 **Forgy** 和 **Random Partition**。Forgy 方法从数据集中随机选择 *k* 个观测值，并将这些作为初始均值。Random Partition方法首先为每个观测值随机分配一个簇，然后进行更新步骤，从而将初始均值计算为簇随机分配点的质心。Forgy 方法倾向于将初始均值分散开，而 Random Partition 将所有均值放置在靠近数据集中心的位置。Random Partition方法通常更适用于 *k* 谐波均值和模糊 *k* 均值等算法；而对于期望最大化和标准*k*- means 算法，Forgy 初始化方法更可取。在本实验中，我选择Forgy方法，随机选取 k 个均值后，首先为每个观测值（样本）寻找最近的一个分区。

(2) 算法实现

```
输入：样本集 D={x1, x2, ..., xm}
      聚类簇数k.
过程：
    从 D 中随机选择 k 个样本作为初始均值向量 {mu1, mu2, ...,muk}
    repeat
        令 Ci = ∅ (1 <= i <= k)
        for j = 1, 2, ..., m do
            计算计算样本 xj 与各均值向量 mui (1 ≤ i ≤ k) 的距离: dji=||xj - mui||2;
            根据距离最近的均值向量确定 xj 的簇标记: lambda_j = arg min i∈{1,  2, ...,
k}dji;
            将样本 xj 划入相应的簇: C_lambda_j ∪ {xj};
        end for
        for i = 1, 2, ..., k do
            计算新均值向量: mui' = 1/|Ci| Σ{x∈Ci}x;
            if mui' ≠ mui then
                将当前均值向量更新为 mui'
            else
                保持当前均值向量不变
            end if
        end for
    until 当前均值向量均未更新
输出：簇划分 C = {C1, C2, ..., Ck}
```

3. GMM（高斯混合聚类）

(1) 算法描述

与k均值用原型向量来刻画聚类结构不同，**高斯混合(Mixture-of-Gaussian)聚类**采用概率模型来表达聚类原型，首先对 n 维样本空间 $\mathcal{X}$ 中的随机向量 $\boldsymbol{x}$, 若 $\boldsymbol{x}$ 服从高斯分布, 其概率密度函数为：

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

其中 $\boldsymbol{\mu}$ 是 n 维均值向量, $\boldsymbol{\Sigma}$ 是 $n \times n$ 的协方差矩阵. 由式(9.28)可看出, 高斯分 布完全由均值向量 $\boldsymbol{\mu}$ 和协方差矩阵 $\boldsymbol{\Sigma i}$ 这两个参数确定，为了明确参数依赖关系，将概率密度函数记为：$p(x|\mu, \Sigma)$.

我们可定义高斯混合分布

$$p_{\mathcal{M}}(\boldsymbol{x}) = \sum_{i=1}^{k} \alpha_i \cdot p\left(\boldsymbol{x} \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)$$

该分布共由 k 个混合成分组成, 每个混合成分对应一个高斯分布. 其中 μ_i 与 $\mathbf{\Sigma}_i$ 是第 i 个高斯混合成分的参数, 而 $\alpha_i > 0$ 为相应的**混合系数** (mixture coefficient), $\sum_{i=1}^{k} \alpha_i = 1$.

根据贝叶斯定理，后验分布有:

$$\begin{aligned} p_{\mathcal{M}}\left(z_j = i \mid \boldsymbol{x}_j\right) &= \frac{P\left(z_j = i\right) \cdot p_{\mathcal{M}}\left(\boldsymbol{x}_j \mid z_j = i\right)}{p_{\mathcal{M}}\left(\boldsymbol{x}_j\right)} \\ &= \frac{\alpha_i \cdot p\left(\boldsymbol{x}_j \mid \boldsymbol{\mu}_i, \mathbf{\Sigma}_i\right)}{\sum_{l=1}^{k} \alpha_l \cdot p\left(\boldsymbol{x}_j \mid \boldsymbol{\mu}_l, \mathbf{\Sigma}_l\right)} \end{aligned}$$

$p_{\mathcal{M}}\left(z_j = i \mid \boldsymbol{x}_j\right)$ 给出了样本 x_j 由第 i 个高斯混合成分生成的后验概率，记为 $\gamma_{ji} (i = 1.2, \ldots, k)$. 那么每个样本的簇标记记为：$\lambda_j = \underset{i \in \{1,2,\ldots,k\}}{\arg\max}\, \gamma_{ji}$.

那么, 对于式 (9.29), 模型参数 $\left\{\left(\alpha_i, \boldsymbol{\mu}_i, \mathbf{\Sigma}_i\right) \mid 1 \leqslant i \leqslant k\right\}$ 如何求解呢? 显然, 给定样本集 D, 可采用极大似然估计, 即最大化(对数)似然

$$\begin{aligned} LL(D) &= \ln\left(\prod_{j=1}^{m} p_{\mathcal{M}}\left(\boldsymbol{x}_j\right)\right) \\ &= \sum_{j=1}^{m} \ln\left(\sum_{i=1}^{k} \alpha_i \cdot p\left(\boldsymbol{x}_j \mid \boldsymbol{\mu}_i, \mathbf{\Sigma}_i\right)\right) \end{aligned}$$

采用 ***EM* 算法** 进行迭代优化求解，若参数 $\left\{\left(\alpha_i, \boldsymbol{\mu}_i, \mathbf{\Sigma}_i\right) \mid 1 \leqslant i \leqslant k\right\}$ 能使上式最大化，则由 $\frac{\partial LL(D)}{\partial \mu_i} = 0$ 有:

$$\sum_{j=1}^{m} \frac{\alpha_i \cdot p\left(\boldsymbol{x}_j \mid \boldsymbol{\mu}_i, \mathbf{\Sigma}_i\right)}{\sum_{l=1}^{k} \alpha_l \cdot p\left(\boldsymbol{x}_j \mid \boldsymbol{\mu}_l, \mathbf{\Sigma}_l\right)}\left(\boldsymbol{x}_j - \boldsymbol{\mu}_i\right) = 0$$

那么可以得到:

$$\boldsymbol{\mu}_i = \frac{\sum_{j=1}^{m} \gamma_{ji} \boldsymbol{x}_j}{\sum_{j=1}^{m} \gamma_{ji}}$$

同样地:

$$\mathbf{\Sigma}_i = \frac{\sum_{j=1}^{m} \gamma_{ji}\left(\boldsymbol{x}_j - \boldsymbol{\mu}_i\right)\left(\boldsymbol{x}_j - \boldsymbol{\mu}_i\right)^{\mathrm{T}}}{\sum_{j=1}^{m} \gamma_{ji}}$$

对于混合系数 α_i, 除了要最大化 $LL(D)$，还需满足 $\alpha_i \geqslant 0$, $\sum_{i=1}^{k} \alpha_i = 1$. 考虑 $LL(D)$ 的拉格朗日形式

$$LL(D) + \lambda\left(\sum_{i=1}^{k} \alpha_i - 1\right)$$

其中 λ 为拉格朗日乘子. 由 上式对 α_i 的导数为 0 , 有:

$$\sum_{j=1}^{m} \frac{p\left(\boldsymbol{x}_j \mid \boldsymbol{\mu}_i, \mathbf{\Sigma}_i\right)}{\sum_{l=1}^{k} \alpha_l \cdot p\left(\boldsymbol{x}_j \mid \boldsymbol{\mu}_l, \mathbf{\Sigma}_l\right)} + \lambda = 0$$

两边同乘以 α_i, 对所有样本求和可知 $\lambda = -m$, 有

$$\alpha_i = \frac{1}{m} \sum_{j=1}^{m} \gamma_{ji}$$

即每个高斯成分的混合系数由样本属于该成分的平均后验概率确定.

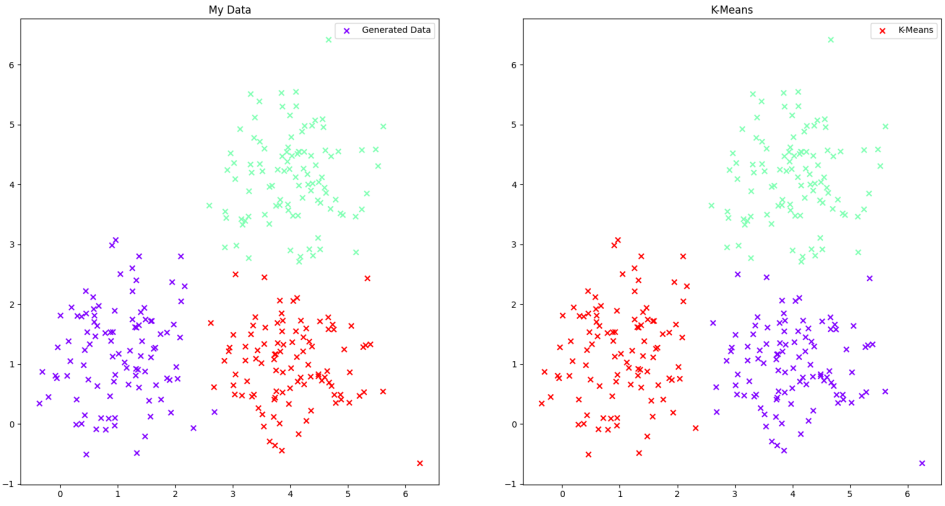
(2) 算法实现

```
输入: 样本集 D = {x1,x2,...,xm}
      高斯混合成分个数 k.
过程:
    初始化高斯混合分布的模型参数 {(alpha_i, mu_i, Sigma_i)| 1 ≤ i ≤ k}
    repeat
        for j = 1,2, ..., m do
            根据式 (9.30)计算 $\boldsymbol{x}_{j}$ 由各混合成分生成的后验概率, 即
            gamma_ji = PM(zi = i |xi)(1 ≤ i ≤ k)
        end for
        for i = 1,2, ..., k do
            计算新的均值向量 mu'_i、协方差矩阵 Sigma'_i、混合系数 alpha'_i
        end for
        将模型参数{(alpha_i, mu_i, Sigma_i)| 1 ≤ i ≤ k}更新
    until 满足停止条件，这里是对数似然几乎不变就停止
    Ci = ∅ (1 <= i <= k)
    for j = 1,2, ..., m do
        确定簇标记，将 xj 划入相应的簇
        C_lambda_j = C_lambda_j ∪ {xj}
    end for
输出：簇划分 C = {C1, C2, ..., Ck}
```

四. 实验结果与分析

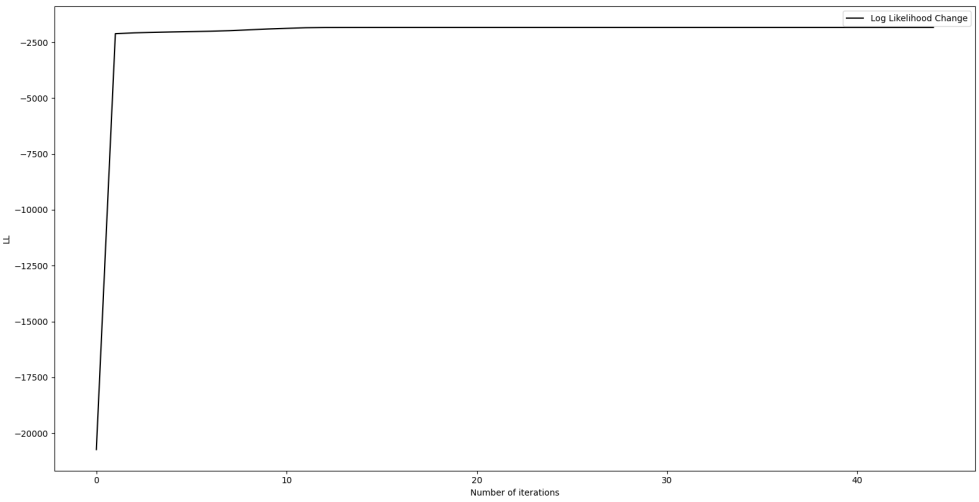
1. K-Means

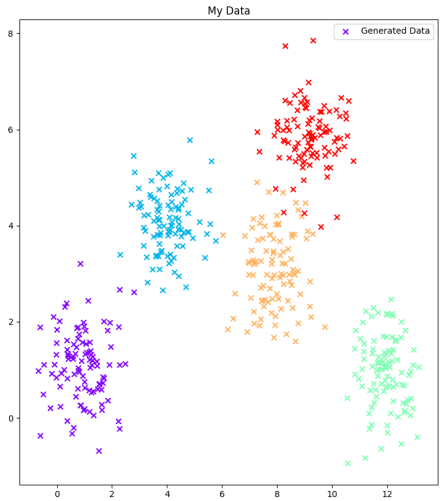
一般情况下都能正确分类，当数据交叉变小，正确聚类概率越大。对于本实验三组高斯数据，如预期聚类，准确率一般有 **97%-98%**.

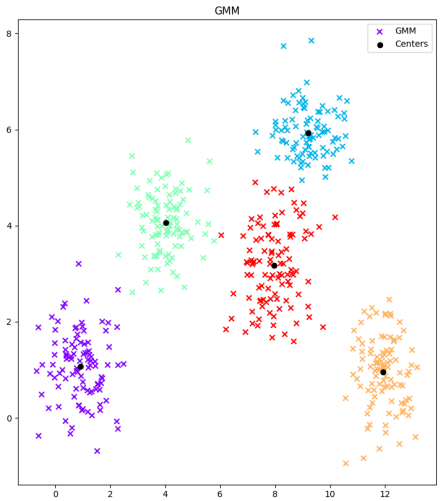


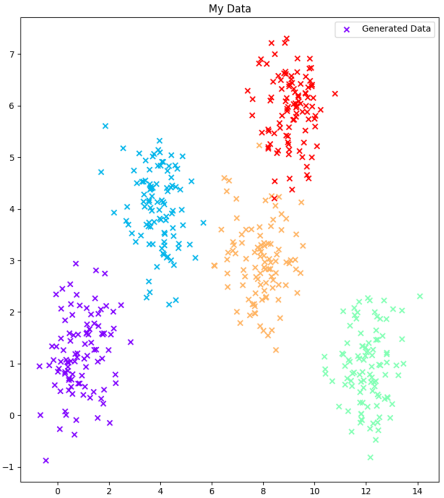
2. GMM

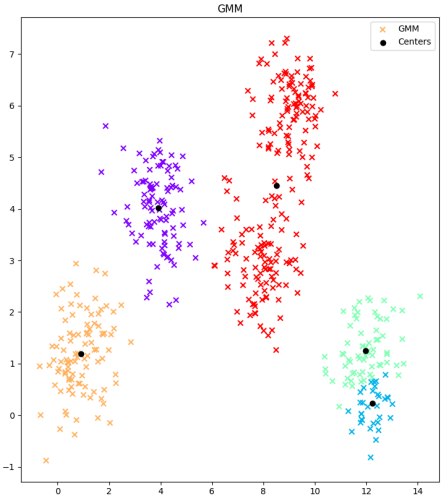
GMM（高斯混合聚类）在最开始几次迭代过程中，对数似然增长得很快，而接下来就缓慢增长。聚类有时合理，有时又不太合理，都会出现。对于自己生成的高斯分布数据，GMM 相较于 K-Means 算法发生偏差的概率更加大，经常会出现临近的两个类被归为一类，而少的那一栏极少数的只有几个数据值被包含。实验中生成五组高斯数据，如果如预期聚类，准确率基本上能够上升至 **99%** 左右.


Log Likelihood Change
LL
Number of iterations


My Data
Generated Data

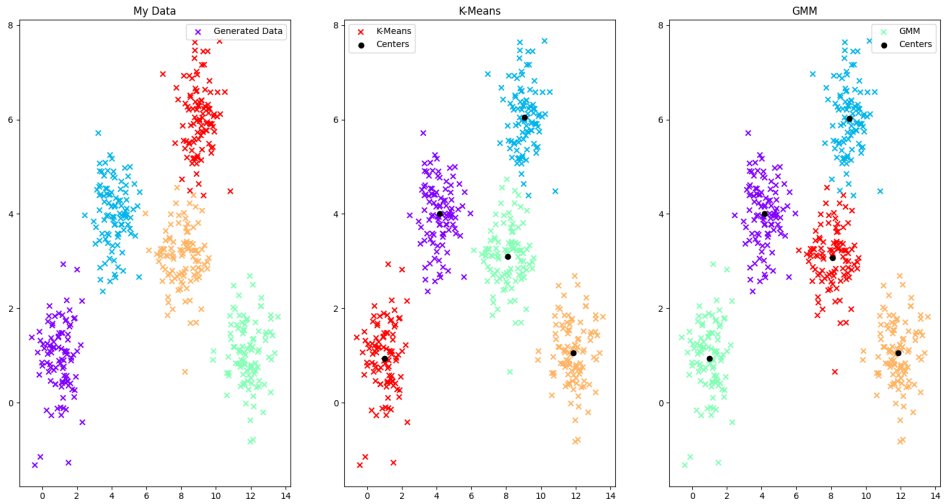

GMM
GMM
Centers


My Data
Generated Data

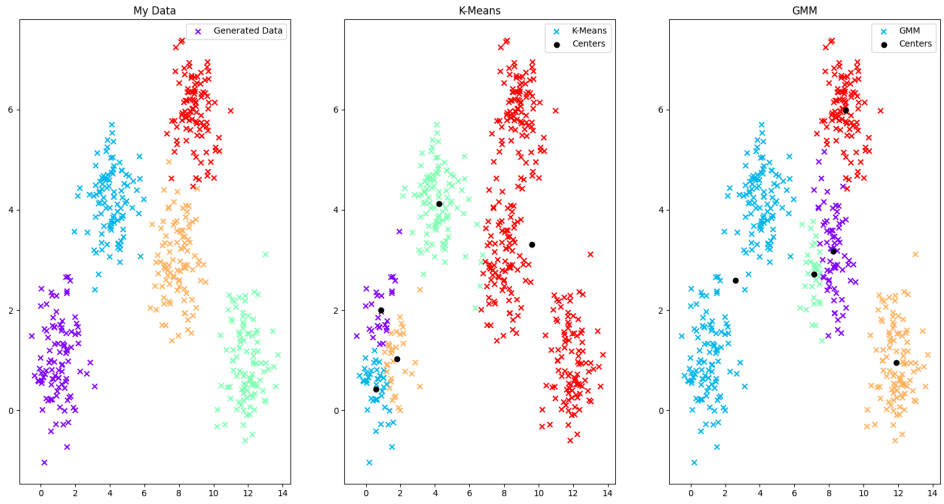

GMM
GMM
Centers

3. 方法对比

无论是 **K-Means** 还是 **GMM** 两种算法都能实现聚类，如下表所示，K-Means 算法迭代次数相较于 GMM 要少一些，当正确聚类的结果下准确率都还算比较高。但与此同时，他们也都有出错的时候，例如接下来就有两类均聚类错误的时候，这也许是由于初始化选取中心点时有些偏差，除了下面的聚类结果，其实还存在GMM迭代次数超过上限后退出，在这里就不去一一列举了。



Methods	Accuracy	Iterations
K-Means	0.994	9
GMM	0.998	61



Methods	Accuracy	Iterations
K-Means	0.48	5
GMM	0.728	177

4. UCI实验数据

(1) Iris Data Set

该数据集包含四个鸢尾花属性：花萼长、宽以及花瓣长、宽，用标签 `0-2` 来替换类别标签。

鸢尾花最好的情况下，即两类方法聚类结果都比较优秀时，准确率能够有 **K-Means Accuracy: 0.893，GMM Accuracy 0.967。**

(2) Cervical Cancer Behavior Risk Data Set

使用实验二中关于宫颈癌的数据（二聚类问题）。

该数据集包含 19 个关于宫颈癌行为风险的属性，类标签为**ca_cervix**，值为 1 和 0，分别表示有和没有宫颈癌的受访者。该数据集包括18个属性，来自八个变量，分别是：**behavior** (sexualRisk，eating, personalHygine); **intention**(aggregation，commitment); **attitude** (consistency, spontaneity); **norm** (significantPerson, fulfillment); **perception** (vulnerability, severity)；**motivation** (strength, willingness); **socialSupport** (emotionality, appreciation, instrumental); **empowerment** (knowledge, abilities, desires)。

如果使用GMM算法进行聚类，那么会出现如下错误，`log` 时会出现负无穷，再用 `multivariate_normal.pdf` 去求概率分布就会有问题。那么只用 K-Means 进行聚类准确率大约最高有： **80%**.

```
d:\Learn\Machine_Learning\Lab\Lab3\GMM.py:80: RuntimeWarning: invalid value encountered in double_scalars
  if self.__LL - last_LL <= self.delta:
Traceback (most recent call last):
  File "d:\Learn\Machine_Learning\Lab\Lab3\sobar.py", line 23, in <module>
    gmm_mu, gmm_label, gmm_iter, gmm_LLs = gmm.GMM()
  File "d:\Learn\Machine_Learning\Lab\Lab3\GMM.py", line 77, in GMM
    self.__expectation()
  File "d:\Learn\Machine_Learning\Lab\Lab3\GMM.py", line 54, in __expectation
    Mixture_distribution = self.__ProbabilityDensity() * self.__alpha
  File "d:\Learn\Machine_Learning\Lab\Lab3\GMM.py", line 46, in __ProbabilityDensity
    ProbabilityDensity[:, i] = multivariate_normal.pdf(
  File "C:\Users\10912\AppData\Roaming\Python\Python39\site-packages\scipy\stats\_multivariate.py", line 516, in pdf
    psd = _PSD(cov, allow_singular=allow_singular)
  File "C:\Users\10912\AppData\Roaming\Python\Python39\site-packages\scipy\stats\_multivariate.py", line 158, in __init__
    s, u = scipy.linalg.eigh(M, lower=lower, check_finite=check_finite)
  File "C:\Users\10912\AppData\Roaming\Python\Python39\site-packages\scipy\linalg\decomp.py", line 445, in eigh
    a1 = _asarray_validated(a, check_finite=check_finite)
  File "C:\Users\10912\AppData\Roaming\Python\Python39\site-packages\scipy\_lib\_util.py", line 293, in _asarray_validated
    a = toarray(a)
  File "C:\Python39\lib\site-packages\numpy\lib\function_base.py", line 488, in asarray_chkfinite
    raise ValueError(
ValueError: array must not contain infs or NaNs
```

五. 结论

1. **K-Means**：当簇近似高斯分布的时候，效果非常不错；对初始的簇中心敏感，不同选取方式会得到不同结果；不适合太离散的分类、样本类别不平衡的分类、非凸形状的分类。
2. **GMM**：不是得到一个确定的分类标记，而是得到每个类的概率。当每个混合模型没有足够多的点时，估算协方差变得困难起来，同时算法会发散并且找具有无穷大似然函数值的解，除非人为地对协方差进行正则化。GMM每一步迭代的计算量比较大，大于k-means。GMM的求解办法基于EM算法，因此有可能陷入局部极值，这和初始值的选取十分相关。

六. 参考文献

[1] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.multivariate_normal.html

[2] https://en.wikipedia.org/wiki/Mixture_model

[3] https://en.wikipedia.org/wiki/K-means_clustering

[4] https://archive.ics.uci.edu/ml/datasets/Cervical+Cancer+Behavior+Risk

[5] https://archive.ics.uci.edu/ml/datasets/iris

七. 附录：源代码（带注释）

数据处理 与基本操作 operations.py

```
import numpy as np
from itertools import permutations


def Data(mean, N=100, naive=True):
    """随机生成一组高斯数据，特征是二维的，可以选择条件是否满足朴素贝叶斯
    Args:
        mean (array): 生成数据的均值中心，是一个一行二列的行向量.
        N (int, optional): 正例数量. Defaults to 100.
        naive (bool, optional): 是否满足朴素贝叶斯条件，True表示满足（即条件独立）.
Defaults to True.
    Returns:
        array: 返回生成的特征数组x和标签y（1或0）
    """
    cov_naive = [[0.5, 0], [0, 0.5]]
    cov_NOT_naive = [[2, 1], [1, 2]]
    # 满足朴素贝叶斯假设
    if naive:
        cov = cov_naive
    # 不满足朴素贝叶斯假设
    else:
        cov = cov_NOT_naive
    x = np.random.multivariate_normal(mean, cov, size=N)
    return x

def Accuracy(real_label, class_label, k):
    """计算分类标签的准确性（与源标签对比）
    Args:
        real_label (array): 样本数量长的列向量，由生成数据决定
        class_label (array): 样本数量长的列向量，由分类结果决定
        k (int): 样本类别数，标签是从 0 到 k-1
    Returns:
        [type]: [description]
    """
    # Full Permutation of labels
    # The highest accuracy is taken as the result
    classes = list(permutations(range(k), k))
    counts = np.zeros(len(classes))
    for i in range(len(classes)):
        for j in range(real_label.shape[0]):
            if int(real_label[j]) == classes[i][int(class_label[j])]:
                counts[i] += 1
    return np.max(counts) / real_label.shape[0]
```

K均值算法 k_means.py

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

```
from operations import Data, Accuracy


class K_Means(object):
    def __init__(self, data, k, delta=1e-6):
        """ 初始化数据集
        Args:
            data (array): 数据数组，每行表示一个样本地特征，行数表示样本数
            k (int): 要分成的类数
            delta (double, optional): tolerance，当类别中心的变化的二范数
                                      小于这个值就认为分类结束. Defaults to 1e-6.
        """
        self.data = data
        self.k = k
        self.delta = delta
        # Randomly select k vertices as the initial cluster center point
        self.mu = np.array(self.data[random.sample(range(data.shape[0]), k)])
        self.label = np.zeros(data.shape[0])

    def k_means(self):
        iter = 0
        while True:
            iter += 1
            distance = np.zeros(self.k)
            # Determine the cluster label of each vector
            # according to the nearest cluster center
            for i in range(self.data.shape[0]):
                for j in range(self.k):
                    distance[j] = np.linalg.norm(
                        self.data[i] - self.mu[j])
                self.label[i] = np.argmin(distance)
            # Calculate new k centers based on the new labels of all points
            new_mu = np.zeros((self.k, self.data.shape[1]))
            count = np.zeros(self.k)
            for i in range(self.data.shape[0]):
                new_mu[int(self.label[i])] += self.data[i]
                count[int(self.label[i])] += 1
            new_mu = new_mu / count[:, None]
            # Use the two-norm of the difference to express precision
            if np.linalg.norm(new_mu - self.mu) < self.delta:
                break
            else:
                self.mu = new_mu
        return self.mu, self.label, iter


if __name__ == "__main__":
    data0 = Data([1, 1])
    data1 = Data([4, 4])
    data2 = Data([4, 1])
    classes = 3
    data = np.vstack((data0, data1, data2))
    real_label = np.concatenate(
        (np.zeros(len(data0)), np.ones(len(data1)), 2 * np.ones(len(data2))))
    plt.subplot(121)
    plt.title("My Data")
    plt.scatter(data[:, 0].tolist(), data[:, 1].tolist(), marker="x",
c=np.array(
```

```
        real_label), cmap='rainbow', label="Generated Data")
    plt.legend()

    kmeans = K_Means(data, classes)
    mu, label, iter = kmeans.k_means()
    # show result
    plt.subplot(122)
    plt.title("K-Means")
    plt.scatter(data[:, 0].tolist(), data[:, 1].tolist(),
                marker="x", c=np.array(label), cmap='rainbow', label="K-Means")
    plt.legend()

    plt.show()
    accuracy = Accuracy(real_label, label, classes)
    print("Accuracy: " + str(accuracy))
    print("Iterations: " + str(iter))
```

高斯混合聚类EM算法 GMM.py

```
import numpy as np
import random
from scipy.stats import multivariate_normal
import collections
import matplotlib.pyplot as plt
from operations import Data, Accuracy


class GaussianMixtureModel(object):
    def __init__(self, data, k, delta=1e-12, max_iteration=1000):
        """初始化GMM
        Args:
            data (array): 数据数组，每行表示一个样本地特征，行数表示样本数
            k (int): 要分成的类数
            delta (double, optional): tolerance，当类别中心的变化的二范数小于这个值就认为分类结束. Defaults to 1e-12.
            max_iteration (int, optional): 最大迭代次数，超过这个值就不再进行迭代. Defaults to 1000.
        """
        self.data = data
        self.k = k
        self.delta = delta
        self.max_iteration = max_iteration
        self.__alpha = np.ones(self.k) * (1.0 / self.k)
        self.__mu = np.array(self.data[random.sample(range(data.shape[0]), k)])
        self.__Sigma = self.__init_Sigma()
        self.label = None
        self.__gamma = None
        self.__LL = -np.inf

    def __init_Sigma(self):
        """初始化Sigma值，对角矩阵，且对角上每个值都设为0.1
        Returns:
            array: Sigma矩阵
        """
```

```
        Sigma = collections.defaultdict(list)
        for i in range(self.k):
            Sigma[i] = np.eye(self.data.shape[1], dtype=float) * 0.1
        return Sigma

    def __ProbabilityDensity(self):
        """根据分布计算每个样本属于某个类别的概率
        Returns:
            array: 每个样本属于每个类别的概率
        """
        ProbabilityDensity = np.zeros((self.data.shape[0], self.k))
        for i in range(self.k):
            ProbabilityDensity[:, i] = multivariate_normal.pdf(
                self.data, self.__mu[i], self.__Sigma[i])
        return ProbabilityDensity

    def __expectation(self):
        """E步，计算后验概率
        """
        # Gaussian mixture distribution
        Mixture_distribution = self.__ProbabilityDensity() * self.__alpha
        sum_ProbabilityDensity = np.sum(Mixture_distribution,
axis=1).reshape(-1, 1)
        self.__LL = np.sum(np.log(sum_ProbabilityDensity))
        print(self.__LL)
        self.__gamma = Mixture_distribution / sum_ProbabilityDensity
        self.label = self.__gamma.argmax(axis=1)

    def __maximization(self):
        """M步，更新mu、Sigma和alpha
        """
        for i in range(self.k):
            gamma = self.__gamma[:, i].reshape(-1, 1)
            mu_i = np.sum(gamma * self.data, axis=0) / np.sum(gamma)
            covariance = (self.data - mu_i).T.dot((self.data -
                                                   mu_i) * gamma) /
np.sum(gamma)
            self.__mu[i], self.__Sigma[i] = mu_i, covariance
        self.__alpha = self.__gamma.sum(axis=0) / self.data.shape[0]

    def GMM(self):
        last_LL = self.__LL
        iter = 0
        LLs = []
        while iter < self.max_iteration:
            self.__expectation()
            self.__maximization()
            iter += 1
            if self.__LL - last_LL <= self.delta:
                break
            last_LL = self.__LL
            LLs.append(last_LL)
        self.__expectation()
        return self.__mu, self.label, iter, LLs


if __name__ == '__main__':
    data0 = Data([1, 1])
```

```
    data1 = Data([4, 4])
    data2 = Data([12, 1])
    data3 = Data([8, 3])
    data4 = Data([9, 6])
    classes = 5
    data = np.vstack((data0, data1, data2, data3, data4))
    real_label = np.concatenate((np.zeros(len(data0)), np.ones(len(
        data1)), 2 * np.ones(len(data2)), 3 * np.ones(len(data3)), 4 *
np.ones(len(data4))))

    plt.subplot(121)
    plt.title("My Data")
    plt.scatter(data[:, 0].tolist(), data[:, 1].tolist(), marker="x",
c=np.array(
        real_label), cmap='rainbow', label="Generated Data")
    plt.legend()

    GMM = GaussianMixtureModel(data, classes)
    mu, label, iter, LLs = GMM.GMM()
    # show result
    plt.subplot(122)
    plt.title("GMM")
    plt.scatter(data[:, 0].tolist(), data[:, 1].tolist(),
                marker="x", c=np.array(label), cmap='rainbow', label="GMM")
    plt.scatter(mu[:, 0], mu[:, 1], color="k", label="Centers")
    plt.legend()
    plt.show()
    accuracy = Accuracy(real_label, label, classes)
    print("Accuracy: " + str(accuracy))
    print("Iterations: " + str(iter))

    plt.plot(LLs, color="k", label="Log Likelihood Change")
    plt.xlabel("Number of iterations")
    plt.ylabel("LL")
    plt.legend(loc='best')
    plt.show()
```

两类方法比较 comparison.py

```
import numpy as np
import matplotlib.pyplot as plt
from operations import Data, Accuracy
from operations import *
from GMM import *
from k_means import *
import prettytable as pt

if __name__ == '__main__':
    data0 = Data([1, 1])
    data1 = Data([4, 4])
    data2 = Data([12, 1])
    data3 = Data([8, 3])
    data4 = Data([9, 6])
    classes = 5
```

```
    data = np.vstack((data0, data1, data2, data3, data4))
    real_label = np.concatenate((np.zeros(len(data0)), np.ones(len(
        data1)), 2 * np.ones(len(data2)), 3 * np.ones(len(data3)), 4 *
np.ones(len(data4))))

    k_means = K_Means(data, classes)
    kmeans_mu, kmeans_label, kmeans_iter = k_means.k_means()
    gmm = GaussianMixtureModel(data, classes)
    gmm_mu, gmm_label, gmm_iter, gmm_LLs = gmm.GMM()

    plt.subplot(131)
    plt.title("My Data")
    plt.scatter(data[:, 0].tolist(), data[:, 1].tolist(), marker="x",
c=np.array(
        real_label), cmap='rainbow', label="Generated Data")
    plt.legend()

    plt.subplot(132)
    plt.title("K-Means")
    plt.scatter(data[:, 0].tolist(), data[:, 1].tolist(),
                marker="x", c=np.array(kmeans_label), cmap='rainbow', label="K-
Means")
    plt.scatter(kmeans_mu[:, 0], kmeans_mu[:, 1], color="k", label="Centers")
    plt.legend()

    plt.subplot(133)
    plt.title("GMM")
    plt.scatter(data[:, 0].tolist(), data[:, 1].tolist(),
                marker="x", c=np.array(gmm_label), cmap='rainbow', label="GMM")
    plt.scatter(gmm_mu[:, 0], gmm_mu[:, 1], color="k", label="Centers")
    plt.legend()
    plt.show()

    plt.plot(gmm_LLs, color="k", label="Log Likelihood Change")
    plt.xlabel("Number of iterations")
    plt.ylabel("LL")
    plt.legend(loc='best')
    plt.show()

    kmeans_accuracy = Accuracy(real_label, kmeans_label, classes)
    gmm_accuracy = Accuracy(real_label, gmm_label, classes)
    tb = pt.PrettyTable()
    tb.field_names = ["Methods", "Accuracy", "Iterations"]
    tb.add_row(["K-Means", kmeans_accuracy, kmeans_iter])
    tb.add_row(["GMM", gmm_accuracy, gmm_iter])
    print(tb)
```

鸢尾花分类 iris.py

```
import numpy as np
import pandas as pd
from operations import *
from GMM import *
from k_means import *
```

```
# 鸢尾花

def GetData():
    data_set = pd.read_csv("./data/iris.csv")
    x = data_set.drop('class', axis=1)
    y = data_set['class']
    y = y.replace('Iris-setosa', 0).replace('Iris-versicolor', 1).replace('Iris-
virginica', 2)
    return x, y

if __name__ == '__main__':
    x, y = GetData()
    data = np.array(x)
    real_label = np.array(y)
    classes = np.unique(real_label).shape[0]

    k_means = K_Means(data, classes)
    kmeans_mu, kmeans_label, kmeans_iter = k_means.k_means()

    gmm = GaussianMixtureModel(data, classes)
    gmm_mu, gmm_label, gmm_iter, gmm_LLs = gmm.GMM()

    plt.plot(gmm_LLs, color="k", label="Log Likelihood Change")
    plt.xlabel("Number of iterations")
    plt.ylabel("LL")
    plt.legend(loc='best')
    plt.show()

    kmeans_accuracy = Accuracy(real_label, kmeans_label, classes)
    gmm_accuracy = Accuracy(real_label, gmm_label, classes)
    print("K-Means Accuracy: ", kmeans_accuracy)
    print("GMM Accuracy", gmm_accuracy)
```

宫颈癌实例 sobar.py

```
import numpy as np
import pandas as pd
from operations import *
from k_means import *
from GMM import *

def GetData():
    data_set = pd.read_csv("./data/sobar-72.csv")
    x = data_set.drop('ca_cervix', axis=1)
    y = data_set['ca_cervix']
    return x, y

if __name__ == '__main__':
    x, y = GetData()
    data = np.array(x)
    real_label = np.array(y)
    classes = np.unique(real_label).shape[0]

    k_means = K_Means(data, classes)
```

```
kmeans_mu, kmeans_label, kmeans_iter = k_means.k_means()

# gmm = GaussianMixtureModel(data, classes)
# gmm_mu, gmm_label, gmm_iter, gmm_LLs = gmm.GMM()

# plt.plot(gmm_LLs, color="k", label="Log Likelihood Change")
# plt.xlabel("Number of iterations")
# plt.ylabel("LL")
# plt.legend(loc='best')
# plt.show()

kmeans_accuracy = Accuracy(real_label, kmeans_label, classes)
# gmm_accuracy = Accuracy(real_label, gmm_label, classes)
print("K-Means Accuracy: ", kmeans_accuracy)
# print("GMM Accuracy", gmm_accuracy)
```