曹瑞升-5140219073 秦格华-5140219335

18/May/2017

Project4 K-means 实验报告

一. K-means 实现细节

1. 调用 Python numpy 包中 *loadtxt* 函数从 data.csv 中读取数据到 *my_matrix* 中

2. 编写 *initialCentroids* 函数, 选取 k 个点作为 k 个簇中心, 初始化条件:
   先随机选取一个点，然后依次选取其他 k-1 个点，到已选簇中心点的最小距离尽可能大

3. 迭代直到收敛：依次将每一个点根据距离最小分配到 k 个簇中 *seperatePoints*，重新计算簇中心点 (簇中数据点的质心)*computeCentroids*

4. 终止条件：前后两次各簇中心点的距离 *computeDifference* 小于 10^{-9}

二. 聚类结果

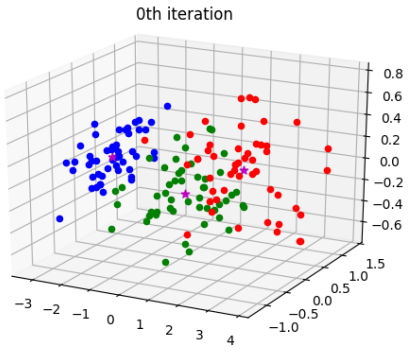

图 1: 原始分类

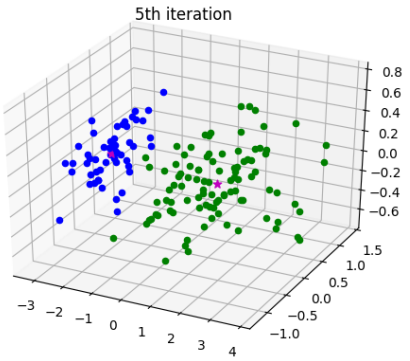

图 2: k=2

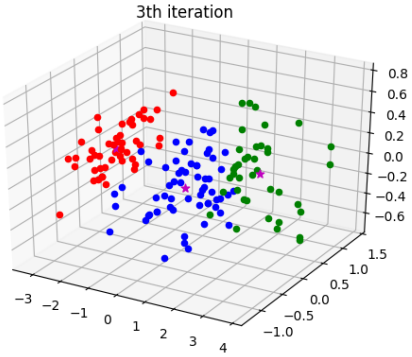

图 3: k=3

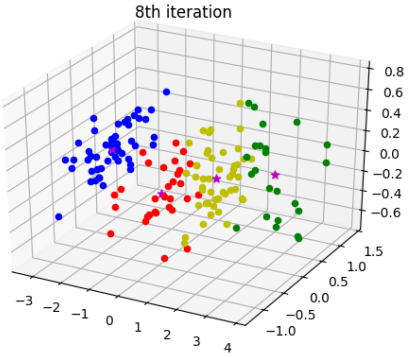

图 4: k=4

二. 不同 clusters 数的影响

簇平均直径：各个簇中任意两点之间最大距离的平均值 *computeAvgDiameter*

聚类数	簇平均直径	迭代次数	运行时间
2	15.3240	5	0.013171
3	6.2597	6	0.017537
4	4.1033	11	0.037445

- 聚类数从 2 变为 3 时簇平均直径变化很大，而从 3 变为 4 时簇直径变化较小，说明 k=3 是一个较好的聚类数
- 随着聚类数的增加，迭代次数相应的也会有所提高，本例中由于数据量较小且聚类数也选取较小，所以 k=2 或 3 时迭代次数比较接近，甚至会出现反常现象，但一般情况下 k=4 的迭代次数都会大于 k=3 时的迭代次数
- 随着聚类数的增加，运行平均时间也会相应提高

三. 不同 seeds 对结果的影响

- 将源程序中的 initialCentroids 函数换为 initialRandomCentroids 函数, 初始化条件改为随机选取 k 个点作为簇中心点
- 用 k 个初始点的离散程度作为衡量初始点质量的指标，computeDispersion 计算 k 个初始点在所有维度的方差之和

从图 5和表格 1可以分析，总体上来说，一开始随着初始点之间离散程度的增加 (点之间越分散)，迭代次数和运行时间随之减小，但当离散程度增加到一定程度之后 (选取的是数据集最边缘的样本点)，迭代次数和运行时间反而有增大的趋势

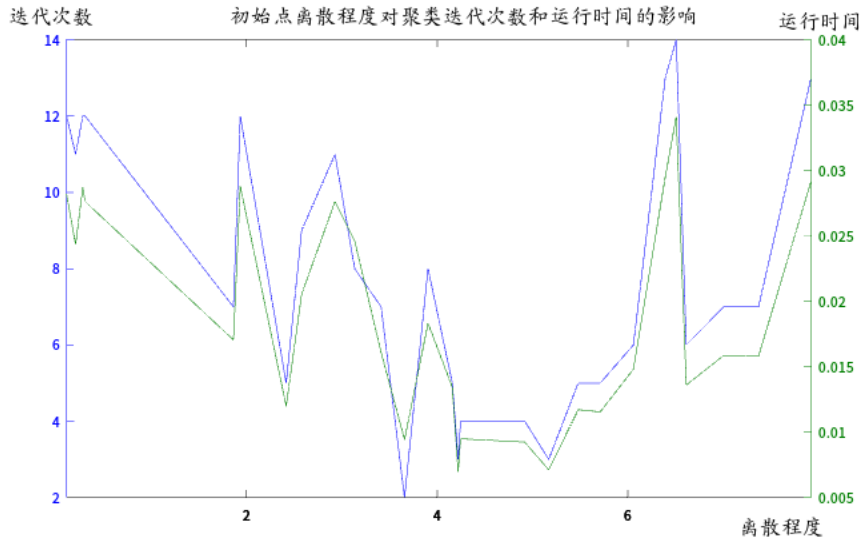

图 5: 初始点对聚类结果的影响

离散程度	迭代次数	运行时间
0.1126	12	0.028186
0.2072	11	0.024348
0.2826	12	0.028698
0.3072	12	0.027628
1.8627	7	0.017059
1.9346	12	0.028802
2.4160	5	0.011992
2.5782	9	0.020550
2.9281	11	0.027635
3.1347	8	0.024623
3.4118	7	0.016091
3.6573	2	0.009432
3.9039	8	0.018327
4.1587	5	0.013439
4.2201	3	0.006968
4.2450	4	0.009495
4.9183	4	0.009241
5.1701	3	0.007108
5.4785	5	0.011718
5.7095	5	0.011549
6.0581	6	0.014810
6.3920	13	0.029488
6.5073	14	0.034111
6.6125	6	0.013605
7.0042	7	0.015858
7.3702	7	0.015798
7.9213	13	0.029168

表 1: seeds 与聚类结果

四. 不同 clusters 数及 sample 对运行时间的影响

- 对 sample 大小的改变：m=50,100 时选取 150 个样本中前 50,100 个，m=300 时将 150 个样本复制一份 (由于存在重复数据这种情况下实际运行时间和迭代次数小于理论值)

- 从表 2可以看出，随着聚类数的增加，运行时间增加 (数据较少且具有随机性，无法定性分析)
- 从表 3可以看出，随着样本数的增加，运行时间增加 (同上)
- 理论上，时间复杂度与聚类数 k 和样本数 m 的关系：I 最大迭代次数，d 数据维度

$$O(I \cdot k \cdot m \cdot d)$$

表 2: 不同 clusters 数

聚类数	运行时间
2	0.013171
3	0.017537
4	0.037445

表 3: 不同 sample 数

样本数	运行时间
50	0.003770
100	0.007325
150	0.017287
300	0.024169

Extra. 基于高斯混合模型的 EM 聚类算法

在上面实现了基于 k-means 的聚类算法后，我们又查阅了相关资料，发现了与 k-means 具有一定相似性的基于高斯混合模型（Guassian Mixture Model，下面简称为 GMM）的 EM（Expectation Maximization）聚类算法[1]。此模型将不同的 cluster 定义为有着不同参数的混合分布，所以聚类的过程就是首先针对不同混合分布的隐藏参数进行学习，然后再对不同的样本点在各个分布下不同的概率大小来最终决定样本点应该归入哪一个 cluster 中。所以在解决隐藏参数计算问题时，我们常见的方法就是利用 EM 算法进行迭代，使参数收敛。实际上，k-means 可以看做 GMM 混合聚类在混合成分方差相等，且每个样本仅指派给一个混合成分时的特例。下面我们给出 GMM-EM 算法的基本过程。

首先初始化模型的参数：$\{(\alpha_i, \mu_i, \Sigma_i) | 1 \le i \le k\}$，其中三项分别是模型的混合参数、向量均值和样本在向量均值下的协方差矩阵。下面我们通过公式可得 EM 算法中的 E-step：

对于数据集中的每一个样本，计算其由每一个混合成分生成的后验概率：

$$\gamma_{ji} = p_{M(z_j=i|x_j)} = \frac{\alpha_i \cdot p(x_j|\mu_i, \Sigma_i)}{\sum_{l=1}^{k} \alpha_i \cdot p(x_j|\mu_l, \Sigma_l)}, \quad p(x) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$$

下面是算法的 M-step，用于更新不同混合成分的特征参数：

(1) 计算均值向量：$\mu_i' = \frac{\sum_{j=1}^{m} \gamma_{ji} x_j}{\sum_{j=1}^{m} \gamma_{ji}}$；

[1]本部分内容可参考：周志华. 机器学习 [M]. 清华大学出版社, 2016.

(2) 计算新的协方差 $\Sigma_i' = \frac{\sum_{j=1}^{m} \gamma_{ji}(x_i - \mu_i')(x_j - \mu_i')^T}{\sum_{j=1}^{m} \gamma_{ji}}$;

(3) 计算新的混合系数 $\alpha_i' = \frac{\sum_{j=1}^{m} \gamma_{ji}}{m}$;

上面的 E-step 与 M-step 交替进行，直至混合系数 α 基本不变为止，此时迭代收敛，对于每一个 γ_{ji}，我们就可以通过其在不同混合成分下的概率大小，选出概率最大的那一种对应的 cluster 作为该样本最后的分类。我们同样地利用 Python 实现了该算法，具体可见 `GMM.py` 和 `main_GMM.py`。具体分类结果可见图。

但是因为 EM 算法本质上仍是一个近似算法，不可避免地会出现陷入局部最优的情况，并且通过多次实验我们可以发现，对比已给出的数据集分类结果，GMM-EM 算法陷入局部最优的情况达到了一半以上。此外，当起始点的位置随机效果不好，两个初始 μ 的距离过近时，很容易造成协方差矩阵 Σ 奇异或因为两个 μ 相同导致其中一个的 $\sum_{j=1}^{m} \gamma_{ji}$ 为零，最终导致计算无法进行的意外情况。

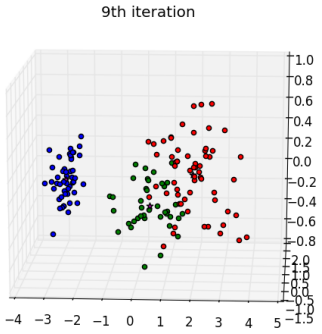

图 6: 迭代次数 =10

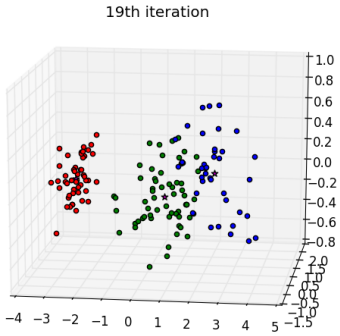

图 7: 迭代次数 =20

当然，因为 GMM 是针对的不同概率下的分布特征进行的分类，在一个点被选择是否要被归入其中的一类时是直接基于概率大小的，这也导致了 GMM-EM 算法进行的优化可能是非凸优化的，这也是在解决比较大的数据集聚类时比较好的性质，因为可以由非凸优化性得出它必然可以进行更为多样的分布聚类。相比之下，k-means 就只能生成凸（超）多面体的聚类结果。