信息基础 II **2023 秋**

Homework 5

学生: 刘浩 (202120120312) *时间: 10 月 12 日*

1.1 实验要求

有 3 类二维空间点，A 类、B 类、C 类。

A 类点以 (0,0) 为中心、(1, 0; 0,1) 为协方差矩阵的二维高斯分布；B 类点以 (2, 0) 为中心、(0.5,0; 0,1) 为协方差矩阵的二维高斯分布；C 类点以 (0, 2) 为中心、(1,0.3; 0.3,1) 为协方差矩阵的二维高斯分布；

随机生成 25 个 A 类点，25 个 B 类点，25 个 C 类点，用 K Means 进行聚类。对于 K Means 的聚类结果使用不同颜色画出聚类后不同类别的点

比较 Random/Farthest Point/K Means++ 三种初始化方法对结果的影响。

1.2 实验原理

1.2.1 K Means 算法原理

K Means 聚类算法是一种无监督分类算法，与前面几次实验不同，本次实验标签是未知的，我们需要在仅仅知道一堆点的情况下去进行分类，其步骤大概如下：

(1) 初始化 k 个中心点，也叫做初始化 k 个簇

(2) 然后计算所有点到初始化中心点的欧氏距离，距离中心最近的点划分为一类

(3) 更新中心点，采取的是对于同一类的点取平均值后作为新的中心点

不断迭代，直到我们的分类结果不变，至此聚类完成

K Means 初始化的时候有三种方式，分别为 Random，Farthest Point 和 K Means++

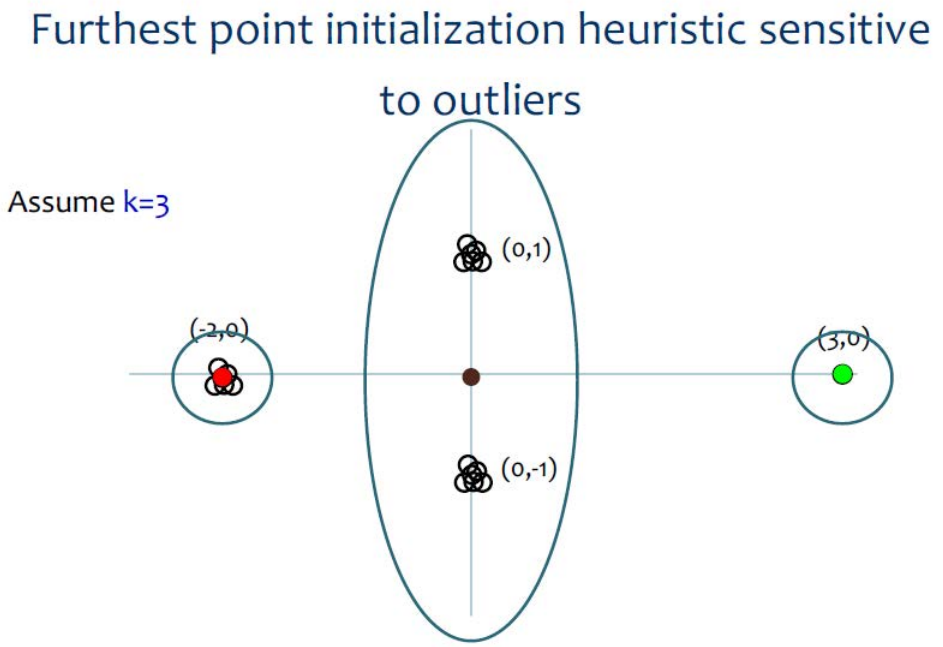
(1) 首先是 Random 方法，这样产生初始化产生的随机点随机化程度太大了，很容易聚类出非常不好的结果

(2)Farthest Point 方法，选取尽可能远的 k 个样本作为聚类中心点，其步骤可以描述如下：

从样本中随机选取一个样本点作为第一个中心点 C_1，之后遍历所有样本点，求样本点到 C_1 的距离，选

取距离 C_1 最远的样本点作为第二个中心点 C_2。再遍历所有样本点，分别求样本点到 C_1 和 C_2 的距离，之后选取其中较小的距离，在最小距离中选择其中最大的样本点，作为第三个中心点 C_3，依次类推，直到确定 C_k 至此确定 K 个聚类中心

这种聚类方法显然有个巨大的缺陷，对于很远处有一个噪点的话，会直接把这个点当作中心点，显然聚类效果是肯定不好的, 如下图所示：



(3)K Means++ 方法，这种方法是相较于前两种来说比较好的方法，此处引入了概率进行中心点的选择，避免了方法 (2) 中，最远处噪点收敛的情况，其步骤可以描述如下：

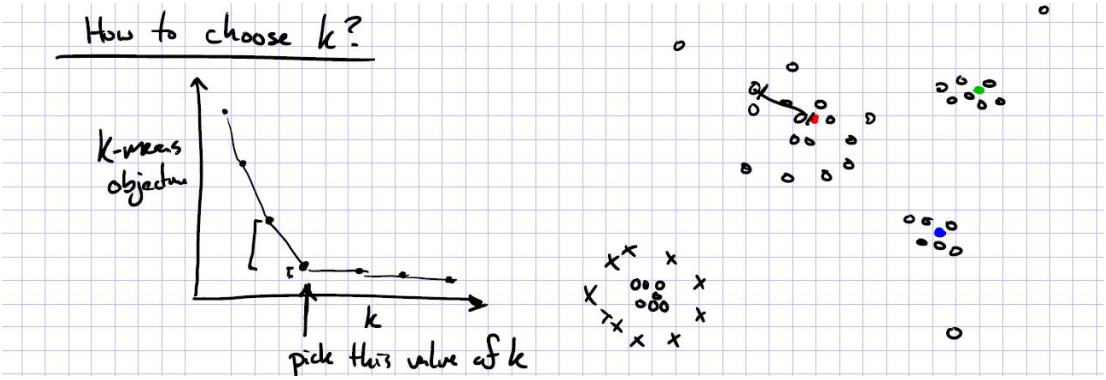
首先，在数据集中随机选择一个样本点作为第一个初始化的聚类中心，然后计算样本中的每一个样本点与已经初始化的聚类中心之间的距离，并选择其中最短的距离，记为 d_i，再按照：距离较大的点，被选取作为聚类中心的概率较大的原则，选择一个新的数据点作为新的聚类中心

重复上述过程，直到 k 个聚类中心都被确定

1.2.2 K 的选择

在本实验中其实 K 是不需要选择的，因为我们很明确的知道 K 就是 3，因为生成了三类点，但是如果在解决其他问题的时候 K 的值是不知道的，我们该如何选择呢？课上也涉及到了这个知识，如下图所

示:



我们只需要引入代价函数 SSE:

$$d\left(x, C_i\right)=\sqrt{\sum_{j=1}^{m}\left(x_j-C_{ij}\right)^2}$$

$$SSE=\sum_{i=1}^{k}\sum_{x\in C_i}\left|d\left(x, C_i\right)\right|^2$$

我们选择 SSE 关于 K 的曲线中的' 肘点',也就是下降速度骤减的拐点处即可

1.3 实验步骤

首先是数据的生成,在 hw3 中我们生成了两组二维的高斯数据,因此我们只需要稍作修改生成 A B C 三组数据即可,生成之后拼接形成我们的数据集

```
def Gauss_data():
    # 设置均值和协方差矩阵
    meanA = np.array([0, 0])
    meanB = np.array([2, 0])
    meanC = np.array([0, 2])
    cov_matrixA = np.array([[1, 0], [0, 1]])
    cov_matrixB = np.array([[0.5, 0], [0, 1]])
    cov_matrixC = np.array([[1, 0.3], [0.3, 1]])
    num_samples = 25

    # 生成随机数据点
    np.random.seed(1)
    random_pointA = np.random.multivariate_normal(meanA, cov_matrixA, num_samples)
    random_pointB = np.random.multivariate_normal(meanB, cov_matrixB, num_samples)
    random_pointC = np.random.multivariate_normal(meanC, cov_matrixC, num_samples)
```

```
    return random_pointA, random_pointB, random_pointC

    random_pointA, random_pointB, random_pointC = Gauss_data()
    dataset = np.vstack((random_pointA, random_pointB, random_pointC))
```

然后是我们的三种初始化 k 个中心点的方法

首先是随机 Random 初始化方法，采用随机选择 k 个索引，然后将这三个索引对应的点作为初始点即可

```
def Random_init(self):
    # 随机选择k个初始点
    indices = np.random.choice(self.num_data, self.k, replace=False)
    initial_points = self.dataset[indices]
    return initial_points
```

然后是 Farthest Point 初始化方法，在此处是以 k=3 举个例子具体该怎么做，首先随机选择一个点，然后计算其他的点到该点的距离，让最远的点作为第二个中心点，然后再按照聚类方法，计算到这两个中心点距离最小值，取其中的最大值作为第三个中心点，至此完成初始中心点的构造

```
def furtherst_init(self):
    indices = np.random.choice(self.num_data, 1, replace=False)
    initial_points = self.dataset[indices]
    initial_points = np.reshape(initial_points, (1, -1))

    for _ in range(1, self.k):
        distances = np.zeros(self.num_data)
        for i in range(self.num_data):
            point_distances = np.linalg.norm(self.dataset[i] - initial_points, axis=1)
            distances[i] = np.min(point_distances)

        farthest_point_index = np.argmax(distances)
        farthest_point = self.dataset[farthest_point_index]

        initial_points = np.vstack((initial_points, farthest_point))

    return initial_points
```

然后是 K means++ 方法，采用概率的方式，选取到避免某些很远的异常点的情况，使用概率的方式进行选择点，先初始化一个中心点，然后利用 $\frac{\text{distance}}{\sum \text{distance}}$ 即可计算概率，再依概率选择另外的中心点即可实现 K means++ 初始化方法

```
def kmeans_add_init(self):
    # 随机选择第一个初始点
    init_index = np.random.choice(self.num_data, 1, replace=False)
    init_point = self.dataset[init_index]

    for _ in range(1, self.k):
        distances = np.zeros(self.num_data)
```

```
        for i in range(self.num_data):
            point_distances = np.linalg.norm(self.dataset[i] - init_point, axis=1)
            distances[i] = np.min(point_distances)

        prob = distances / np.sum(distances)  # 计算每个样本被选择为新的初始点的概率
        next_index = np.random.choice(self.num_data, 1, p=prob)  # 根据概率随机选择一个样本
        next_point = self.dataset[next_index]

        initial_points = np.vstack((init_point, next_point))

    return initial_points
```

训练更新的时候我们需要找到这个点距离哪个中心点最近并且保存下来，为此我完成 find_close 函数，首先初始化一个保存所有数据点距离哪个中心点的 id 矩阵，然后我们计算这些点到这 k 个中心点的距离，寻找其中最小的距离的索引，存储到我们的矩阵中即可

```
# 寻找距离其他中心点最短的距离
def find_close(self, center):
    close_id = np.zeros((self.num_data, 1))
    for data_index in range(self.num_data):
        distances = np.linalg.norm(self.dataset[data_index] - center, axis=1)
        closest_center_index = np.argmin(distances)
        close_id[data_index] = closest_center_index
    return close_id
```

还有就是找到最近的中心点后我们需要更新中心点，为此我完成 update 函数，思路主要找到所有点距离哪个中心点最近的矩阵中，对于同一簇的点，我们对他取平均值，作为新的中心点，至此完成更新过程

```
# 更新中心点的位置
def update(self, close_id):
    center_update = np.zeros((self.k, self.dataset.shape[1]))  # 初始化更新后的中心点

    for i in range(self.k):
        # 找到属于第i个簇的数据点的索引
        cluster_indices = np.where(close_id == i)[0]

        if len(cluster_indices) > 0:
            # 计算该簇内所有数据点的均值作为新的中心点位置
            center_update[i] = np.mean(self.dataset[cluster_indices], axis=0)  # axis = 0表示按列求平均
                值

    return center_update
```

最后就是训练过程了，我们只需要循环 epoch 次，不断执行找最近距离以及更新中心点即可：

```
def train(self, epoch):
    # # Random方法选择初始点
    # center = self.Random_init()
```

```
    # # 最远法选择初始点
    center = self.furtherst_init()
    # k_means++方法找初始点
    # center = self.kmeans_add_init()
    # 找这七十五个点到哪个中心点距离最近，初始矩阵75*1
    for _ in range(epoch):
        close_id = self.find_close(center)
        # 更新中心点的位置
        center = self.update(close_id)

    return center, close_id
```

1.4 实验探究

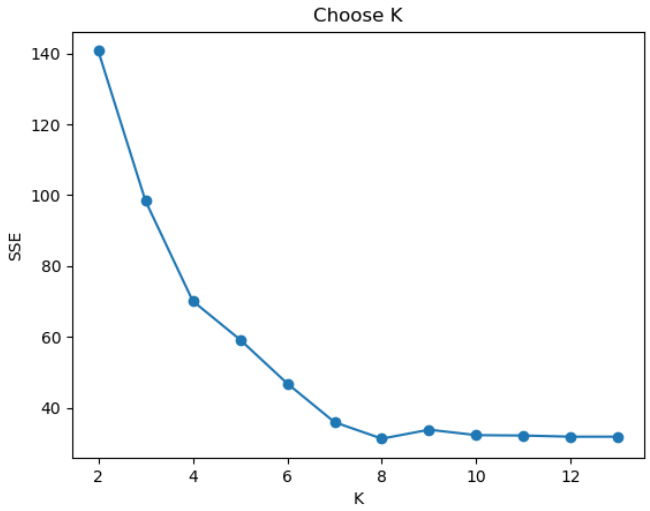
在这里我对 K 的选择进行了探究，首先我定义了计算 SSE 的函数:

```
def calculate_sse(dataset, center, close_id):
    sse = 0
    for i in range(center.shape[0]):
        cluster_indices = np.where(close_id == i)[0]
        sse += np.sum(np.square(dataset[cluster_indices] - center[i]))
    return sse
```

然后就是对于不同的 K 值去计算对应的 SSE 并且画图，只需要遍历 K 进而计算 SSE 并且可视化即可：

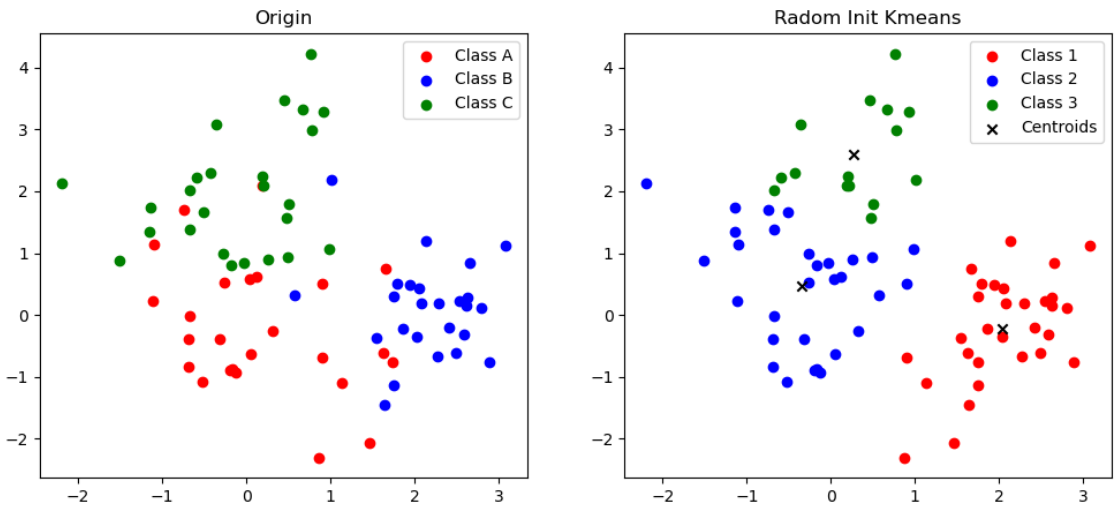
```
def choose_k(dataset, max_k):
    k_values = []
    sse_values = []

    for k in range(2, max_k + 1):
        MyKmeans = kmeans(dataset, k)
        center, close_id = MyKmeans.train(epoch=1000)
        sse = calculate_sse(dataset, center, close_id)

        k_values.append(k)
        sse_values.append(sse)

    plt.plot(k_values, sse_values, marker='o')
    plt.xlabel('K')
    plt.ylabel('SSE')
    plt.title('Choose K')
    plt.show()
```

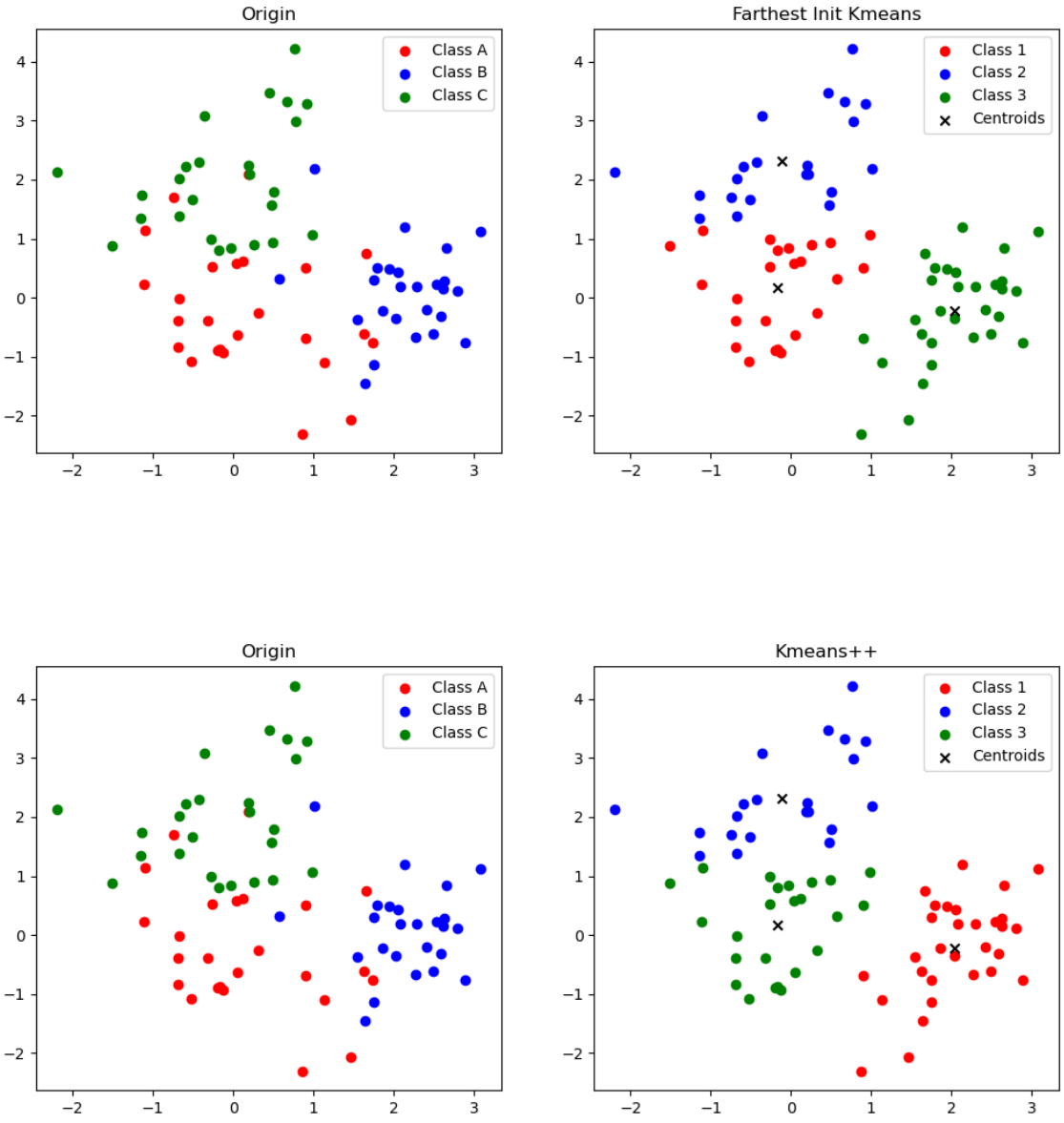
最终我得到的结果如下图所示



为此我们选择的“肘点”为 K=3 与我们期望得到的 K 是相同的，也验证了这个方法的可行性，对于未知数据的类别的时候，我们就可以按照这个方法去选择最佳的 K，更加具有普适性！

1.5 实验结果





1.6 实验感想

本次实验是无监督学习，是在不给出标签的情况下进行分类，其实原理是相对前几次实验来说比较简单的，主要是对比这三种初始化的不同，很明显 K Means++ 相对另外两种效果是最好的，可以很好的避免另外两种的缺陷，分类效果也是比较好的，同时在本实验中我还探究了选择 K 的方法，收获颇丰，知道了该如何应对无标签点进行分类的任务！

1.7 参考资料

[1] https://zhuanlan.zhihu.com/p/619739126?utm_id=0

[2] http://t.csdnimg.cn/ES1zC

[3] https://zhuanlan.zhihu.com/p/78798251