山东大学

信息科学与工程学院

2022－2023 学年第一学期

实 验 报 告

课程名称：信息基础 II

专 业 班 级：崇新学堂

学 生 学 号：

学 生 姓 名：

课 程 报 告：k-means 聚类方法

1. 作业要求

有 3 类二维空间点，A 类、B 类、C 类。
A 类以(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，并用 K-Means 进行聚类，并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。

2. 理论部分

2.1 数据向量格式说明

$$X=\begin{bmatrix} x_1^{(1)} & \cdots & x_n^{(1)} \\ \vdots & \vdots & \vdots \\ x_1^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \qquad Z=\begin{bmatrix} z_1 & \cdots & z_m \end{bmatrix} \qquad C=\begin{bmatrix} c_1^{(1)} & \cdots & c_n^{(1)} \\ \vdots & \vdots & \vdots \\ c_1^{(k)} & \cdots & c_n^{(k)} \end{bmatrix}$$

X 是输入的样本点，共计 m 个样本，每个样本的特征为 n 维度。Z 矩阵用于存储每个样本所对应的分类类别。C 矩阵用于存储每个类别的聚类中心点。

2.2 K-means 初始化方法

K-means 是一种局部搜索的聚类搜索算法，它的最终结果依赖于初始点的选择，很容易陷入局部最优解，而无法获得全局最优解，因此提出了很多初始化的方法：

1). 在给定了 K 之后，从样本点中随机选取 K 个样本点作为初始聚类中心，多次选取，使用损失函数进行评估，选取效果最好的一组。这种方法的缺点是存在过大的随机性，很容易产生不良的聚类结果。

2). 选取距离尽量远的 K 个样本点作为聚类中心：从样本中随机选取一个样本点作为第一个中心点 C1，之后遍历所有样本点，求取样本点到 C1 的距离，选取距离 C1 最远的样本点作为第二个中心点 C2。针对第三个中心点的求取：仍然需要遍历所有样本点，分别求取样本点到 C1 和 C2 的距离，之后选取其中较小的距离作为最后的选择标准，所有样本点中选择标准最大的样本点，作为第三个中心点 C3，其数学表达如下：

$$C_3=\arg\max_{x^{(i)}\in D}[\min(\| x^{(i)}-C_1\|^2,\| x^{(i)}-C_2\|^2)] \tag{1}$$

更一般化的表达为：

$$C_k=\arg\max_{x^{(i)}\in D}[\min(\| x^{(i)}-C_1\|^2,\| x^{(i)}-C_2\|^2,\cdots\| x^{(i)}-C_{\text{k-1}}\|^2)] \tag{2}$$

依据这种方法所选择出的中心点能够达到距离最大化，能够尽可能分散地实现聚类，但是这种方法也存在明显的缺点：依靠最大距离所选择的中心点容易受到离群点和异常点的影响，从而造成不良的聚类分析结果。

3). 在第二种方法的基础上，引入概率，距离已初始化中心点越远的点，被选为下一个中心点的概率越大，从而可以有效地避免离群点和异常点的影响，编程实现的流程如下：

1. 随机选取一个样本点作为第一个中心点

2. **repeat**

3. 对于每一个非中心点，计算它于最近的中心点之间的距离 D（x）并保存在一个数组中，然后把这些距离加起来得到 Sum(D(x))

4. 取一个落在 Sum(D(x))范围中的随机值 R，重复计算 R=R-D（x）直至得到 R<0，选取此时的点作为下一个中心点。

5. **until** 选取了 k 个中心点。

以上算法被称为 K-means++，观察以上过程可以发现，距离现有中心点越远的点被选中的概率越大，可以有效避免离群点、异常点所造成的影响，因此是很有效的初始化中心点方法。

2.3 K-means 方法

在给定训练集集合并且初始化完毕后，反复进行一下过程：

$$c = \arg\min_{c} \sum_{i=1}^{n} || x^{(i)} - c_{z^{(i)}} ||^2 \tag{3}$$

$$z = \arg\min_{z} \sum_{i=1}^{n} || x^{(i)} - c_{z^{(i)}} ||^2 \tag{4}$$

我们将这两个过程分为优化 C 和优化 Z 的过程，反复进行以上两个过程，即可完成聚类。

2.3.1 C 优化过程

C 的优化过程用语言描述就是，找到当前被划分为该类的所有样本点，求取对应样本点的各个特征的均值，即为新的中心点，其数学表达如下:

$$c_j = \frac{1}{N_j} \sum_{i:z^{(i)}=j} x^{(i)} \tag{5}$$

其中 N_j 表示属于该类别下的所有样本点的个数。

2.3.2 Z 优化过程

Z 的优化过程比较简单，对于每一个样本点，求取每个样本点到每个聚类中心点的距离，之后选取距离最近的聚类中心点，作为新的分类标签，其数学表达如下:

$$z_i = \arg\min_{j} || x^{(i)} - c_j ||^2 \tag{6}$$

按照以上流程反复迭代即可获得聚类结果。

2.4 如何选择 K

由于 k-means 方法适用的场景为无监督学习，虽然在本次作业中我们已知分类为三点，但是对待更一般的问题时间不再使用，所以在此我们讨论应当如何选择 k。我们在此引入偏差，即每个点到分类中心的总距离，如下表达:

$$e = \sum_{i=1}^{n} || x^{(i)} - c_{z^{(i)}} ||^2 \tag{5}$$

由此我们可以绘制出 k 和偏差之间的折线图，我们选取夹角最小的点作为我们选取的 k 值点。由此完成了 k 的选取

3. 代码实现

3.1 数据产生部分

利用高斯随机数产生函数和 for 循环创建了三类高斯随机点和标签值，同时使用 numpy 库中的 vstack 函数将三种不同的数据点堆叠起来，得到统一的 x。此处创建标签值并不是用于训练，而是用于后续绘图进行对比。

```
def Gauss_random(mean1, cov1, mean2, cov2, mean3, cov3, number):
    x1 = np.random.multivariate_normal(mean1, cov1, number)
    x2 = np.random.multivariate_normal(mean2, cov2, number)
    x3 = np.random.multivariate_normal(mean3, cov3, number)
    y = []
    for i in range(3):
        for j in range(number):
            y.append(i)
    x4 = np.vstack((x1,x2))
    x = np.vstack((x3,x4))
    return x,y
```

3.2 最小距离求取函数

该函数的输入变量为样本中的一个特征点和已经完成初始化的中心点的集合，此函数用于求解该样本点到已经初始化的中心点的最小距离，该函数主要在 k-means++方法中调用，主要用于初始化函数中。

```
def Mini(x, centers):
    """
    :param x: 一个样本点
    :param c: 各类别中心点
    :return: 最小距离
    """
    min_dist = 2000000000000000.0
    m = np.shape(centers)[0]  # 当前已经初始化聚类中心的个数
    n = len(x)
    for i in range(m):
        # 计算该样本与每个聚类中心之间的距离
        d = 0
        for j in range(n):
            d += (x[j]-centers[i,j])**2
        # 选择最短距离
        if min_dist > d:
            min_dist = d
    return min_dist
```

3.3 随机选取初始化函数

该函数的输入变量为训练集和分类个数，本函数随机选取 k 个点作为初始中心点。

```python
def Initialization_a0(x, k):
    index = []
    m, n = np.shape(x)
    centers = np.mat(np.zeros((k, n)))              #建立中心点向量矩阵
    for i in range(k):                              #随机产生k个随机数
        a = np.random.randint(0, m)
        if a not in index:                          #如果产生的随机数在已有列表中不存在，则直接选为中心点
            index.append(a)
            centers[i] = np.copy(x[a])
        else:
            for j in range(100):                    #如果产生的随机数在列表中已经存在，则需要重新产生随机数
                a = np.random.randint(0, m)
                if a not in index:
                    index.append(a)
                    centers[i] = np.copy(x[a])
                    break
    return centers
```

3.4 最远点初始化函数

该函数实现最远点初始化，按照 2.2 中所述的方法进行编程，首先随机选取第一个中心点，之后依照最远准则完成初始点的初始化。

```python
def Initialization_a00(x, k):
    m, n = np.shape(x)
    centers = np.mat(np.zeros((k, n)))
    # 随机选取一个样本点作为第一个聚类中心
    index = np.random.randint(0, m)
    centers[0] = np.copy(x[index])
    # 初始化最小距离序列
    max_d = 0
    distance = [0.0 for _ in range(m)]
    for i in range(1, k):                                   #针对第i个中心点
        for j in range(m):                                  #针对第j个样本
            distance[j] = Mini(x[j], centers[0:i], )        #求取每个样本到达所有已经初始化中心点的最小距离
        max_d = max(distance)                               #选取最小距离最大的样本点作为下一个中心点
        for j in range(m):
            if distance[j] == max_d:
                centers[i] = np.copy(x[j])
    return centers
```

3.5 计算偏差函数

在此函数中，遍历所有样本并且求取其到分类中心点之间的距离，最后求和，即可获得不同 k 值下的偏差值。

```python
k = []
E = []
for i in range(1,10):
    centers1 = Initialization_a2(x, i)
    z1, centers1 = K_means(x,centers1)
    k.append(i)
    E.append(Error(z1, centers1, x))
print(Choose_k(k, E))
plt.plot(k, E, marker='o', mec='r', mfc='w')
plt.show()
```

3.6 k-means++初始化方法

k-means++方法引入了概率，且距离越远的点被选中的概率越大，而概率的实现依赖于随机数，具体实现过程已在 2.2 中做出阐述，在此不加赘述。

```python
def Initialization_a2(x, k):
    """
    :param x:样本
    :param k:k—means参数
    :return:各类别中心点
    """
    m, n = np.shape(x)
    centers = np.mat(np.zeros((k, n)))
    # 随机选取一个样本点作为第一个聚类中心
    index = np.random.randint(0, m)
    centers[0, ] = np.copy(x[index, ])

    distance = [0.0 for _ in range(m)]                          #用于存储距离
    for i in range(1, k):
        sum_dis = 0
        for j in range(m):
            distance[j] = Mini(x[j, ], centers[0:i], )
            sum_dis += distance[j]                              #将所有最小距离求和

        sum_dis *= np.random.random()                           #在距离和的范围内随机选取一个数字
        for j, dj in enumerate(distance):
            sum_dis -= dj
            if sum_dis < 0:                                     #持续作差直至找到使得作差结果为负数
                centers[i] = np.copy(x[j, ])
                break
    return centers
```

3.7 Z 优化过程

Z 的优化过程需要求取每个样本到中心点的距离，构建了一个距离列表，用于存储样本点到每个聚类中心点的距离，选取列表中的最小项，其最小项对应的索引就是更新后 Z 的种类，通过 for 循环遍历所有样本和所有中心点，之后便可完成对于 Z 的优化过程，具体优化代码如下图所示：

```python
def Z_optimize(x,z,centers):
    '''
    :param x: 样本集合
    :param z: 对应点归类
    :param centers: 各类中心点坐标
    :return: 更新后的归类
    '''
    for j in range(len(x)):
        dj = [0 for _ in range(len(centers))]               #用于存储样本点到每个中心点的距离
        for i in range(len(centers)):                       #循环求取样本到每个点的距离
            d = 0
            index = 0
            for a in range(len(x[0])):
                d += (x[j,a]-centers[i,a])**2
            dj[i] = d
        min_dis = min(dj)                                   #将距离最近的聚类中心点更新为样本的分类
        for c in range(len(dj)):
            if min_dis == dj[c]:
                index =c
        z[j] = index
    return z
```

3.8 C 优化过程

C 优化的过程，只需要根据 Z 中的数值将数据点进行划分，并且根据分类将各个种类下的数据点进行求和，随后除以该类别下的总的数据点的个数，即可完成 C 的优化过程。

```python
def C_optimize(x,z,centres):
    '''
    :param x: 样本集合
    :param z: 所述分类中心标签
    :param centres: 分类中心点
    :return: 更新后的分类中心点
    '''
    num =[0 for _ in range(len(centres))]                    #用于存储每个类别下的数据点的个数
    d = np.mat(np.zeros((len(centres),len(x[0]))))           #用于存储不同种类下各个数据点求和
    for i in range(len(centres)):
        for j in range(len(z)):
            if z[j] ==i:
                d[i] = d[i]+x[j]
                num[i] += 1
    for i in range(len(centres)):
        d[i] = d[i]/num[i]                                   #不同种类下的数据点求和除以对应的数据点个数
    return d
```

3.9 聚类分析函数

该函数输入变量为所有数据点集合，输出聚类结果，其过程为调用初始化函数，设置迭代次数，之后反复调用优化函数，完成最终迭代。

```python
def K_means(x):
    centers = Initialization_a2(x,3)
    z = [0 for _ in range(len(x))]
    n= 1000
    for i in range(n):
        z = Z_optimize(x, z, centers)
        centers = C_optimize(x,z,centers)
    return z,centers
```

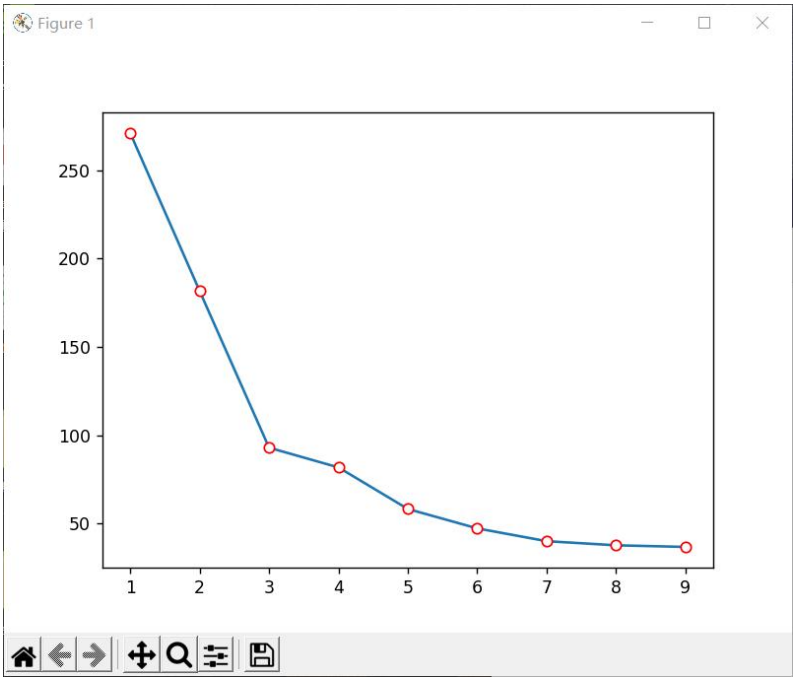
3.10 k 值选取函数

使用计算偏差函数之后，得到了 k 和偏差值之间的折线图，接下来要求取夹角最小的点，借助向量知识来完成，夹角绝大多数为钝角，因此我们获得一个点和前后两个点形成的向量之后，将两个向量进行单位化，之后计算两个向量的点积，从而求取出所有向量点积中的最大值即可确定 k 值。

```python
def Choose_k(k, E):
    MAX = -2
    k_final = 0
    a = 0
    for i in range(1, len(k)-1):
        k1 = [-1, E[i-1] - E[i]]
        k2 = [1, E[i+1] - E[i]]
        print(k1,k2)
        k1[0] /= (k1[0]**2 + k1[1]**2)**0.5
        k1[1] /= (k1[0] ** 2 + k1[1] ** 2)**0.5
        k2[0] /= (k2[0]**2 + k2[1]**2)**0.5
        k2[1] /= (k2[0] ** 2 + k2[1] ** 2)**0.5
        a = k1[0]*k2[0] + k1[1]*k2[1]
        print(a)
        if a > MAX:
            MAX = a
            k_final = i+1
    return k_final
```

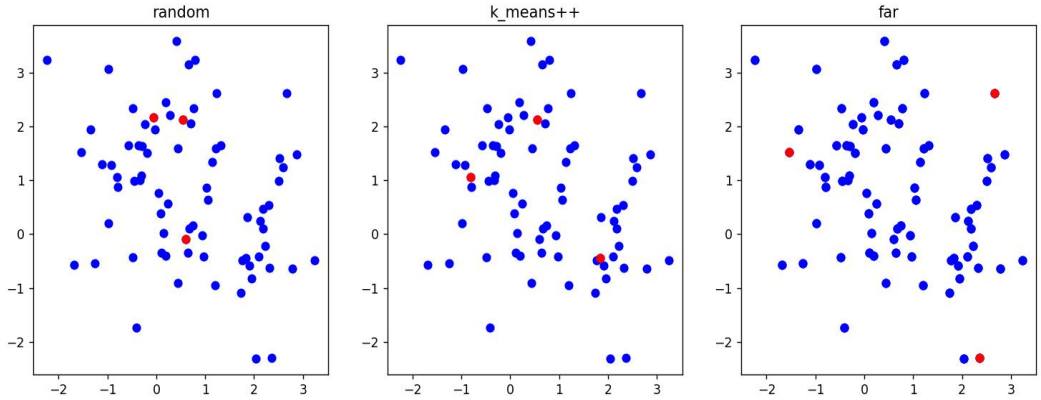
4. 实验结果

4.1 不同 k 的选择：



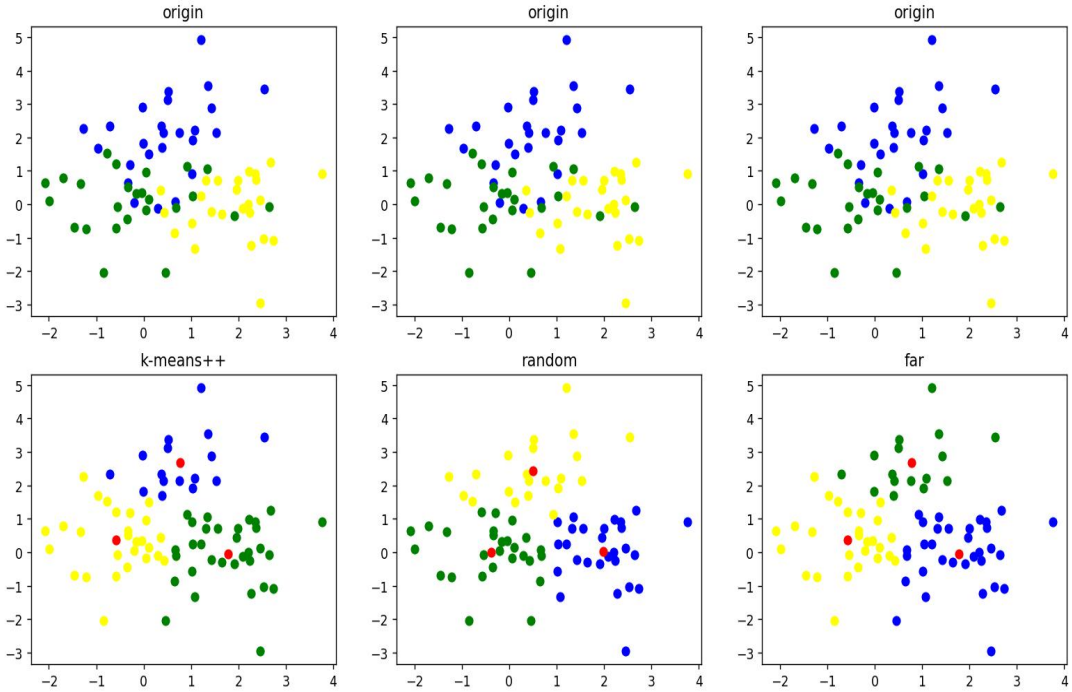
通过上图我们可以发现，在 k 和偏差的曲线中，在 k=3 这一点处，两条折线的夹角是最小的，因此我们在本次实验中将 k 选择为 3，进行后续的分析。这与我们初始的数据种类是一致的，由此我们证明这种 k 的选择方法是合理的，从而可以用于本次实验。

4.2 不同初始化方法的对比：

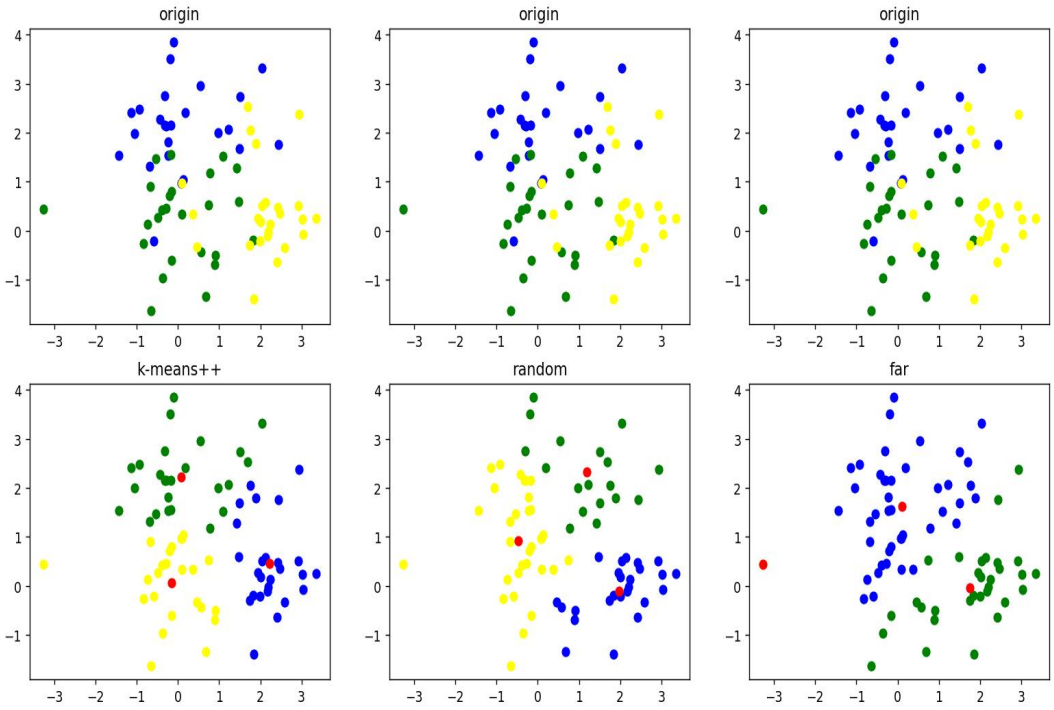


由上图可以得出不同初始化方法的区别：1. 随机选择法的随机性过强，可能存在不稳定的现象，同时选取的两个点也可能过近，如图所示。2. 选取最远点的方法容易受到异常点和离群点的影响，如上图所示，三个初始点的选择很分散，这满足我们的需求，但是很容易受到过于边角的离群点和异常点的影响，这可能会使得最终的分类结果不理想。3. k-means++的初始化方法是目前来看最有效的初始化方法，从图中可以看出其初始化的中心点分布较为分散且合理，不易受到离群点和异常点的影响。

4.3 不同初始化方法对于最终结果的影响



上图是采用三种不同初始化结果得到的聚类结果，其中的不同颜色仅代表不同类型的点，而不具有其他的意义，第一排为原始的数据点分类，第二排为通过三种不同方法聚类的结果，可以观察到三种初始化方法最终的表现是相差不大的，但是通过观察我们可以发现对于个别的一些点聚类表现还有待提高，之前提及到随机选取和最远点的缺点，其缺点体现在下图：



通过观察上图右下角，我们可以明显发现采用最远点初始化的方法的弊端，受到了离群点的影响，造成了极差的聚类分析结果。而采用随机点进行初始化的方法，在目前的多次运行中，并没有出现极其糟糕的情况。

5. 实验心得

此次作业的知识在上课时听得比较清晰，在实现的过程中遇到的唯一阻碍是k-means++中的依概率选择中心点应当怎么实现，在上网查阅了相关材料之后，很快有了思路，利用随机数来实现这一过程，在本次作业的整个过程中思路和进展相对都比较顺利，看到了 k-means 方法实际的聚类效果，也对无监督学习有了更加直观的认识。辛苦老师、学长学姐!!