暨南大学本科实验报告专用纸

课程名称 数据挖掘 成绩评定

实验项目名称 K-means 算法的实现与改进 指导教师 韩旭明

实验项目编号 0806011602 实验项目类型 验证性 实验地点 DF116

学生姓名 陈宇 学号 2020101642

学院 信息科学技术学院 系 计算机 专业 软件工程

实验时间 2023 年 4 月 7 日 上午～4 月 7 日 下午 温度 ℃湿度

一、实验目的

1. 掌握无监督聚类分析的思想与主要类型划分；
2. 掌握 K-means 聚类算法实现步骤；
3. 了解 K-means 聚类算法的优点和缺陷；
4. 掌握部分 K-means 聚类算法的创新和改进；
5. 掌握数据挖掘聚类算法的相关应用。

二、实验环境

安装 Python，使用 Python 环境，完成 K-means 算法的程序实现。

三、实验步骤

1. 无监督聚类分析的思想与主要类型划分；
2. K-means 算法实现步骤；
3. 使用 Python 环境，完成 K-means 算法的程序实现；
4. 文献查找 K-means 聚类算法的主要创新及改进；
5. 文献查找 K-means 算法的相关应用。

四、实验内容

1. 无监督聚类分析的思想与主要类型划分：

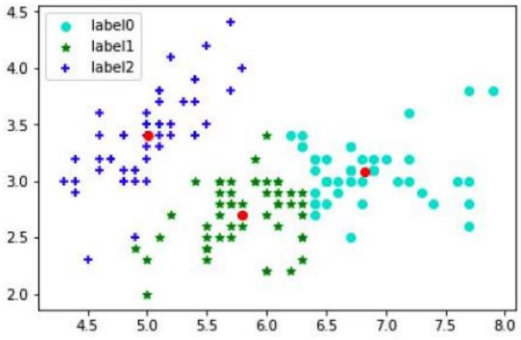
将多个无明显分类特征的对象，按照某种相似性分成多个簇的分析过程。
目前有许多聚类算法和技术：

1. 基于划分的方法

基于划分的聚类方法是一种自顶向下的方法，对于给定的 n 个数据对象的数据集 D，将数据对象组织成 k(k≤n) 个分区，其中，每个分区代表一个簇。

基于划分的聚类方法是一种自顶向下的方法，对于给定的 n 个数据对象的数据集 D，

将数据对象组织成 k(k≤n) 个分区，其中，每个分区代表一个簇。



2. 基于层次的方法

基于层次的聚类方法是指对给定的数据进行层次分解，直到满足某种条件为止。该算法根据层次分解的顺序分为自底向上法和自顶向下法，即凝聚式层次聚类算法和分裂式层次聚类算法。

3. 基于密度的方法

基于密度的聚类方法的主要目标是寻找被低密度区域分离的高密度区域。与基于距离的聚类算法不同的是，基于距离的聚类算法的聚类结果是球状的簇，而基于密度的聚类算法可以发现任意形状的簇。

基于密度的聚类方法是从数据对象分布区域的密度着手的。如果给定类中的数据对象在给定的范围区域中，则数据对象的密度超过某一阈值就继续聚类。

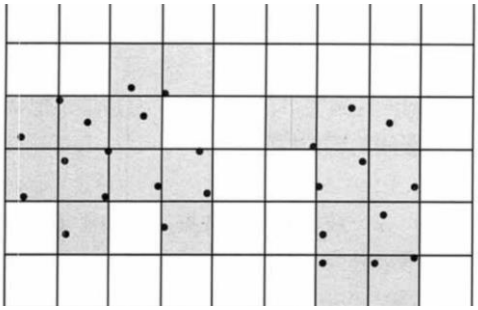
这种方法通过连接密度较大的区域，能够形成不同形状的簇，而且可以消除孤立点和噪声对聚类质量的影响，以及发现任意形状的簇。

4. 基于概率的方法

数据集 D 被认为是由这 K 个簇产生的，有了这一个前提之后，基于概率模型的聚类分析的任务是推导出最可能产生数据集 D 的 K 个聚类簇。接下来就是度量 K 个聚类簇的集合和它们的概率产生观测数据集的似然。

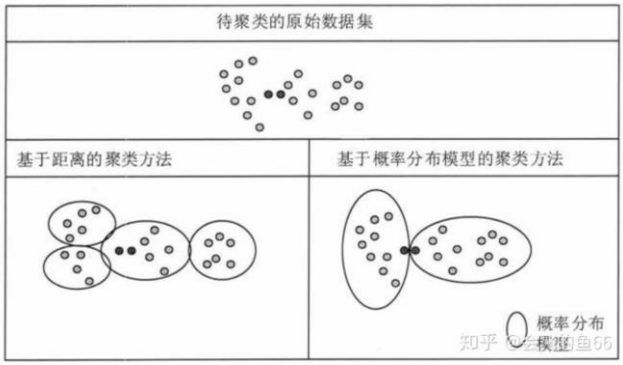
5. 基于网格的方法

基于网格的聚类方法将空间量化为有限数目的单元，可以形成一个网格结构，所有聚类都在网格上进行。基本思想就是将每个属性的可能值分割成许多相邻的区间，并创建网格单元的集合。每个对象落入一个网格单元，网格单元对应的属性空间包含该对象的值。



6. 基于模型的方法

基于模型的聚类方法是试图优化给定的数据和某些数学模型之间的适应性的。该方法给每一个簇假定了一个模型，然后寻找数据对给定模型的最佳拟合。假定的模型可能是代表数据对象在空间分布情况的密度函数或者其他函数。这种方法的基本原理就是假定目标数据集是由一系列潜在的概率分布所决定的。



2. K-means 算法实现步骤：

1. 数据预处理

剔除离群点，数据归一化，数据标准化

（来源于输入，来源于真实数据，存在不完美的情况，要进行预处理）

2. 初始化

随机选择 K 个中心点$\mu_1^{(0)},\mu_2^{(0)},\ldots,\mu_K^{(0)}$（根据实际确定 K 值，上标代表迭代次数）

3. 定义损失函数

$$J(c,\mu)=\min\sum_{i=1}^{K}\sum_{x\in c_i}\|x-\mu_{c_i}\|^2$$

（机器学习，数据挖掘算法都有损失函数，按照损失函数的学习方法来学习）

t 为迭代步骤，重复下面过程至 J 收敛

1） 对于每个样本点，将其分配到距离最近的簇：

$$c_i^{(t)}\leftarrow\operatorname{argmin}_k\|x-\mu_{c_i}\|^2$$

2） 对于每个簇，重新计算聚类质心：

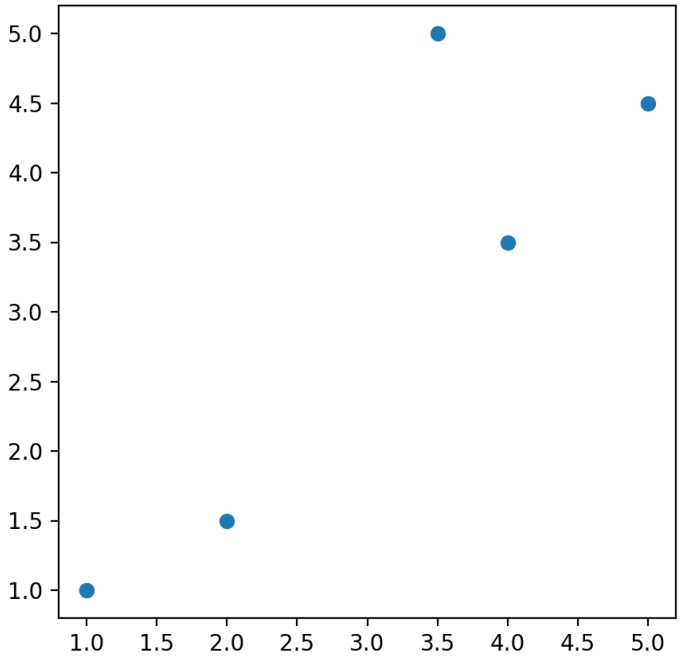
$$\mu_k^{(t+1)} \leftarrow \frac{1}{\left\|c_i^{(t)}\right\|}\sum_{x \in c_i^{(t)}} x$$

3. K-means 算法完整的程序代码：

用于 K-means 算法的数据集：

Instance	x	y
1	1.0	1.0
2	2.0	1.5
3	4.0	3.5
4	5.0	4.5
5	3.5	5

数据集散点图表示：



K-means 算法代码实现：

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

class Kmeans:
      # 计算欧氏距离
      def get_distance(self, x, y):
            return np.sqrt(np.sum(np.square(x-y)))

      # 定义中心初始化函数，中心点选择样本特征
```

```
    def center_init(self, k, X):
        n_samples, n_features = X.shape
        centers = np.zeros((k, n_features))
        selected_centers_index = []
        for i in range(k):
            # 每一次循环随机选择一个类别中心,判断不让 centers 重复
            sel_index = random.choice(list(set(range(n_samples)) -
set(selected_centers_index)))
            centers[i] = X[sel_index]
            selected_centers_index.append(sel_index)
        return centers

    # 判断一个样本点离哪个中心点近， 返回的是该中心点的索引
    ## 比如有三个中心点，返回的是 0，1，2
    def closest_center(self, sample, centers):
        closest_i = 0
        closest_dist = float('inf')
        for i, c in enumerate(centers):
            # 根据欧式距离判断，选择最小距离的中心点所属类别
            distance = Kmeans.get_distance(self, sample, c)
            if distance < closest_dist:
                closest_i = i
                closest_dist = distance
        return closest_i

    # 定义构建聚类的过程
    # 每一个聚类存的内容是样本的索引，即对样本索引进行聚类，方便操作
    def create_clusters(self, centers, k, X):
        clusters = [[] for _ in range(k)]
        for sample_i, sample in enumerate(X):
            # 将样本划分到最近的类别区域
            center_i = Kmeans.closest_center(self, sample, centers)
            # 存放样本的索引
            clusters[center_i].append(sample_i)
        return clusters

    # 根据上一步聚类结果计算新的中心点
    def calculate_new_centers(self, clusters, k, X):
```

```
            n_samples, n_features = X.shape
            centers = np.zeros((k, n_features))
            # 以当前每个类样本的均值为新的中心点
            for i, cluster in enumerate(clusters):   # cluster 为分类后每一类的
索引
                  new_center = np.mean(X[cluster], axis=0)   # 按列求平均值
                  centers[i] = new_center
            return centers

      # 获取每个样本所属的聚类类别
      def get_cluster_labels(self, clusters, X):
            y_pred = np.zeros(np.shape(X)[0])
            for cluster_i, cluster in enumerate(clusters):
                  for sample_i in cluster:
                        y_pred[sample_i] = cluster_i
                        # print('把样本{}归到{}类'.format(sample_i,cluster_i))
            return y_pred

      # 根据上述各流程定义 kmeans 算法流程
      def kmeans_exe(self, X, k, max_iterations):
            # 1.初始化中心点
            centers = Kmeans.center_init(self, k, X)
            # 遍历迭代求解
            for _ in range(max_iterations):
                  # 2.根据当前中心点进行聚类
                  clusters = Kmeans.create_clusters(self, centers, k, X)
                  # 保存当前中心点
                  pre_centers = centers
                  # 3.根据聚类结果计算新的中心点
                  new_centers = Kmeans.calculate_new_centers(self, clusters, k,
X)
                  # 4.设定收敛条件为中心点是否发生变化
                  diff = new_centers - pre_centers
                  # 说明中心点没有变化，停止更新
                  if diff.sum() == 0:
                        break
            # 返回最终的聚类标签
            return Kmeans.get_cluster_labels(self, clusters, X)
```

```
if __name__ == "__main__":
    # 绘制测试样例散点图
    X = np.matrix([[1, 1.0, 1.0],
                   [2, 2.0, 1.5],
                   [3, 4.0, 3.5],
                   [4, 5.0, 4.5],
                   [5, 3.5, 5.0]])
    x = X[:, 1]
    y = X[:, 2]

    plt.figure(figsize=(5, 5), dpi=100)
    plt.scatter(x.tolist(), y.tolist())
    plt.show()

    myKmeans = Kmeans()
    # 执行 K-means 算法 输出结果
    labels = myKmeans.kmeans_exe(X[:, 1:], 2, 10)
    # 打印每个样本所属的类别标签
    print("分类结果", label)
```

代码分类结果：

```
D:\python3.8.0\python.exe D:/code/python/神经网络/K-means.py
分类结果 [0. 0. 1. 1. 0.]

进程已结束,退出代码0
```

4. K-means 聚类算法的主要创新及改进

K-means 聚类方法的缺点和相关改进：

1. **对于离群点和孤立点敏感；**

   离群点检测的 LOF 算法，通过去除离群点后再聚类，可以减少离群点和孤立点对于聚类效果的影响。

2. **k 值选择；**

   可以通过在一开始给定一个适合的数值给 k,通过一次 K-means 算法得到一次聚类中心。对于得到的聚类中心，根据得到的 k 个聚类的距离情况，合并距离最近的类，因此聚类中心数减小，当将其用于下次聚类时，相应的聚类数目也减小了，最终得到合适数目的聚类数。可以通过一个评判值 E 来确定聚类数得到一个合适的位置停下来，而不

继续合并聚类中心。重复上述循环，直至评判函数收敛为止，最终得到较优聚类数的聚类结果。

3. **初始聚类中心的选择；**

选择批次距离尽可能远的 K 个点：首先随机选择一个点作为第一个初始类簇中心点，然后选择距离该点最远的那个点作为第二个初始类簇中心点，然后再选择距离前两个点的最近距离最大的点作为第三个初始类簇的中心点，以此类推，直至选出 K 个初始类簇中心点。

选用层次聚类或者 Canopy 算法进行初始聚类，然后利用这些类簇的中心点作为 KMeans 算法初始类簇中心点。该方法对于 k 值的选择也是十分有效的。

4. **只能发现球状簇。**

在李荟娆的硕士论文 K_means 聚类方法的改进及其应用中提到了基于 2 种测度的改进，改进后，可以去发现非负、类椭圆形的数据。

如果数据集中有不规则的数据，往往通过基于密度的聚类算法更加适合，比如 DESCAN 算法。

在这里，我尝试通过修改对初始聚类中心点的选择，每次尽量选取尽可能远的点作为质心来对 K-means 算法进行改进。

以下是我的算法描述：

1) 随机选取一个样本作为第一个聚类中心 c1；
2) 计算每个样本与当前已有类聚中心最短距离（即与最近一个聚类中心的距离），用 D(x) 表示，并计算平均距离，选取平均距离最大值的点作为另一个聚类中心 c2;
3) 重复 2，直到选出 k 个聚类中心。

以下是我对初始化质心函数 center_init 的修改:

```
# 初始质心选择距离较远的初始化函数：
def center_init_adv(self, k, X):
      global best_index
      n_samples, n_features = X.shape
      centers = np.zeros((k, n_features))
      selected_centers_index = []
      # 随机选择一个初始点
      sel_index = random.choice(list(range(n_samples)))
      centers[0] = X[sel_index]
      selected_centers_index.append(sel_index)
      for i in range(1, k): # 仍需计算 k-1 个质心
            # 目前最远的点：
            best_index = -1          # 最远平均距离
            best_dis = 0
            for j in range(n_samples):         # 对于每个其余点有
                  next_index = random.choice(list(set(range(n_samples)) -
set(selected_centers_index)))
```

```
                dis_aver = 0 # 平均距离
                for l in centers:            # 对于每个已选的质心
                    dis_aver += self.get_distance(l, X[next_index])
                dis_aver = dis_aver / len(centers)
                if dis_aver > best_dis:
                    best_dis = dis_aver
                    best_index = next_index
            centers[i] = X[best_index]
            selected_centers_index.append(sel_index)
        return centers
```

修改后运行结果：

```
D:\python3.8.0\python.exe D:/code/python/数据挖掘/K-means.py
分类结果 [0. 0. 1. 1. 1.]
```

在理论上，在 k-means 算法上对初始化质心进行优化，可以提高对算法的收敛速度。最后，经过网上查阅资料，K-means 算法的一种改进变体为 K-means++算法，其原理也是对随机初始质心进行优化，其原理是：计算每个样本与当前已有类聚中心最短距离（即与最近一个聚类中心的距离），用 D(x)表示；这个值越大，表示被选取作为聚类中心的概率较大；最后，用轮盘法选出下一个聚类中心。尽管计算初始点时花费了额外的时间，但是在迭代过程中，k-mean 本身能快速收敛，因此算法实际上降低了计算时间。

5. K-means 算法的相关应用

1. 文档分类器

根据标签、主题和文档内容将文档分为多个不同的类别。这是一个非常标准且经典的 K-means 算法分类问题。首先，需要对文档进行初始化处理，将每个文档都用矢量来表示，并使用术语频率来识别常用术语进行文档分类,这一步很有必要。然后对文档向量进行聚类，识别文档组中的相似性。 这里是用于文档分类的 K-means 算法实现案例。

2. 物品传输优化

使用 K-means 算法的组合找到无人机最佳发射位置和遗传算法来解决旅行商的行车路线问题，优化无人机物品传输过程。

3. 识别犯罪地点

使用城市中特定地区的相关犯罪数据，分析犯罪类别、犯罪地点以及两者之间的关联，可以对城市或区域中容易犯罪的地区做高质量的勘察。

4. 客户分类

聚类能过帮助营销人员改善他们的客户群（在其目标区域内工作），并根据客户的购买历史、兴趣或活动监控来对客户类别做进一步细分。对客户进行分类有助于公司针对特定客户群制定特定的广告。

6. 参考文献。

[1] 什么是聚类分析？聚类分析方法的类别.[https://zhuanlan.zhihu.com/p/139924042]
[2] 李芳. K-Means 算法的 k 值自适应优化方法研究[D]. 安徽大学, 2015.
[3] 李荟娆. K–means 聚类方法的改进及其应用[D]. 东北农业大学, 2014