聚类实验报告

1711290 李涵 信息安全

一、问题描述

聚类问题是将N个物品，划分为K个类别，并让这种划分整体上“误差”最小。这些类不是事先设定的，而是根据数据的特征确定的。在同一类中这些对象在某种意义上趋向于彼此相似，而在不同类中对象趋向于彼此不相似。

二、解决方法

1.解决思路

采用凝聚层次聚类：首先将每个对象作为一个簇，然后合并这些原子簇为越来越大的簇，直到某个终结条件被满足。

2.基本理论

聚类

聚类算法与分类算法最大的区别是：聚类算法是无监督的学习算法，而分类算法属于监督的学习算法。在聚类算法中根据样本之间的相似性，将样本划分到不同的类别中，对于不同的相似度计算方法，会得到不同的聚类结果，常用的相似度计算方法有欧式距离法。

K-Means算法

事先确定常数K（最终的聚类类别数），首先随机选定初始点为质心，并通过计算每一个样本与质心之间的相似度(这里为欧式距离)，将样本点归到最相似的类中，接着，重新计算每个类的质心(即为类中心)，重复这样的过程，知道质心不再改变，最终就确定了每个样本所属的类别以及每个类的质心。由于每次都要计算所有的样本与每一个质心之间的相似度，故在大规模的数据集上，K-Means算法的收敛速度比较慢。

层次聚类

先计算样本之间的距离，每次将距离最近的点合并到同一个类。然后，再计算类与类之间的距离，将距离最近的类合并为一个大类。不停的合并，直到合成了一个类。其中类与类的距离的计算方法有：最短距离法，最长距离法，中间距离法，类平均法等。比如最短距离法，将类与类的距离定义为类与类之间样本的最短距离。

层次聚类算法根据层次分解的顺序分为：自下底向上和自上向下，即凝聚的层次聚类算法和分裂的层次聚类算法。自下而上法就是一开始每个个体都是一个类，然后根据linkage寻找同类，最后形成一个“类”。自上而下法就是反过来，一开始所有个体都属于一个“类”，然后根据linkage排除异己，最后每个个体都成为一个“类”。要在实际应用的时候要根据数据特点以及想要的类的个数，来考虑是自上而下更快还是自下而上更快。根据Linkage判断“类”的方法有最短距离法、最长距离法、中间距离法、类平均法等等。

基于密度的方法

k-means解决不了不规则形状的聚类。于是就有了Density-based methods来系统解决这个问题。该方法同时也对噪声数据的处理比较好。其原理简单说画圈儿，其中要定义两个参数，一个是圈儿的最大半径，一个是一个圈儿里最少应容纳几个点。只要邻近区域的密度（对象或数据点的数目）超过某个阈值，就继续聚类,最后在一个圈里的，就是一个类。DBSCAN（Density-Based Spatial Clustering of Applications with Noise）就是其中的典型

基于网络的方法

这类方法的原理就是将数据空间划分为网格单元，将数据对象集映射到网格单元中，并计算每个单元的密度。根据预设的阈值判断每个网格单元是否为高密度单元，由邻近的稠密单元组形成"类"。

基于模型的方法

为每簇假定了一个模型，寻找数据对给定模型的最佳拟合，这一类方法主要是指基于概率模型的方法和基于神经网络模型的方法，尤其以基于概率模型的方法居多。这里的概率模型主要指概率生成模型（generative Model），同一"类"的数据属于同一种概率分布，即假设数据是根据潜在的概率分布生成的。其中最典型、也最常用的方法就是高斯混合模型（GMM，Gaussian Mixture Models）。基于神经网络模型的方法主要就是指SOM（Self Organized Maps）了，也是我所知的唯一一个非监督学习的神经网络了。下图表现的就是GMM的一个demo，里面用到EM算法来做最大似然估计。

基于模糊的聚类（FCM模糊聚类）

基于模糊集理论的聚类方法，样本以一定的概率属于某个类。比较典型的有基于目标函数的模糊聚类方法、基于相似性关系和模糊关系的方法、基于模糊等价关系的传递闭包方法、基于模 糊图论的最小支撑树方法，以及基于数据集的凸分解、动态规划和难以辨别关系等方法。FCM算法是一种以隶属度来确定每个数据点属于某个聚类程度的算法。该聚类算法是传统硬聚类算法的一种改进。

衡量聚类算法优劣的标准

不同聚类算法有不同的优劣和不同的适用条件。大致上从跟数据的属性（是否序列输入、维度），算法模型的预设，模型的处理能力上看。具体如下： 1.算法的处理能力：处理大的数据集的能力（即算法复杂度）；处理数据噪声的能力；处理任意形状，包括有间隙的嵌套的数据的能力； 2.算法是否需要预设条件：是否需要预先知道聚类个数，是否需要用户给出领域知识；

3.算法的数据输入属性：算法处理的结果与数据输入的顺序是否相关，也就是说算法是否独立于数据输入顺序；算法处理有很多属性数据的能力，也就是对数据维数是否敏感，对数据的类型有无要求。

3.算法流程

层次聚类：

（1）将每个对象看作一类，计算两两之间的最小距离(三种算法分别以最近点距离、最远点距离、平均点距离作为类之间的距离)；

（2）将距离最小的两个类合并成一个新类；

（3）重新计算新类与所有类之间的距离；

（4）重复（2）（3），直到所有类最后合并成k个类

三、实验分析

1.实验数据

由函数生成实验数据

```
def create_data(centers,num=100,std=0.7):
    '''
    生成用于聚类的数据集
    :param centers: 聚类的中心点组成的数组。如果中心点是二维的，则产生的每个样本都是二维的。
    :param num: 样本数
    :param std: 每个簇中样本的标准差
    :return: 用于聚类的数据集。是一个元组，第一个元素为样本集，第二个元素为样本集的真实簇分类标记
    '''
    X, labels_true = make_blobs(n_samples=num, centers=centers, cluster_std=std)
    return  X,labels_true
```

其中centers是数据生成的中心点，可以自行设置，在实验中，最好让数据点集有交叉，以更好测试聚类算法性能

2.实验设计

建立类AverageLinkage（另外两种同理），其中定义初始化函数：

```
class AverageLinkage:

    def __init__(self, data, k):
        self.k = k
        self.data = data
        self.fit()
```

在类里定义函数merge，找到要合并的两个类（距离最小的两个类）：

```
    def merging(self):
        mini = 1e99
        merge = (None, None)

        for i in list(map(int, self.clusters.keys())):
            for j in [x for x in list(map(int, self.clusters.keys())) if x >= i+1]:

                if self.dist[i][j] < mini:
                    mini = self.dist[i][j]#距离最小的两个类进行合并
                    merge = (i, j)

        return merge
```

在类里定义fit函数，将点合成k个类：

```
 def fit(self):
        n = len(self.data)
        self.clusters = {}

        for i in range(n):
          self.clusters[i] = []
          self.clusters[i].append(i)#先把每个点各自初始化成一个类

        self.dist = np.sqrt((np.square(self.data[:,np.newaxis]-self.data).sum(axis=2)))
```

```
        self.dist_copy=self.dist.copy()#计算各个点之间的距离，并存一个副本


        for i in range(n-self.k):
            merge = self.merging()#找到要合并的两个类
            print(merge)
            self.clusters[merge[0]] = self.clusters[merge[0]] + self.clusters[merge[1]]
            self.clusters.pop(merge[1])#进行合并
            for j in range(n):
                ssum=0
                for each in self.clusters[merge[0]]:
                    ssum=ssum+self.dist_copy[j,each]
                aver=ssum/len(self.clusters[merge[0]])
                self.dist[j,merge[0]]=aver
                self.dist[merge[0],j]=aver
            #距离调整，把新的合成后的类的距离调整为各个点的距离的平均值


        for i in range(self.k):
            while not i in self.clusters:
                for j in [x for x in list(map(int, self.clusters.keys())) if x >= i+1]:
                    self.clusters[j-1] = self.clusters.pop(j)
        #最后生成k个类，将他们重新给予标签
        for i in self.clusters.keys():
            self.clusters[i].sort()
```

对于SingleLinkage和CompleteLinkage，只需要修改距离调整部分的代码

SingleLinkage

```
for j in range(n):
                if self.dist[j,merge[0]]>self.dist[j,merge[1]]:
                    self.dist[j,merge[0]]=self.dist[j,merge[1]]
                    self.dist[merge[0],j]<self.dist[merge[1],j]
```

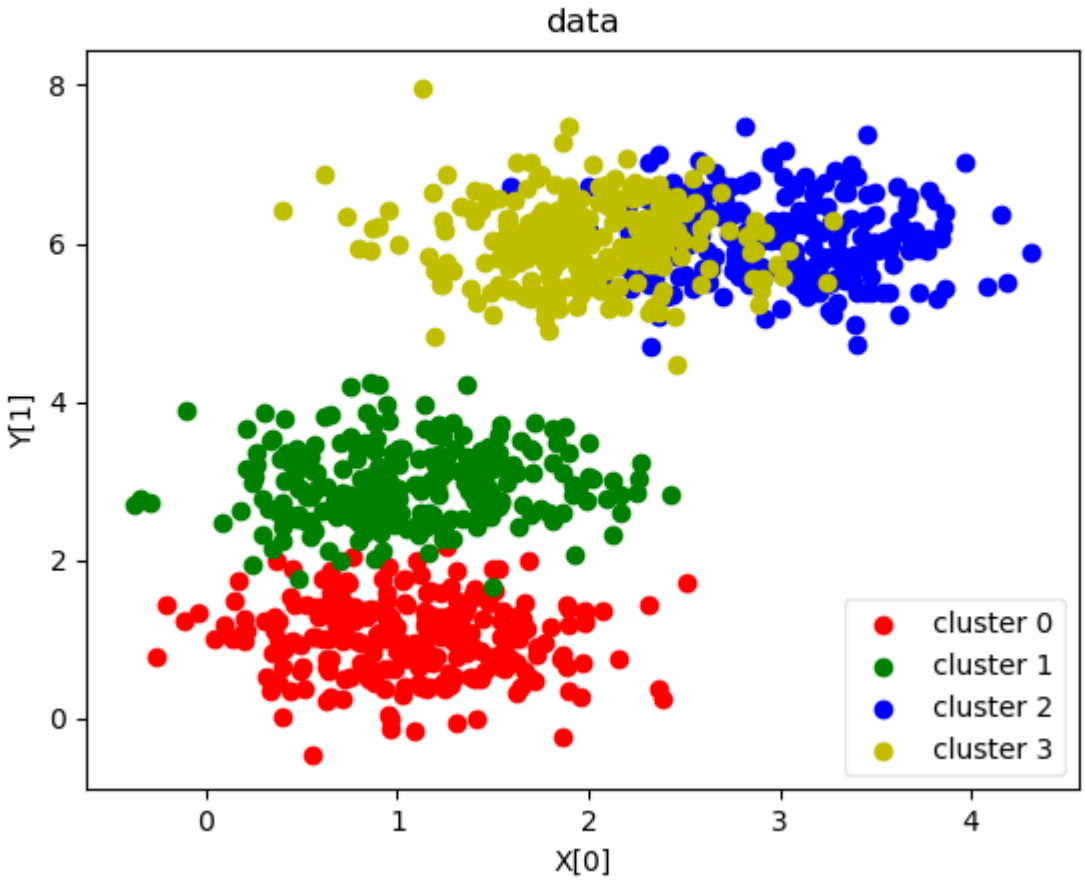
CompleteLinkage

```
for j in range(n):
                if self.dist[j,merge[0]]<self.dist[j,merge[1]]:
                    self.dist[j,merge[0]]=self.dist[j,merge[1]]
                    self.dist[merge[0],j]<self.dist[merge[1],j]
```

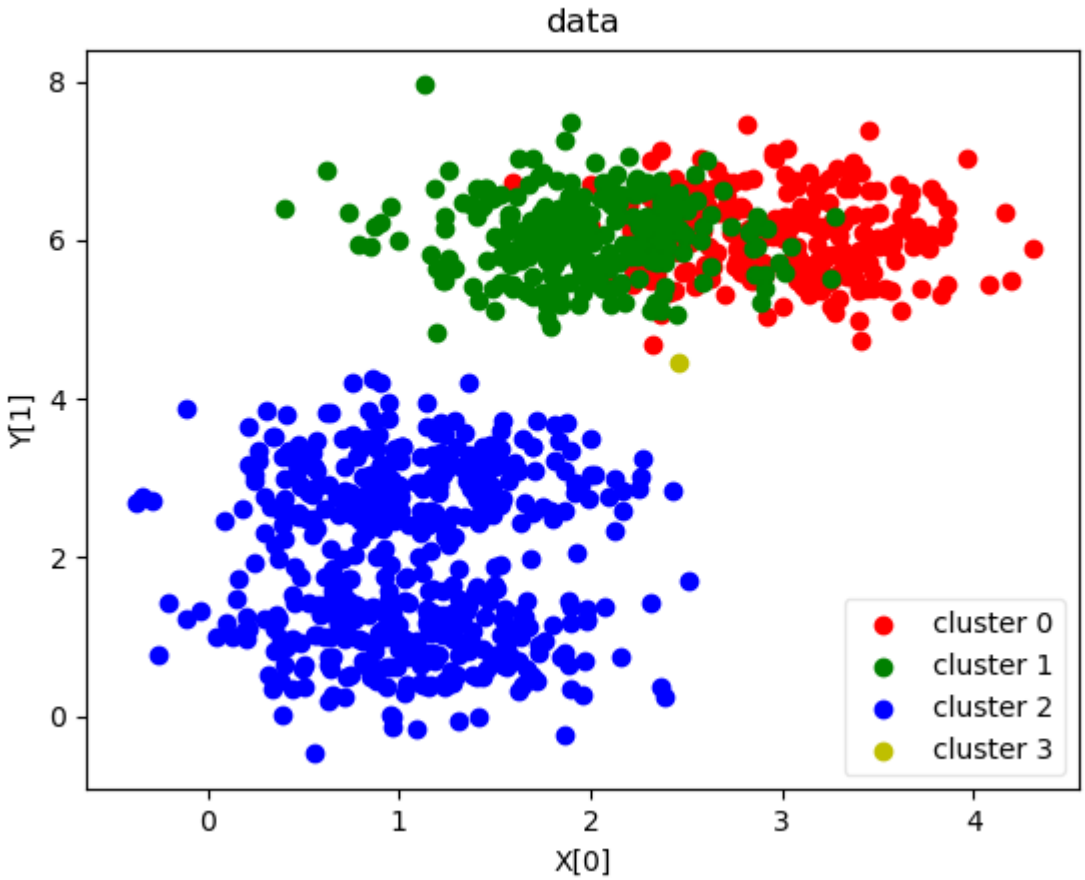
3.实验结果

测试样例1：

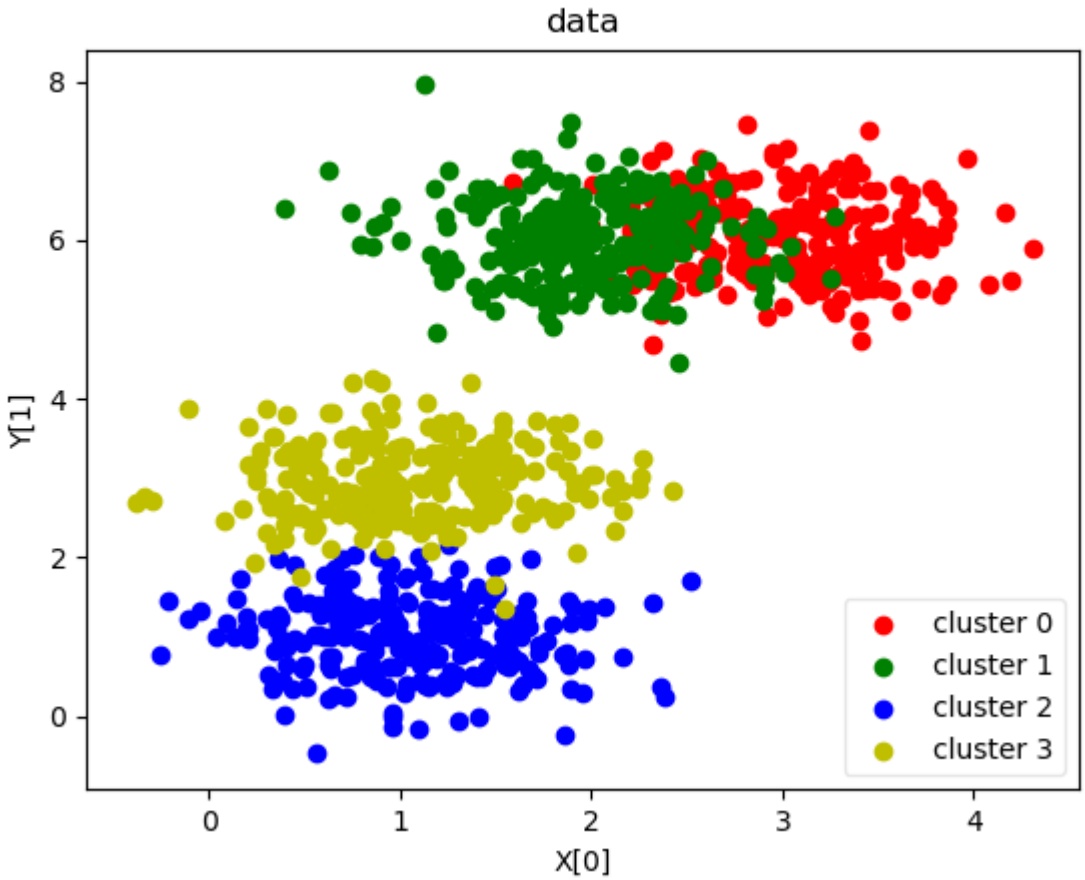
数据本身的分类结果：



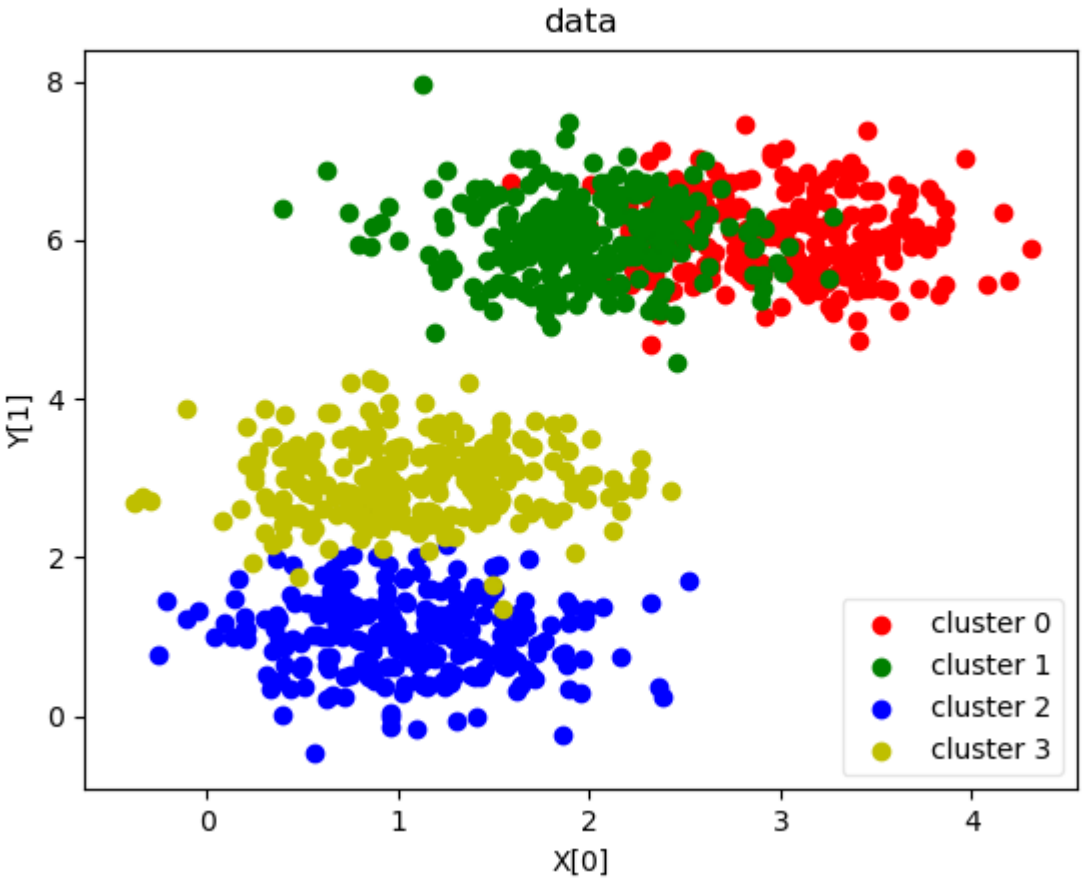
SingleLinkage：



CompleteLinkage：



AverageLinkage：



其他运行结果见运行截图的文件夹

实验结果可印证如下性质：

single linkage使用数据的相似度矩阵或距离矩阵，定义类间距离为两类之间数据的最小距离。这个方法不考虑类结构。可能产生散乱的分类，特别是在大数据集的情况下。因为它可以产生chaining现象，当两类之间出现中间点的时候，这两类很有可能会被这个方法合成一类。single linkage也可以用于分裂式聚类，用来分开最近邻距离最远的两组。

complete linkage，同样从相似度矩阵或距离矩阵出发，但定义距离为两类之间数据的最大距离。同样不考虑到类的结构。倾向于找到一些紧凑的分类。

average linkage，跟前两个方法一样，从相似度矩阵或距离矩阵出发，但定义距离为类间数据两两距离的平均值。这个方法倾向于合并差异小的两个类。（距离）介于单连锁和全连锁之间。它考虑到了类的结构，产生的分类具有相对的鲁棒性。