C#下实现的多维基础K-MEANS聚类

#算法 #机器学习#

前言

最近由于上C # 课的时候，老师提到了-我们的课程成绩由几个部分组成.分别是「最终作品展示」「小组合作聊天记录评分」「组内成员匿名互评」「报告书评分」这四项综合评价.老师希望我能够通过这四个项目对所有同学进行聚类，然后根据离每簇的中心距离来评价最终的分数.由于我没有接触过这方面的算法，所以就选了实现较为方便并且直观的聚类方法K-MEANS.所以下文中就会对我这次学习到的一些心得进行分享.由于是C # 课程，所以本次的算法将以C# 为例子进行介绍.

聚类&K-Means

聚类

百科上对于聚类的解释是这样的:

> 将物理或抽象对象的集合分成由类似的对象组成的多个类的过程被称为聚类.由聚类所生成的簇是一组数据对象的集合，这些对象与同一个簇中的对象彼此相似，与其他簇中的对象相异.

简单的来说聚类就是将相似的归在一起产生一个簇，使不相似的分在不同的簇里.

聚类和分类的区别

分类:从特定的数据中根据人为打上的数据表浅进行数据挖掘，做出判断.是将类别已知.并且样本数据已经做了标记的数据进行归类.是一个有监督的过程.

聚类:目的是分类数据,但是在分类结束前我们都不知道数据是怎么分类的，有什么特点，只是根据算法，自动的将相似性的分类在了一起.数据本身没有标记.本身也没有类别.通过聚类的算法将相似性高的数据通过算法打上的标签归类在一起.是一种无监督的过程.

K-Means算法

K-Means算法是基于距离(我在这次中使用了欧几里德距离)的聚类算法 ，采用距离或者特征向量作为相似程度的考量，数据之间的距离 / 向量余弦越接近, 其相似度就越大.在K-Means聚类算法中-簇是由距离较为相近的数据对象构成的,故用K-Means算法的目的是想要得到数据对象相对紧凑且独立的不同簇.

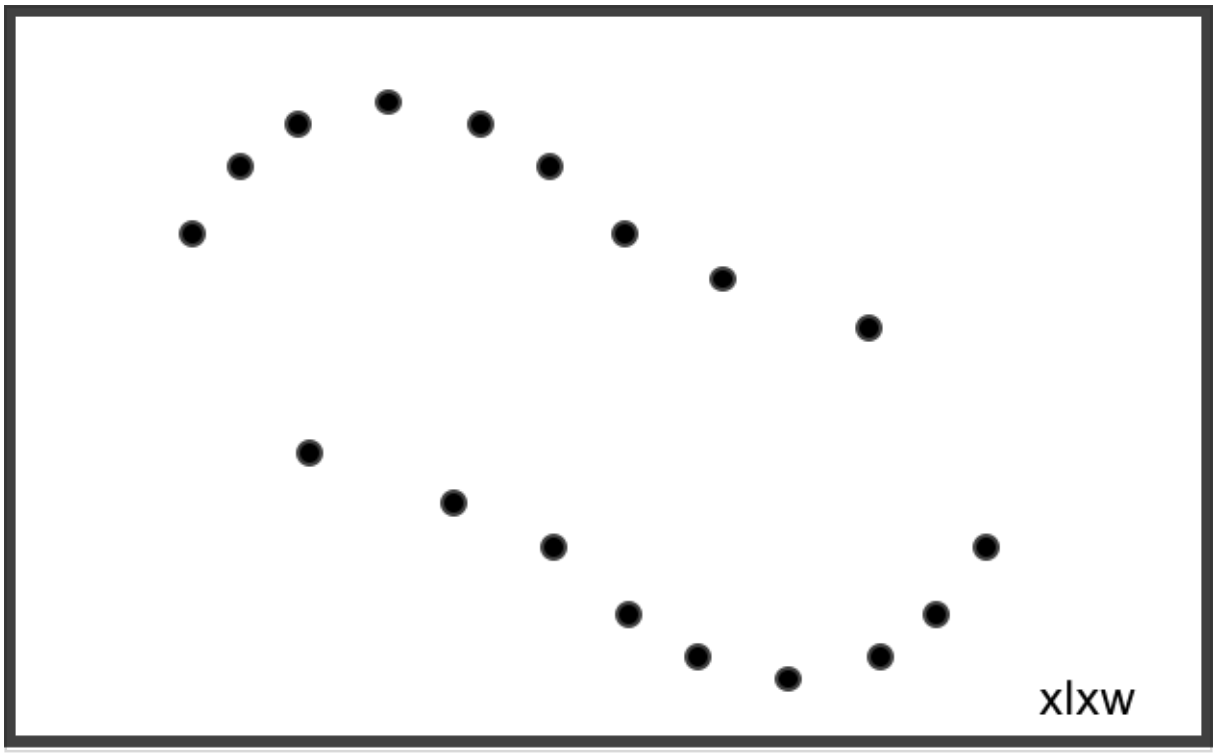
「评分系统」和K-Means的结合

这次老师让我做的就是希望我能通过给出的「最终作品展示」「小组合作聊天记录评分」「组内成员匿名互评」「报告书评分」这四个维度的数据对所有同学的成绩使用K-Means聚成四个类，分别对应90-100/80-90/70-80/60-70/不及格这五个等第.分出登第后，五个类的中心分别是95/85/75/65/55，根据离数据中心的欧几里得距离按比例来决定最终的成绩.这就是老师让我实现的功能.

K-Means的非适用性

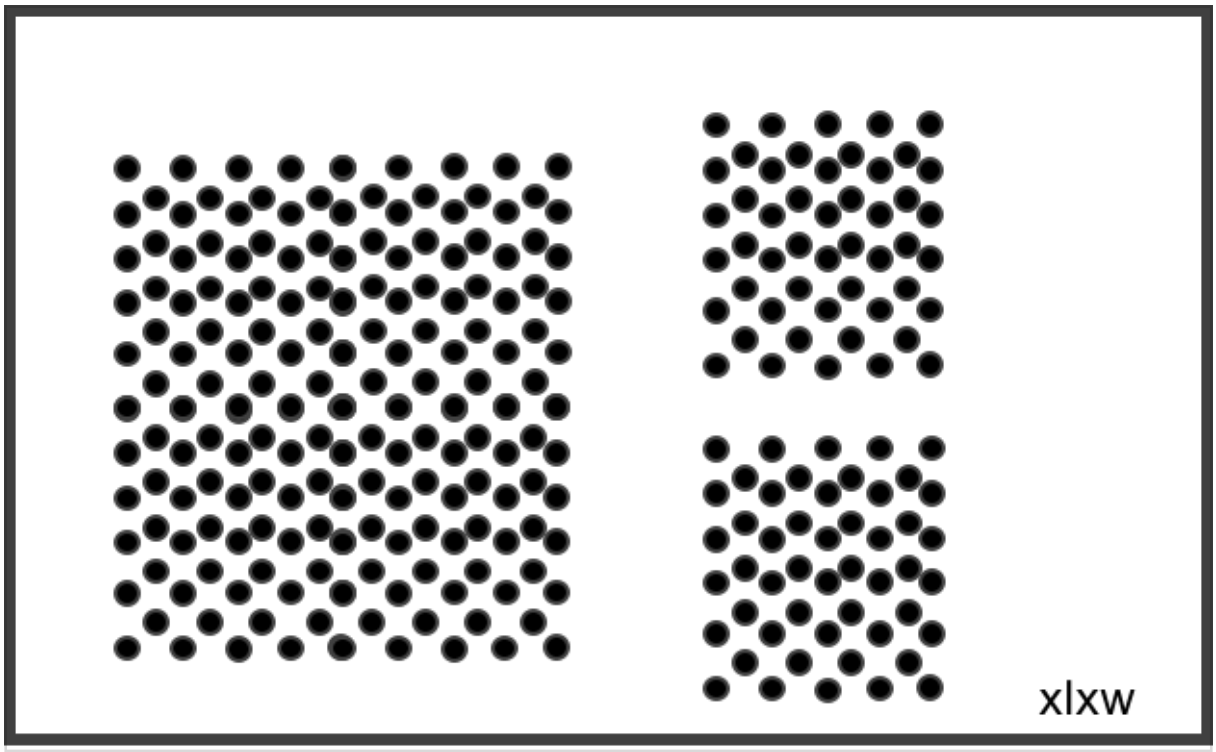
K-Means算法适应于连续形式的数据，由于数据的分布不同，有时候就无法分得准确的类，比如下面的两种形式

1. 非标准正态分布

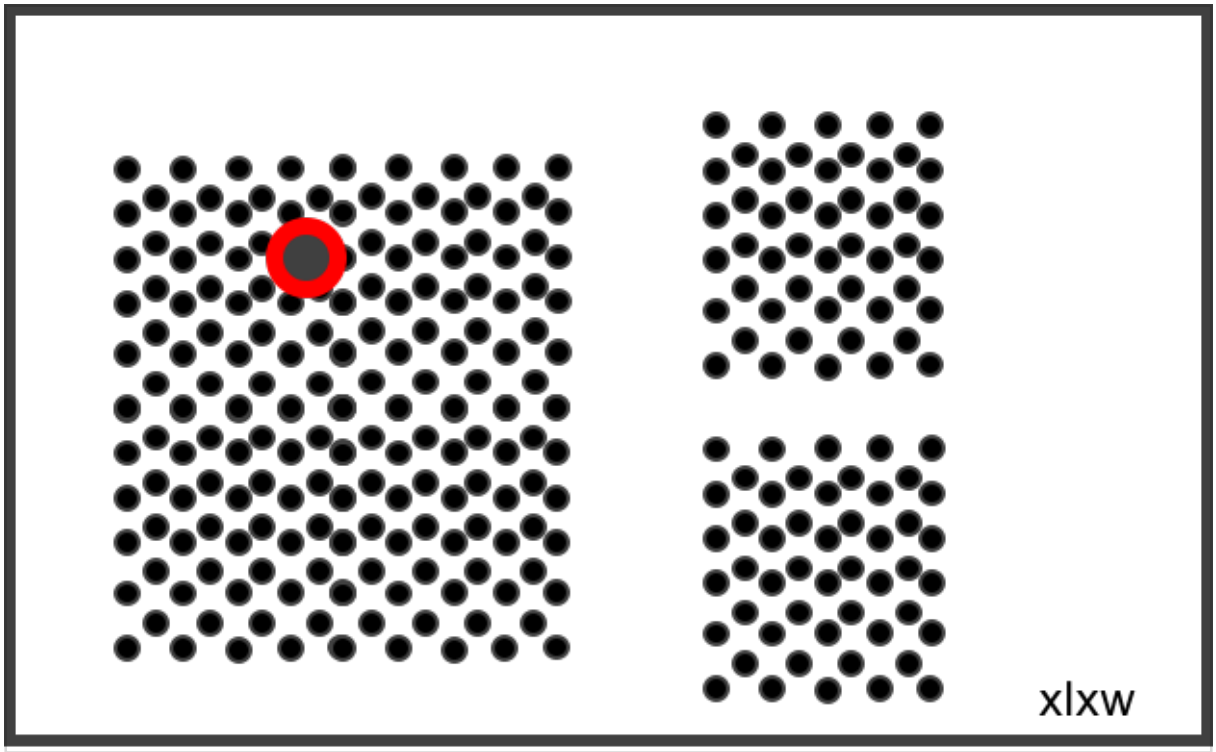


K-Means用于表示聚类趋势只是说在数据符合正态分布或偏态不太严重时才是合理的.如果偏态严重，或者有异常的极大极小值，对均数影响很大，这时K
-MEANS不能代表聚类的趋势.
故在实际的样本中，我们作出的假设可能会并不适用，这个时候用基础的K-Means算法就不是特别的恰当了.这个时候我们就应该用到一些改进的算法,比如 Kernel K-means和Spectral Clustering.这两种算法会在后面的文章中进行介绍.

2. 非均匀分布



由于分布的不均匀，疏密程度就会对中心的选取造成影响.就会偏向左侧较密的地方.如下所示:



K-Means的优点和缺点

优点:

1. 算法简单并且效率很高，迭代次数.
2. K-Means聚类算法的时间复杂度是O(nkt) {n数据量 / k簇的个数 / t算法迭代次数}故时间复杂度近似为线性.

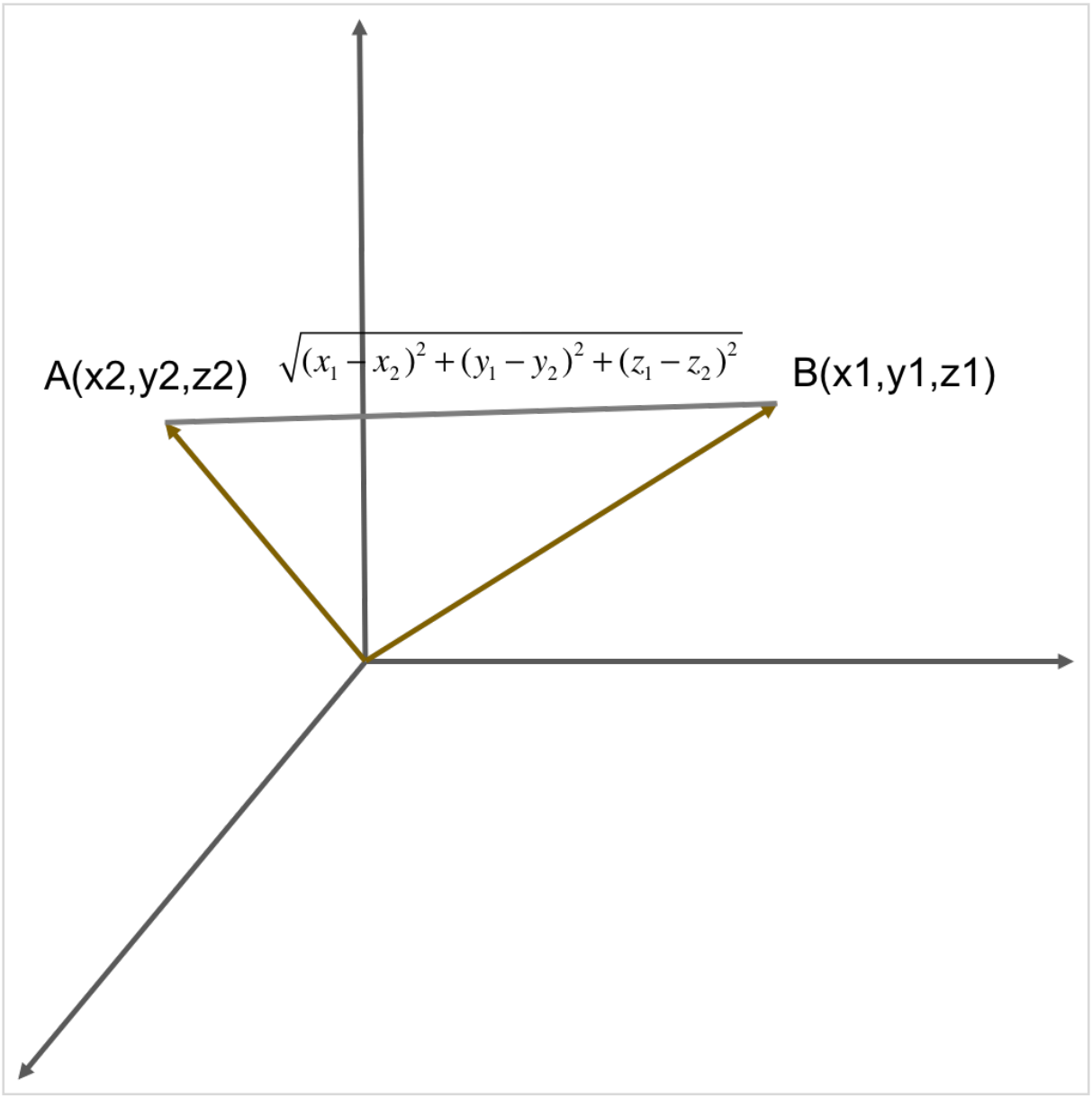
3. 自身具有优化迭代的特点，通过不断的计算中心来修正聚类结果.
4. 对于数据有着很好的伸缩性.

缺点:

1. 需要指定K的值　我们在每次聚类之前需要指定数据会被分成几类即要有几个中心点.
2. 对于离群值较为敏感　由于在每次聚类之后都要重新生成簇的中心.而中心是根据所有簇种数据对象的"质心"来的，所以当数据集不大的时候，这些离群的特殊点对中心就会产生很大的影响.这可以通过对数据集进行预处理，筛出离群的点.离群的点可以全部归类为第(k+1)类,因为它们本身有着非常独特的性质，是我们分析时候可以研究的对象.
3. 初始点的选取对结果会产生影响 初始的点我们一般通过随机选定,但是随机选出的点可能点和点之间并不合理，会造成最终聚类的结果会不相同.
4. 最终的聚类结果是一个球形的簇　由于使用了欧几里得距离，最终一定是以簇中心为球心(圆心)的一个球形(圆形)的簇.
5. 维度对聚类结果的贡献无法得到　无法确定哪一个维度对聚类结果的产生的影响比较大.

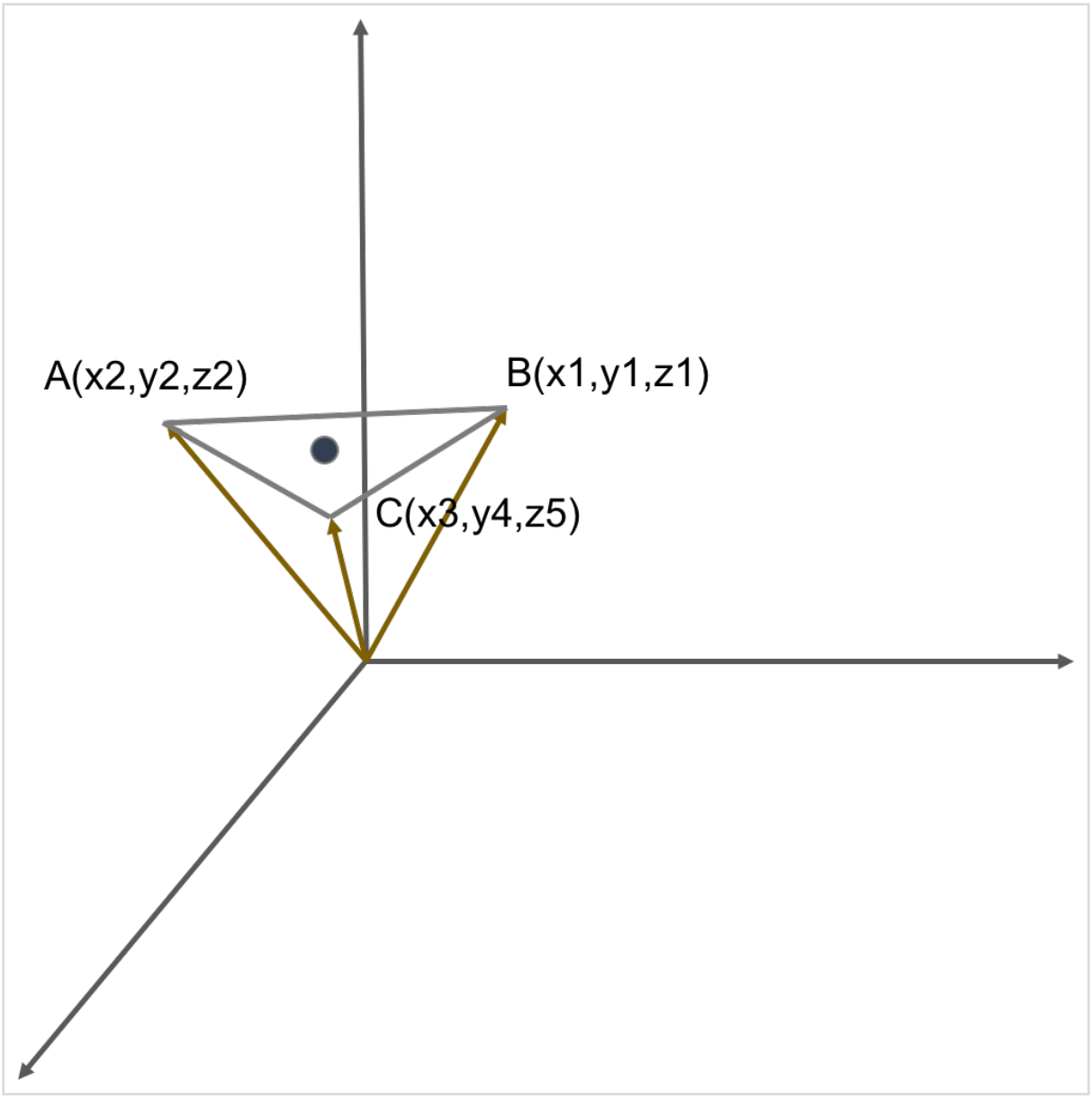
K-Means的一些细节

- 欧几里德距离
  欧几里德距离简称欧式距离，是我们平时经常用到的一个距离度量公式，具体就是基础求距离公式,并且拓展至了n维空间.我们以三维空间举一个例子，如下图所示.A,B点根据欧几里得距离公式算出来的就是中间灰色线上方的距离公式



- “质心”公式计算

  “质心”的选取是因为每一次聚类后都需要通过簇中的数据对象生成新的中心，而生成新中心的方式就是产生所有数据对象的”质心”.公式也可以拓展至n维,这里我们以3维的空间为例子



A / B / C三个点的"质心"就是图中的灰点，坐标为((X1+X2+X3)/3),(Y1+Y2+Y3)/3,(Z1+Z2+Z3)/3)拓展到n维就是((X1+…+Xn) / n,…,((Z1+…Zn) / n)).

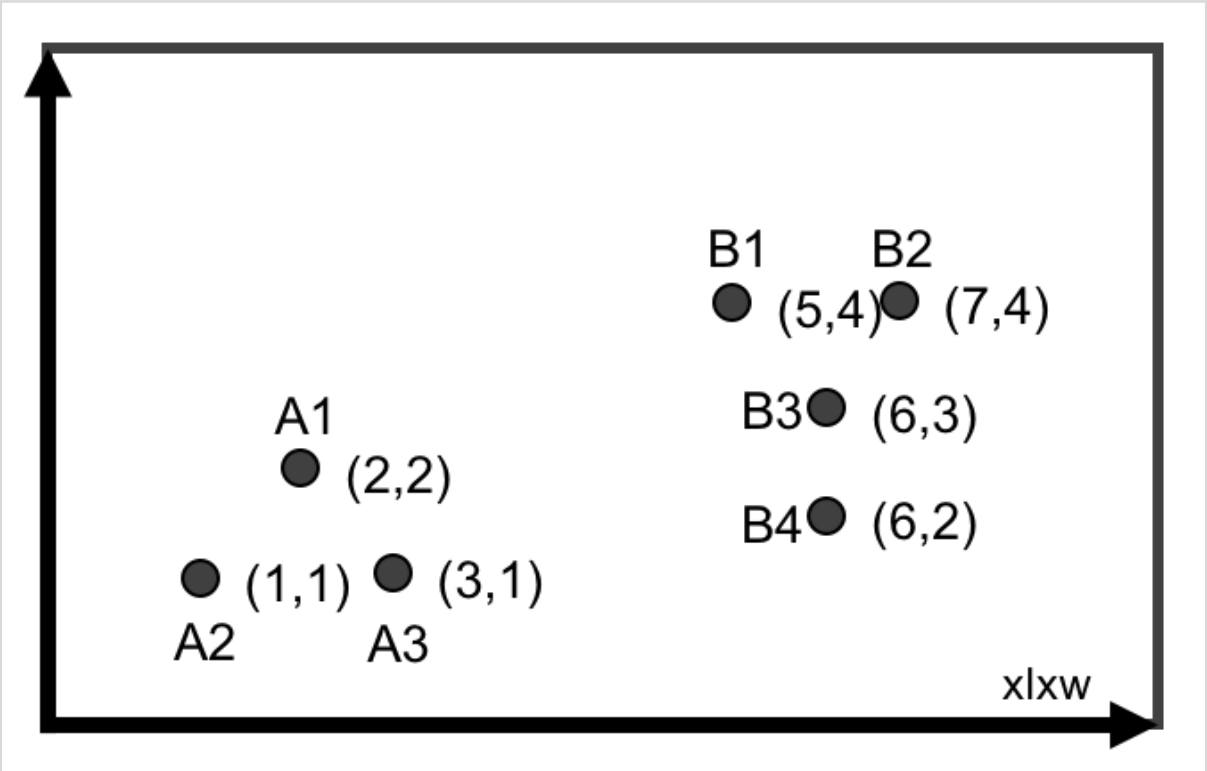
K-Means算法实现步骤

这里我们介绍的是K-Means算法的本身步骤,没有加上任何的优化方式，总共分为 5 步，分别如下:

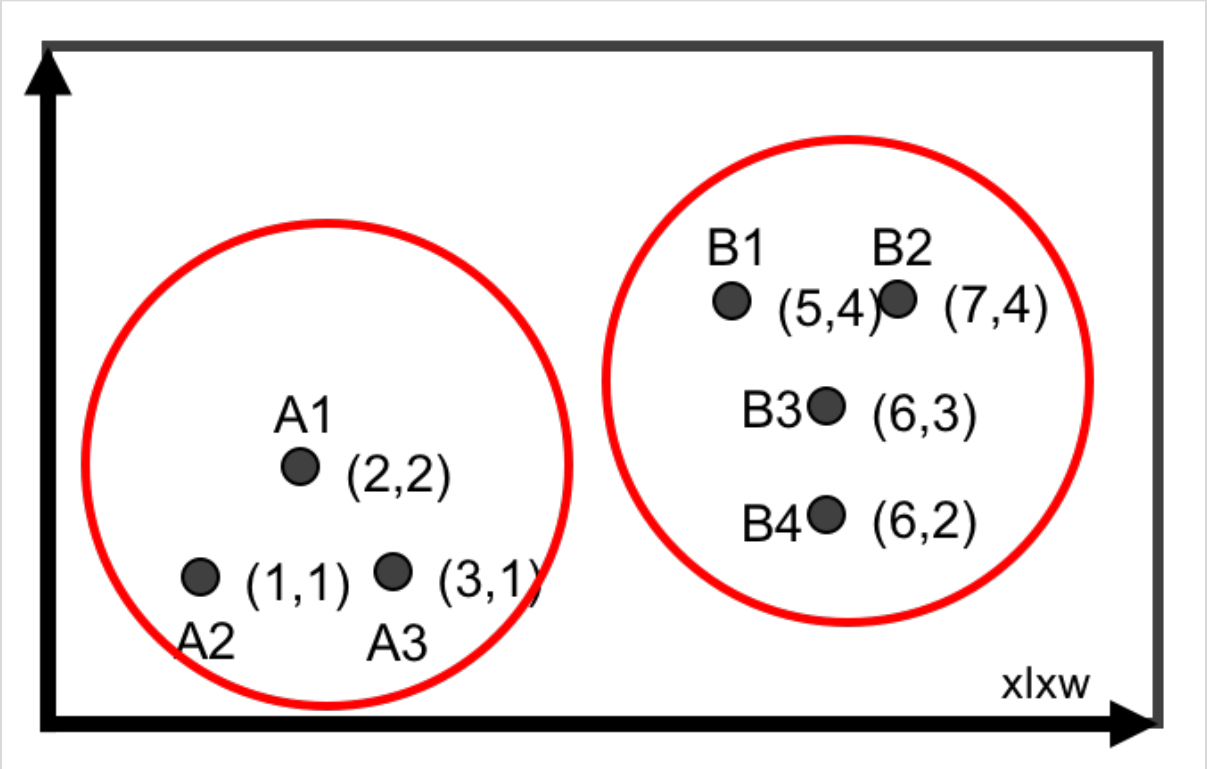
1. 首先，就像上文中说的一样.K-Means算法必须提供k即簇的树目.使我们能够通过聚类算法得到k个分组
2. 从我们要进行聚类的数据集中随机选择k个点作为每个簇的初始中心
3. 通过一定的计算方法(欧几里德距离)来计算每个数据对象距离每一个簇中心的距离.并且得到距离最近的簇的中心点，那么这个数据对象就被归类到了这个簇中.
4. 当所有的点都被聚类后,对于每一个簇算出新的中心.(通过算法找到"质心").
5. 迭代3，4两个步骤，直到4步骤选出来的新中心和原来的中心小于某个我们默认的阈值(几乎中

心不再发生变化),算法停止.我们已经通过聚类算法得到了我们最终的结果.

接下来我们通过举例子的方法来更好的理解一下K-Means算法，我们随机生成了七个点,它们的位置即坐标如下图所示

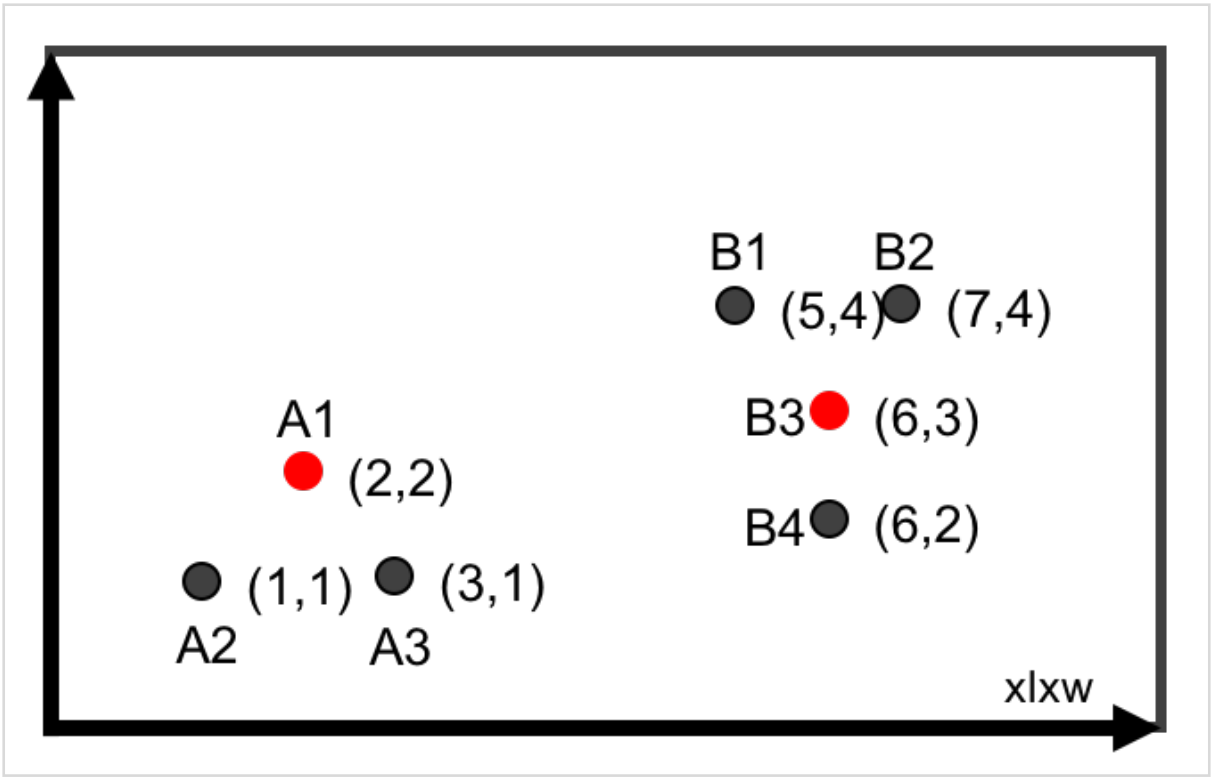


很明显,我们如果有监督的将他们分类，那么结果一定是如下图所示的情况



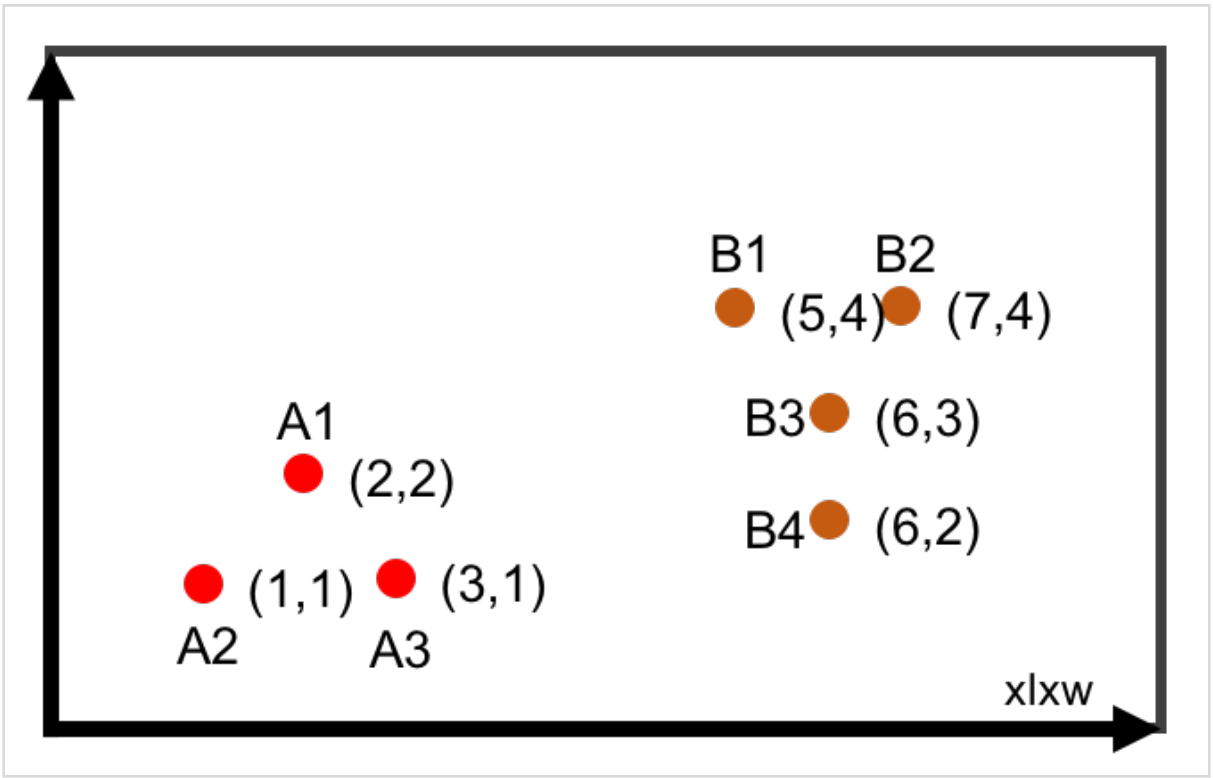
那么我们按照K-Means算法的步骤，看一下聚类的过程是怎么样的.

1.首先我们先随机选择两个点作为簇的中心，就选A1/B3这两个点



2.计算么个数据对象距离各个簇中心的距离

对于A2 - 距A1的距离就是((1-2)^2 + (1-2)^2)^(1/2) = √2 / 距A2的距离就是((1-6)^2+(1-3)^2)^(1/2) = √29 由于√29>√2，所以A2被归类为以A1为中心的簇，同理，其他的点经过归类后的结果如下图所示:



3.重新计算每个簇的中心

我们现在得到了两个簇,现在我们把每个簇里的数据对象的中心求出来,根据质心的求法，我们可以知道红色A组的新中心是((2+1+3)/3,(2+1+1)/3) = (2,4/3)，棕色B组的新中心是((5+6+6+7)/4,(4+4+3+2)/4) = (6,3)这个时候我们发现B的中心就是B3这个点，所以对于B组

而言不需要再次进行聚类了.
对于A组我们可以发现，A1，A2，A3距离新的中心(2,4/3)相等,故再进行计算新中心的话中心也不会发生改变,所以聚类停止.
4.最终的聚类结果就是A1/A2/A3一组.B1/B2/B3/B4一组.

K-Means在C#下算法的实现

(在这里我展示的是没有优化过的K-Means实现)

1.首先是初始化中心.

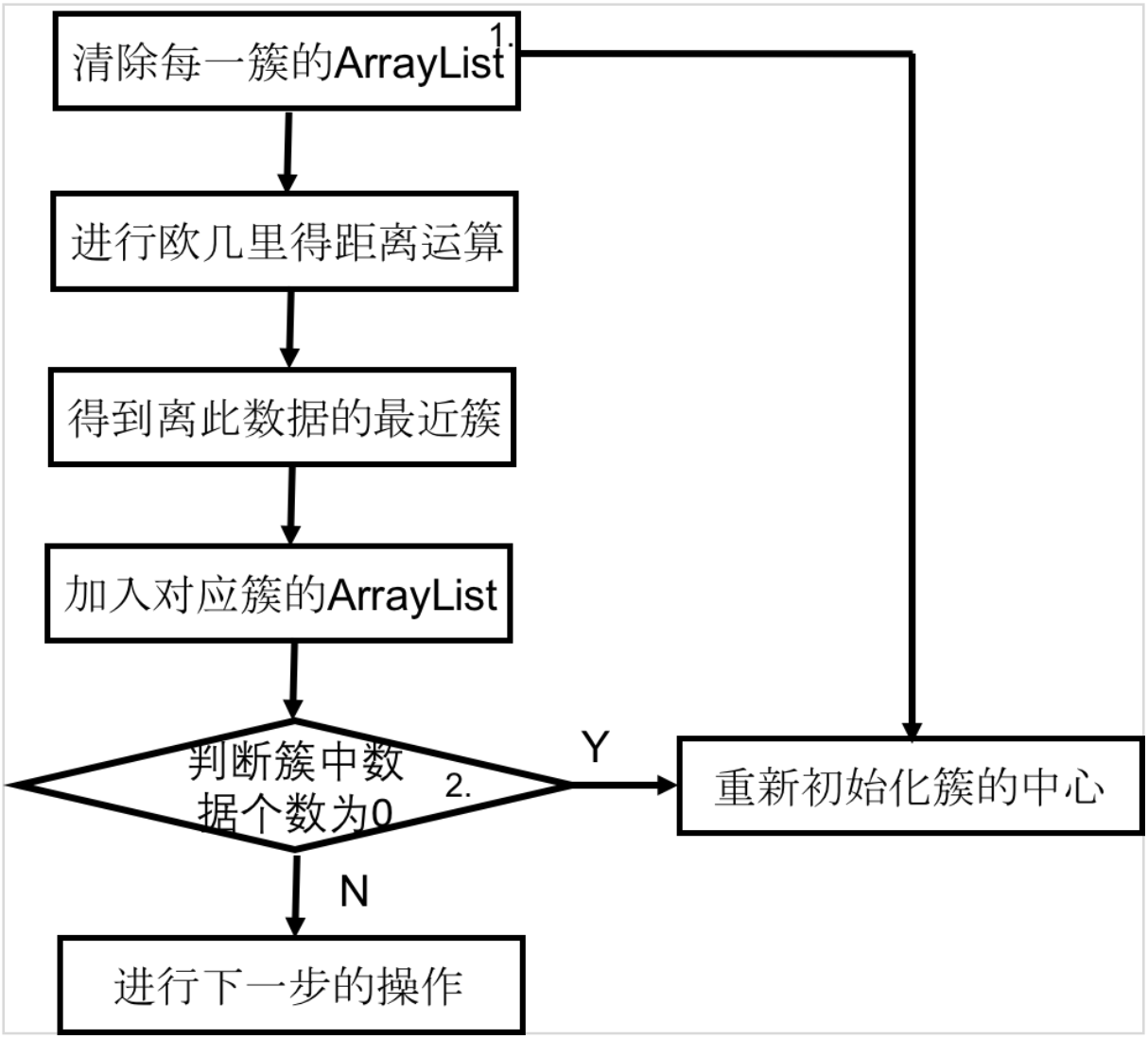
根据我们设定的簇的数量,我们可以随机从所有的数据对象中选择k个点作为初始的中心点.这里只需要使用Random类的Next方法即可,所以此步骤不进行介绍.

2.聚类步骤的实现

代码特点解释:

1. 我的程序是先将数据从Excel的Xls文件中导出到了C#控件中的ListView里，这个控件的名字是:XlsDataSh,在程序的代码里可以看到这个控件的名字.
2. 由于我们的Xls文件中并不是只有这四个项的数据，其中的其他几项是另外的一些信息，有着编号 / 姓名 / 学号姓息分别占了ListView的0/1/2这三列，而3-6这四个列分别是对应四个维度的.这就是为什么循环中经常会出现类似for (int k = 3; k < 7; k++)的原意.

我们先来看一下聚类这个方法中的运行步骤，如下所示:



解释:

1. ArrayList中存储的事每一个簇所拥有的数据的编号.所以当我们每一次重新计算出了簇的中心之后，会对数据所在的簇进行重新聚类，故首先我们要把ArrayList中所存储的数据进行清除.
2. 判断簇为0是一个重要的步骤.这也是我"偷懒"了之后用的方法，由于我们的初始中心是随意选择的，所以很有可能选择的数据中心再一次聚类之后没有分得任何的数据点，这时候这个类就是无效的.我们应该对它进行处理，而我用的方法就是偷懒的让它再次随机选定中心，直到每个类都一定能分到其他的数据点.
3. ClassNum - 设定的簇的数目.即k
4. RowCount - 数据对象的个数.即n

下面就是聚类方法中的代码:

```
private void Cluster()
        {
            int tmpclass = 0;
            double tmpClusDis = 0, tmpClusMinDis = 0;
            //清除每一个簇里原来的项
```

```
            for (int i = 0; i < ClassNum; i++)
            {
                ClusterAssem[i].Clear();
            }

            //进行欧几里德距离计算并聚类
            for (int i = 0; i < RowCount - 1; i++)
            {
                tmpClusMinDis = vurMax;
                for (int j = 0; j < ClassNum; j++)
                {
                    tmpClusDis = 0;
                    //这里是取出每个维度的数据进行加和
                    for (int k = 3; k < 7; k++)
                    {
                        tmpClusDis +=
Math.Pow((System.Convert.ToDouble(XlsDataSh.Items[i].SubItems[k].Text) -
CenterArrayParams[j, k - 3]), 2);
                    }
                    if (tmpClusDis < tmpClusMinDis)
                    {
                        tmpclass = j;
                        tmpClusMinDis = tmpClusDis;
                    }
                }
                ClusterAssem[tmpclass].Add(i);
            }
            //重新初始化
            if (ClusterAssem[0].Count == 0 || ClusterAssem[1].Count == 0 ||
ClusterAssem[2].Count == 0 || ClusterAssem[3].Count == 0 ||
ClusterAssem[4].Count == 0)
            {
                InitCenter();//重新初始化中心的方法.
                Cluster();
            }
        }
```

3.重新生成每个簇的中心

解释:

ClusterAssem - 存储每一个簇中包含的数据对象ArrayList

RenewCenterArrayParams - 存储每个簇中心的各个维度的ArrayList

```
private void RenewCenter() {
            double tmpSameDis = 0;

            for (int i = 0; i < ClassNum; i++) {
                for (int k = 3; k < 7; k++)
                {
                    tmpSameDis = 0;
                    //遍历每个簇的点，求出中心点
                    foreach (object n in ClusterAssem[i]) {
                        tmpSameDis +=
System.Convert.ToDouble(XlsDataSh.Items[System.Convert.ToInt16(n)].SubItems[k].
Text);
                    }
                    RenewCenterArrayParams[i, k - 3] = (tmpSameDis * 1.0 /
(ClusterAssem[i].Count+1));
                }
            }
        }
```

4.计算结束的标记

结束标志的意图就是当中心不再发生变化或小于一个阈值的时候就停止算法的进行了.

解释:

1. BalEndFlag:用于存储前一次的中心加和用于和本次进行比对

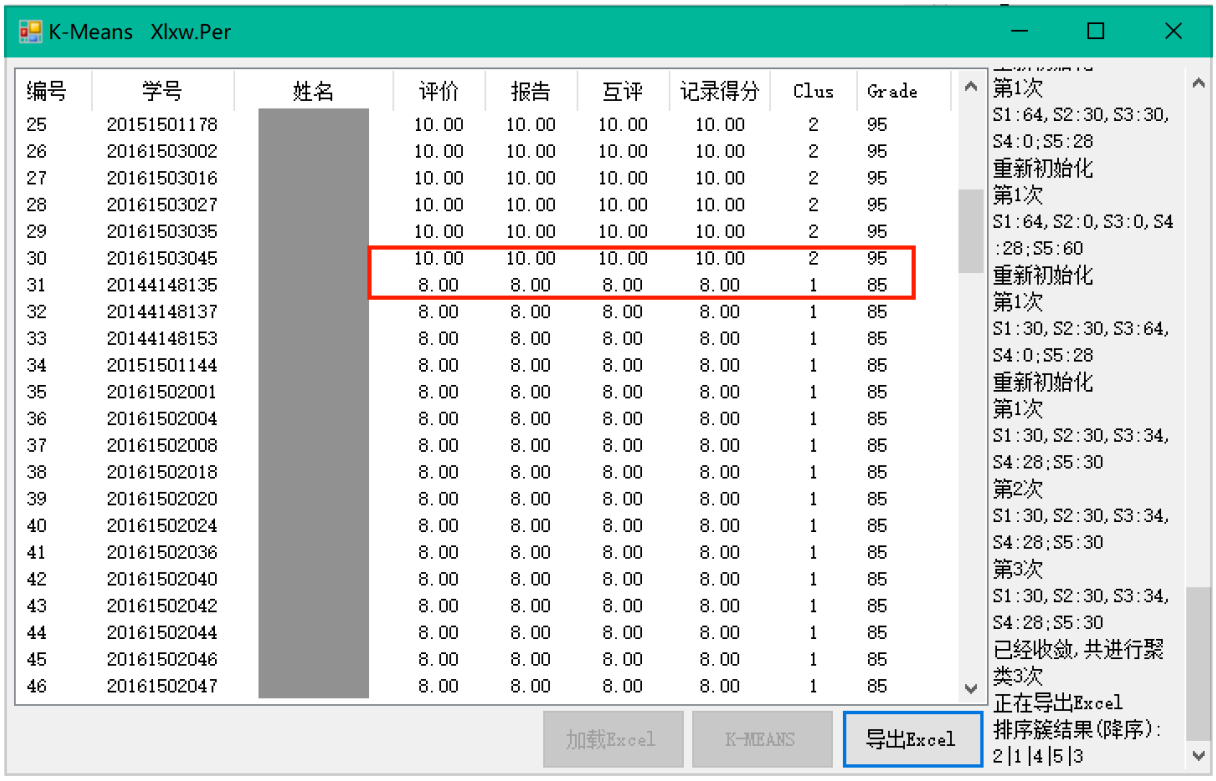
```
private Boolean CalEndFlag() {
            double tmpDifferDis = 0, tmpSameDis = 0;

            for (int i = 0; i < ClassNum; i++) {
                tmpSameDis = 0;
                for (int j = 0; j < 4; j++) {
                    tmpSameDis += Math.Pow((RenewCenterArrayParams[i, j] -
CenterArrayParams[i, j]),2);
                }
                tmpDifferDis += Math.Pow(tmpSameDis,1.0/2);
            }
            //判断中心点和前一次的偏移
```

```
            if ((BalEndFlag - tmpDifferDis) <= EndFlag) return false;
            else {
               //算法没有结束，所以将新的中心赋给用于计算的ArrayList
                for (int i = 0; i < ClassNum; i++) {
                    for (int j = 0; j < 4; j++) {
                        CenterArrayParams[i, j] = RenewCenterArrayParams[i, j];
                    }
                }
                BalEndFlag = tmpDifferDis;
                tmpDifferDis = 0;
                return true;
            }
        }
```

结果展示:

下面就是我做的基于K-Means的自动聚类评分系统的界面.首先我们先看看有监督下的聚类是否符合预期,如下图所示:



我们通过上图的红色框中可以看出这个聚类的结果和我们的预期是十分的符合的.下面就是我通过随机的生成方法生成的数据,来观察K-Means算法在无监督下的运行结果.如下图所示:

以上就是本次探究的最终成果展示.

K–Means的收敛性

K-Means算法一定是收敛的，否则就会陷入一直寻找新的"质心"而没有最终的结果了.具体的证明步骤较为复杂.通过参考一些文章(会在下文中参考中标出)涉及到了E-M算法.将一个聚类问题转化为一个最大似然估计问题可证明K-Means是E-M算法的一个特例.在E-M算法下，求得的参数一定是收敛的.在K-Means算法中目标是使损失函数最小，所以在E-step时，找到一个最逼近目标的函数,在M-step时，固定这个函数(数据对象的分配)，更新均值(簇的中心),所以最后一定会收敛了.

基础K-Means算法的优化

由于K-Means是非常成熟的算法，所以有很多先辈们改良了这个算法,在这里仅仅只是介绍在各个方面有哪些能够参考改进的方法.

- 通过上文，我们可以知道.K-Means算法容易受到离群值的干扰.所以我们可以通过预处理比如「LOF算法」来先检测出离群点.单独处理这部分点.
- K-Means的k值在没有指定的时候是很难选择的,这个时候,可以通过「k-Means算法的k值自适应优化方法」
- 聚类的中心的选择.由于本文是随机选择初始的k个点，但是这是不合理的,应该是选择数据中距离尽可能远的K个点,这里也有一个算法「Canopy算法」
- 现在有许多改进了的K-Means的算法,比较出名的有「K-means++」「ISODATA」「Kernel K-means」等,在后面的文章中我会进行分享.

参考

1.(传送门)K-Means算法的收敛性和如何快速收敛超大的KMeans？ - 大数据躺过的坑 - 博客园
2.「请问如何用数学方法证明K-means是EM算法的特例？」(传送门)https://www.zhihu.com/question/49972233?sort=created