**K-Means介绍**

  1967年，James MacQueen提出“K-Means”（K均值），这个被应用得最广泛的基于划分的聚类算法，其实是一种硬聚类算法，属于典型的局域原型的目标函数聚类的代表。算法首先随机选择k个对象，每个对象初始地代表一个簇的平均值或者中心。对于剩余的每个对象，根据其到各个簇中心的距离，把它们分给距离最小的簇中心，然后重新计算每个簇的平均值。重复这个过程，直到聚类准则函数收敛。

该算法是很典型的基于距离的聚类算法，采用距离作为相似性的评价指标，即认为两个对象的距离越近，其相似度就越大。该算法认为簇是由距离靠近的对象组成的，因此把得到紧凑且独立的簇作为最终目标。

k个初始类聚类中心点的选取对聚类结果具有较大的影响，因为在该算法第一步中是随机的选取任意k个对象作为初始聚类的中心，初始地代表一个簇。该算法在每次迭代中对数据集中剩余的每个对象，根据其与各个簇中心的距离将每个对象重新赋给最近的簇。当考察完所有数据对象后，一次迭代运算完成，新的聚类中心被计算出来。如果在一次迭代前后，J的值没有发生变化，说明算法已经收敛。

**k-means算法步骤**

（1） 从 n个数据对象任意选择 k 个对象作为初始聚类中心；

（2） 根据每个聚类对象的均值（中心对象），计算每个对象与这些中心对象的距离；并根据最小距离重新对相应对象进行划分；

（3） 重新计算每个（有变化）聚类的均值（中心对象）；

（4） 计算标准测度函数，当满足一定条件，如函数收敛时，则算法终止；如果条件不满足则回到步骤（2）。

### 算法伪代码

k-means算法：

for k = 1, … , K 令 r(k) 为从D中随机选取的一个点；

while 在聚类Ck中有变化发生 do 形成聚类：

For k = 1, … , K do

Ck = { x ∈ D | d(rk,x) <= d(rj,x) 对所有j=1, … , K, j != k}；

End;

计算新聚类中心：

For k = 1, … , K do

Rk = Ck 内点的均值向量

End;

End;

**k-means优点**

优点：本算法确定的K 个划分到达平方误差最小。当聚类是密集的，且类与类之间区别明显时，效果较好。对于处理大数据集，这个算法是相对可伸缩和高效的，计算的复杂度为O(NKt)，其中N是数据对象的数目，t是迭代的次数。一般来说，K<<N，t<<N 。

**k-means缺点**

① 在 K-means 算法中 K 是事先给定的，这个 K 值的选定是非常难以估计的。很多时候，事先并不知道给定的数据集应该分成多少个类别才最合适。这也是 K-means 算法的一个不足。

② 在 K-means 算法中，首先需要根据初始聚类中心来确定一个初始划分，然后对初始划分进行优化。这个初始聚类中心的选择对聚类结果有较大的影响[26-29]，一旦初始值选择的不好，可能无法得到有效的聚类结果，这也成为 K-means算法的一个主要问题。

③ 从 K-means 算法框架可以看出，该算法需要不断地进行样本分类调整，不断地计算调整后的新的聚类中心，因此当数据量非常大时，算法的时间开销是非常大的。所以需要对算法的时间复杂度进行分析、改进，提高算法应用范围。

**K-means++算法介绍**

kmeans++算法的主要工作体现在种子点的选择上，基本原则是使得各个种子点之间的距离尽可能的大，但是又得排除噪声的影响。

k-means++算法基本步骤

1、从输入的数据点集合（要求有k个聚类）中随机选择一个点作为第一个聚类中心

2、对于数据集中的每一个点x，计算它与最近聚类中心(指已选择的聚类中心)的距离D(x)

3、选择一个新的数据点作为新的聚类中心，选择的原则是：D(x)较大的点，被选取作为聚类中心的概率较大

4、重复2和3直到k个聚类中心被选出来

5、利用这k个初始的聚类中心来运行标准的k-means算法

算法的关键是第3步，如何将D(x)反映到点被选择的概率上，一种算法如下：

先从我们的数据库随机挑个随机点当“种子点”

对于每个点，我们都计算其和最近的一个“种子点”的距离D(x)并保存在一个数组里，然后把这些距离加起来得到Sum(D(x))。

然后，再取一个随机值，用权重的方式来取计算下一个“种子点”。这个算法的实现是，先取一个能落在Sum(D(x))中的随机值Random，然后用Random -= D(x)，直到其<=0，此时的点就是下一个“种子点”。

重复2和3直到k个聚类中心被选出来

利用这k个初始的聚类中心来运行标准的k-means算法

k-means++优点

k-means需要人为地确定初始聚类中心，不同的初始聚类中心可能导致完全不同的聚类结果。

针对这个缺陷，可以使用k-means++算法来解决。

改进之后的算法

#include<iostream>

#include<cmath>

#include<iomanip>

#include<ctime>

using namespace std;

int main()

{

int i,j,k;

int kpoint=0;//记录有多少个k点

cout<<"请输入要聚类多少块"<<endl;

cin>>kpoint;

float wei[100][100]={0}; //wei是用来存储聚类的点。

int wei1,wei2; //wei1是点的总数,wei2是一个点有几个维度

cout<<"请输入要聚类点的总个数wei1"<<ends;

cout<<"和每个点的维度wei2"<<endl;

cin>>wei1>>wei2;

srand(time(0));

cout<<"请输入"<<wei1<<"个点的值"<<endl; //cout.width

for(i=1;i<=wei1;++i)

{

// cout<<"这是输入的第"<<setw(3)<<i <<"个点";

for(j=1;j<=wei2;++j)

{

wei[i][j]=rand()%100;

// cin>>wei[i][j];

}

}

for(i=1;i<=wei1;++i)

{

cout<<"这是输入的第"<<setw(3)<<i <<"个点";

for(j=1;j<=wei2;++j)

{

cout<< wei[i][j]<<ends;

//if(j==10)

//cout<<endl;

// cin>>wei[i][j];

}

cout<<endl;

}

cout<<"请输入聚类"<<kpoint<<"个点的值"<<endl;

float kpointkey[kpoint+1][wei2+1]; //所有下标从1开始

for(i=1;i<=kpoint;++i)

{

cout<<"这是输入的第"<<setw(3)<<i <<"个点";

for(j=1;j<=wei2;++j)

{

cin>>kpointkey[i][j];

}

}

cout<<endl;

int storebelongkmun[kpoint+1][wei2+1]={0};//存储每个k点分别聚了那些点；

int contraststorebelongkmun[kpoint+1][wei2+1]={0}; //用来修改flag的值，确保每个K点处的点不变化聚类结束

//应该要再修改下

int roundcount=1;

int flag=1;

//自动计算各点 ，知道没有变化

while(flag)

{

cout<<"这是第"<<setw(4)<<roundcount<<"次计算"<<endl;

roundcount++;

// 以下四行从while语句上取出来的

int belongknum[kpoint+1]={0};//每个k点处聚了多少个点；

float sumweikpoint[kpoint+1][wei2+1]={0};//存储属于每个k点处的所有点的总和，用来重新计算中心

float avgstorekpoint[kpoint+1][wei2+1]={0}; //存储每个k点处聚类的平均值；

int wei2jia[kpoint+1]={0}; //第k个点要被顺序存储 在 storebelongkmun。每次加1

//统计每个点属于那些k

for(i=1;i<=wei1;++i)

{

float sumkdistance[kpoint+1]={0};

//计算I这个点到每个k的距离，正好用最小的下标统计它属于低级点

for(j=1;j<=kpoint;++j)

for(k=1;k<=wei2;++k) //wei2+1 改了

{

sumkdistance[j]=sumkdistance[j]+ abs(wei[i][k]\*wei[i][k]-kpointkey[j][k]\*kpointkey[j][k]);

}

// youwenti

float mindistance=sumkdistance[1]; //找到最小距离

for(j=2;j<=kpoint;++j)

mindistance=mindistance<sumkdistance[j]?mindistance:sumkdistance[j];

for(j=1;j<=kpoint;++j)

if(mindistance == sumkdistance[j])

break;

belongknum[j]++;

wei2jia[j]++;

storebelongkmun[j][wei2jia[j]]=i; //第I个点属于第k类了，并把这个点的序号存起来；

cout<<"这是第"<<setw(4)<<i<<"个点，它属于第"<<j<<"类"<<endl;

for(k=1;k<=wei2;++k)

sumweikpoint[j][k]= sumweikpoint[j][k]+wei[i][k];

}

//计算每个点处的平均数：重新确定聚类点

for(i=1;i<=kpoint;++i)

for(j=1;j<=wei2;++j)

{

avgstorekpoint[i][j]=sumweikpoint[i][j]/belongknum[i];

}

for(i=1;i<=kpoint;++i)

{

cout<<"属于第"<<i<<"个点的总数为"<<setw(4)<<belongknum[i]<<"。"<<"这些点分别是：";

for(j=1;j<=belongknum[i];++j)

cout<< storebelongkmun[i][j]<<' ';

cout<<endl;

}

for(i=1;i<=kpoint;++i)

{

cout<<"属于第"<<i<<"个点的中心为:";

for(j=1;j<=wei2;++j)

{

cout<<avgstorekpoint[i][j]<<' ';

}

cout<<endl;

}

cout<<endl;

//把得到的k值重新分配进去。

for(i=1;i<=kpoint;++i)

for(j=1;j<=wei2;++j)

{

kpointkey[i][j]=avgstorekpoint[i][j];

}

int count=0;

//用count来修改flag的值。。。。。。。两种选择，数据大时，可以确定每个k出点的个数，小的时候可以确定每个具体的点处是否有变化

for(i=1;i<=kpoint;++i)

for(j=1;j<=wei2;++j)

if(storebelongkmun[i][j] != contraststorebelongkmun[i][j])

{

contraststorebelongkmun[i][j]=storebelongkmun[i][j] ;

}

else{

count++;

}

if((count == kpoint\*wei2 )|| (roundcount==100))

flag=0;

}

return 0;

}