**课程名称：** 新技术相关课程2

**实验名称：** K-means算法

# 一、实验目的

熟悉并掌握hadooop中MapReduce的用法；

理解并掌握数据挖掘十大算法中一种算法的设计与实现。

# 二、实验内容

对于已经给定的若干二维点，利用MapReduce使用K-means算法为其聚类。

1. 在原数据中随机选择K个类的初始中心；
2. 在第N次迭代中，对任意一个坐标，求其到K个中心的距离，将该坐标归到距离最短的中心所在的类。这里使用Map实现。
3. 利用均值等方法更新该类的中心值。这里使用Reduce实现。新聚类中心的X=∑（x1+x2+…+xn）/n，Y=∑（y1+y2+…+yn）/n。
4. 对于所有的K个聚类中心，如果利用B、C的迭代更新后，值保持不变或改变小于给定精度，则迭代结束，否则继续迭代。

三、实验环境

Linux;

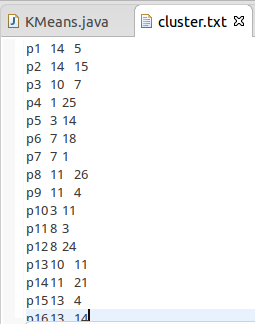
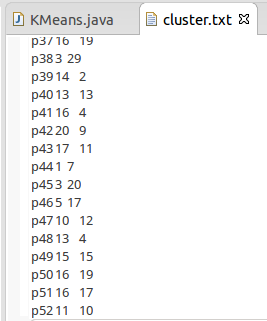
hadoop-0.20.203.0

Eclipse;

# 四、实验结果

**1、数据：**

（1）、为方便聚类展示，我们录入了52个坐标， 如下图：

**…**

（2）、将聚类个数K设为5，最大迭代次数设为100，精度设为0。如下图：



# 2、在集群环境中运行该项目：

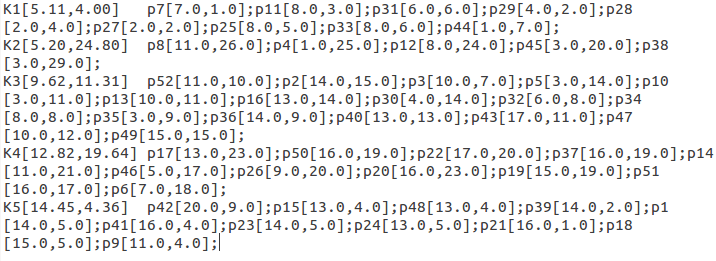
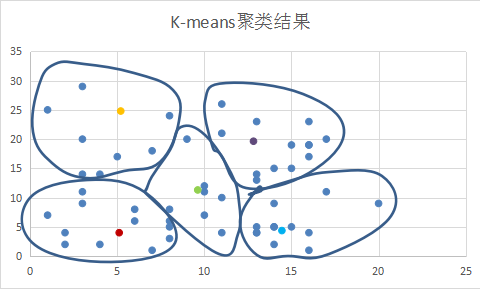


# 3、实验结果：

（1）、迭代次数：



（2）、聚类结果

以上为某一次实验结果，当数据源增加、聚类数个增加、精度更趋于0时，迭代次数便会加大，但迭代次数仍被最大迭代次数限制。

五、附录

**1、方案设计：**

**(1)、理解K-means算法：**

K-means算法以空间中K个点为中心进行聚类，对最靠近他们的对象归类。该算法的最大优势在于简洁和快速，算法的关键在于初始中心的选择和距离公式。算法步骤描述如下：

|  |  |
| --- | --- |
| 序号 | 步骤 |
| 1 | 适当选择c个类的初始中心； |
| 2 | 在第K次迭代中，对任意一个样本，求其到c个中心的距离，将该样本归到距离最短的中心所在的类； |
| 3 | 利用均值等方法更新该类的中心值； |
| 4 | 对于所有的c个聚类中心，如果利用B、C的迭代更新后，值保持不变，则迭代结束，否则继续迭代。  对于所有的c个聚类中心，如果利用B、C的迭代更新后，值保持不变，则迭代结束，否则继续迭代。 |

**(2)、结合MapReduce与K-means算法：**

MapReduce可以有效地将聚类操作中相似的操作拆分开来，让不同计算机去运行，最终整合结果，极高的提高了效率。此时步骤如下：

|  |  |
| --- | --- |
| 序号 | 步骤 |
| 1 | 将数据存放在文件中，使用文件作为输入。并在数据源中随机选择K个数据作为的初始聚类中心； |
| 2 | 使用Map对数据归类：对任意一个数据，求其到K个聚类中心的各自距离，将该数据归类到距离最短的中心所在的类； |
| 3 | 使用Reduce更新该类的中心值，此程序利用均值方法更新； |
| 4 | 对于所有的c个聚类中心，如果利用B、C的迭代更新后，值保持不变，则迭代结束，否则继续迭代。  对于所有的K个聚类中心，如果利用2、3的迭代更新后，值的变化小于等于给定精度，则迭代结束，将数据输出，否则继续迭代。 |

此时有几方面需要注意：

1. 对于某些数据，若完全按照要求迭代至最终结果才退出，迭代的次数可能会超出人们的接受范围，比如成千上万次，造成程序无休止的运行。因此程序应该可以限制最大迭代次数，保证程序的迭代次数在人们可接受范围内。
2. 对于聚类个数K，由于人们的需求值不同，导致各人所需要的K值不同。因此在程序中不可将K值写死，它应该是一个可随时改变的数值。
3. 在实际应用中，若完全迭代至最终收敛聚类中心，可能会极大的增加迭代次数，而这个聚类中心可能与只需要迭代几次就得到的聚类中心相差不多。为解决该问题，我们引进了一个精度值，当新的聚敛中心相比原先的聚类中心偏移不超过该值时，我们认为该中心便是最终收敛聚类中心。

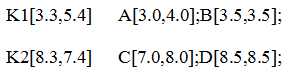
**（3）、数据处理：**

1. 新建坐标类，坐标信息包含坐标名、X坐标、Y坐标。数据源与初始聚类中心以该形式存储。在该类中给出计算两个坐标点之间距离的方法。

B、对于数据源与初始聚类中心，采用文件读入的方式读取。每行代表一个坐标，坐标的名字、X、Y之间以“\t”相隔。例如：



C、结果以聚类中心+“\t”+坐标点+“;”+坐标点+“;”……形式存储至文件中，每行代表一个聚类中心的聚类结果。例如：



**（4）、程序实现：**

**①coord.java**(定义一个坐标类coord)：

1. 坐标类包含属性：坐标名（String）、X坐标(double)、Y坐标(double)。类的初始化支持有参及无参。
2. 给出所有属性的getter、setter方法。
3. 在类中定义一个距离计算方法，其接受参数为一个坐标点coord，返回本坐标与该坐标的距离(double)，距离=√￣[(x1-x2)^2+(y1-y2)^2]。

**②getFileCoord.java**(定义一个获取初始聚类中心的类)：

1. 该类包含一个属性，类型为Map<String,coord>，用来存储坐标，其中String是坐标名，coord是坐标。其初始化接受一个字符串作为读取初始聚类中心的文件的路径。初始化时读取该文件，文件默认一行代表一个坐标点，坐标的信息用“\t”隔开，通过该方式得到文件中的所有坐标。
2. 定义一个查找坐标方法，该方法接受一个字符串作为查找的坐标名，返回类型是coord坐标。该方法通过接受参数查找Map中与之相同坐标名的坐标，将之返回；若查找不到，则返回一个空坐标。
3. 定义一个返回Map里数据个数的方法，不需接受参数，直接返回Map的数据个数，方便在随机抽取初始聚类中心时定义范围。

**③KMeans.java**(用MapReduce实现K-means)：

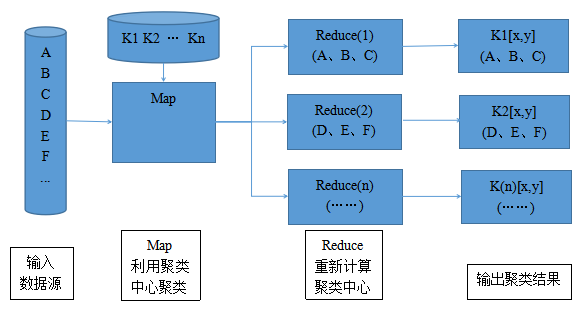
1. 首先定义几个属性：聚类个数K值大小(int)、最大迭代次数(int)、收敛精度(double)、旧聚类中心（ArrayList<coord>）、新聚类中心（ArrayList<coord>），并初始化getFileCoord(“文件路径”)类。
2. 定义Mapper，其接收值(value)为数据源中的坐标点信息，将包含坐标点信息Text类型的value转化为对应的坐标(coord)，比较该坐标到每一个中心点（从旧聚类中心获取）的距离，取得距离最小的中心点。将该中心点的所有信息（名字、x、y）转化为Text作为key输出，该坐标的所有信息（名字、x、y）转化为Text作为value输出。
3. 定义Reducer，使用Text接受key，Iterable接收value。遍历Iterable中所有的value，先将每个value转化为一个坐标，求得所有坐标的x之和与y之和，然后计算x的均值和y的均值作为新聚类中心，并将该点赋值到新聚类中心点列表其所在的位置中。将新聚类中心的所有信息（名字、x、y）转化为Text作为key输出，本聚类中所有坐标的信息（名字、x、y）转化为Text作为value输出。
4. 编写main：初始化新旧聚类中心点，并随机抽取源数据中的K个数据作为初始聚类中心，添加到旧聚类中心中。迭代如下步骤：若输出路径已存在，则删除该文件夹。定义一个Job，定义该Job的Mapper为B定义的Mapper, 定义该Job的Reduce为C定义的Reducer；定义该Job的Map、Reduce输出类型（Text）；设置该Job的输入路径为数据源文件路径，并设置输出路径；等待Job完成再往下运行。判断聚类中心是否已经收敛或迭代次数是否已达最大迭代次数，若不收敛或未达到最大迭代次数，则将所有新聚类中心点赋值到旧聚类中心点，继续迭代；否则迭代结束。

**（5）、项目检测：**

编写完项目后，将项目导出，在hadoop集群上运行，检测项目是否有误。

1. **设计图：**

迭代过程如下：



**3、调试心得：**

（1）本来要使用hbase作为数据存储，但在创建完表和数据后发现在hadoop里找不到hadoop.hbase包，导致无法使用hbase接口，无法实现hbase操作。后改为文件存储。

（2）采用文件存储数据，坐标的存储格式要与定义的格式一致，否则读取出现问题。

（3）可能为版本的原因，在Reduce中使用Iterator接收value值失败，改为Iterable成功。

（4）Map和Reduce的输入输出类型要设置与实际输入输出类型一致，曾不小心将Map的实际Text输出设置为了IntWritable类型，导致报错。

（5）在Map的聚类中，有最近距离的判断，其中最小距离有个初始值。每个坐标点的聚类判断都应该再次设置此初始值，否则会沿用上一次的最小距离，导致结果错误。

（6）迭代计算时，Job的输出路径会重叠，每次迭代前要删去该文件夹。因为文件夹里面有内容，所以无法简单使用File.delete()，要先删去文件夹里面的东西，再删去空文件夹。

（7）Map获取的value似乎是文件中一行一行的数据，原本以为是整个文件的数据，导致使用时发生了错误。

（8）随机抽取原数据中K个数据作为聚类中心点时要注意不要抽到相同的点。

1. **人员分工：**

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 学号 | 分工 | 工作量 |
| 林锦泽 | 2013212124 | 将其他成员提供的K-means伪代码与hadoop的MapReduce相结合，写成最终项目。 | 20% |
| 张学源 | 2013212131 | 将其他成员提供的K-means伪代码与hadoop的MapReduce相结合，写成最终项目。 | 20% |
| 暴智伟 | 2013212075 | 学习K-means算法，并将之写成初步伪代码。 | 20% |
| 梁欣媛 | 2013212043 | 学习K-means算法，并将之写成初步伪代码。 | 20% |
| 赵烨 | 2013212106 | 学习K-means算法，并将之写成初步伪代码。 | 20% |

1. **程序：**

（1）、coord.java:

**public** **class** coord {

//坐标信息

**private** String name;

**private** **double** x;

**private** **double** y;

//无参初始化

**public** coord(){

**this**.name = "";

**this**.x=0.0;

**this**.y=0.0;

}

//有参初始化

**public** coord(String name,**double** x,**double** y){

**this**.name=name;

**this**.x=x;

**this**.y=y;

}

//与另一个坐标的距离

**public** **double** getDistance(coord anothercoord){

**return** Math.*sqrt*(Math.*pow*(**this**.x-anothercoord.getX(),2) + Math.*pow*(**this**.y-anothercoord.getY(), 2));

}

//getter和setter

**public** String getName(){

**return** **this**.name;

}

**public** **void** setName(String name){

**this**.name=name;

}

**public** **double** getX(){

**return** **this**.x;

}

**public** **void** setX(**double** x){

**this**.x=x;

}

**public** **double** getY(){

**return** **this**.y;

}

**public** **void** setY(**double** y){

**this**.y=y;

}

}

(2) getFileCoord.java:

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.util.HashMap;

**import** java.util.Map;

**import** org.apache.hadoop.io.IOUtils;

**public** **class** getFileCoord {

**private** Map<String,coord> urlMap = **new** HashMap<String,coord>();

**private** **int** count = 0;

//获取聚点文件对象

**public** **void** initialize(File file) **throws** IOException {

BufferedReader in = **null**;

**try** {

in = **new** BufferedReader(**new** InputStreamReader(**new** FileInputStream(file)));

String line;

**while** ((line = in.readLine()) != **null**) {

String [] strKey = line.split("\t");

urlMap.put(strKey[0],parse(line));

count++;

}

} **finally** {

IOUtils.*closeStream*(in);

}

}

//通过一行数据生成一个聚点坐标

**public** coord parse(String line){

String [] strPlate = line.split("\t");

coord Dmurl = **new** coord(strPlate[0],Integer.*parseInt*(strPlate[1]),Integer.*parseInt*(strPlate[2]));

**return** Dmurl;

}

//返回Map数据个数

**public** **int** getCount(){

**return** **this**.count;

}

//通过名字查找聚点坐标

**public** coord getUrlCode(String cluster){

coord returnCode = **null**;

coord dmUrl = (coord)urlMap.get(cluster);

**if**(dmUrl == **null**){

returnCode = **null**;

}**else**{

returnCode =dmUrl;

}

**return** returnCode;

}

}

(3)KMeans.java:

**import** java.io.File;

**import** java.io.IOException;

**import** java.text.DecimalFormat;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Random;

**import** org.apache.hadoop.conf.Configuration;

**import** org.apache.hadoop.fs.Path;

**import** org.apache.hadoop.io.LongWritable;

**import** org.apache.hadoop.io.Text;

**import** org.apache.hadoop.mapreduce.Job;

**import** org.apache.hadoop.mapreduce.Mapper;

**import** org.apache.hadoop.mapreduce.Reducer;

**import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

**public** **class** KMeans {

**static** **int** *K* = 5; //聚类数

**static** **int** *iteration\_number* =100; //最大迭代次数

**static** **double** *accuracy* = 2; //精度

**static** **int** *over* = 0; //是否所有聚类中心符合精度

**static** **int**[] *center* = **new** **int**[*K*]; //初始聚类中心的位置

**static** getFileCoord *drp* ; //用于读取初始聚类中心

**static** List<coord> *oldCenter* = **new** ArrayList<coord>(*K*); //旧聚类中心

**static** List<coord> *newCenter* = **new** ArrayList<coord>(*K*); //新聚类中心

**public** **static** **class** KmeansMapper **extends** Mapper<LongWritable, Text, Text, Text>

{

**private** coord record0 = **new** coord();

**private** coord record1 = **new** coord();

**private** **static** **double** *Min\_distance* = 9999;

**private** **int** tmpK = 0;

**private** Text tKey = **new** Text();

**private** Text tValue = **new** Text();

//根据聚类坐标，把文件中的点进行类别划分

**public** **void** map(LongWritable key, Text value,Context output)

**throws** IOException, InterruptedException {

String [] strArr = value.toString().split("\n");

String[] line;

**double** tempx = 0;

**double** tempy = 0;

//将所有点分类

**for**(**int** n=0;n<strArr.length;n++){

*Min\_distance* = 9999;

line = strArr[n].split("\t");

record1.setName(line[0]);

record1.setX(Double.*parseDouble*(line[1]));

record1.setY(Double.*parseDouble*(line[2]));

//聚类到最近点

**for**(**int** i=1; i <= *K*; i++){

record0 = *oldCenter*.get(i-1);

**if**(record0.getDistance(record1) < *Min\_distance*){

tmpK = i;

tempx = record0.getX();

tempy = record0.getY();

*Min\_distance* = record0.getDistance(record1);

}

}

//输出

tKey.set("C"+tmpK+"\t"+tempx+"\t"+tempy);

tValue.set(record1.getName()+"\t"+record1.getX()+"\t"+record1.getY());

output.write(tKey, tValue);

}

}

}

//计算新的聚类中心

**public** **static** **class** KmeansReducer **extends** Reducer<Text, Text, Text, Text> {

**private** Text tKey = **new** Text();

**private** Text tValue = **new** Text();

**public** **void** reduce(Text key, Iterable<Text> value,Context output)

**throws** IOException, InterruptedException {

**double** avgX=0;

**double** avgY=0;

**double** sumX=0;

**double** sumY=0;

**int** count=0;

String [] strValue = **null**;

String coords="";

//整合此聚类的所有坐标

**for**(Text one\_value : value){

strValue = one\_value.toString().split("\t");

Double.*valueOf*(strValue[1]).doubleValue();

sumX = sumX + Double.*valueOf*(strValue[1]).doubleValue();

sumY = sumY + Double.*valueOf*(strValue[2]).doubleValue();

coords+=strValue[0]+"["+strValue[1]+","+strValue[2]+"];";

count++;

}

//获取聚类中心的新坐标

avgX = sumX/count;

avgY = sumY/count;

//比较新旧聚类中心是否符合精度,并提交新聚类中心

String[] s = key.toString().split("\t");

**int** cc = Integer.*parseInt*(key.toString().split("\t")[0].substring(1));

coord oc = **new** coord("OC",Double.*parseDouble*(s[1]),Double.*parseDouble*(s[2]));

coord nc = **new** coord("p"+cc,avgX,avgY);

*newCenter*.set(cc-1, nc);

**if**(oc.getDistance(nc)<=*accuracy*)

*over* ++;

//输出

DecimalFormat df = **new** DecimalFormat("######0.00");

tKey.set("K"+key.toString().split("\t")[0].substring(1)+"["+df.format(avgX )+ "," + df.format(avgY)+"]");

tValue.set(coords+"");

output.write(tKey, tValue);

}

}

**public** **static** **void** main(String[] args) **throws** Exception {

*configure*();

**int** run\_count = 0;

**for**(**int** i = 0;i<*K*;i++){

coord co = **new** coord("p1",0,0);

*oldCenter*.add(co);

*newCenter*.add(co);

}

//设置初始聚类中心

*setCenter*();

**while**(*iteration\_number* > 0 && *over*<*K*)

{

*over* = 0;

//若输出路径存在,删去该文件夹

File file = **new** File("./src/Result/a");

*DeleteFile*(file);

//新建Job

Configuration configuration = **new** Configuration();

Job job = **new** Job(configuration,"Kmeans");

job.setJarByClass(KMeans.**class**);

// 设置Map输出的key和value的类型

job.setMapOutputKeyClass(Text.**class**);

job.setMapOutputValueClass(Text.**class**);

// 设置Reduce输出的key和value的类型

job.setOutputKeyClass(Text.**class**);

job.setOutputValueClass(Text.**class**);

// 设置Mapper和Reducer

job.setMapperClass(KmeansMapper.**class**);

job.setReducerClass(KmeansReducer.**class**);

// 设置输入输出目录

Path DocumentCollection = **new** Path("./src/cluster.txt");//输入文件路径

Path PostingList = **new** Path("./src/Result/a");//输出文件路径

FileInputFormat.*addInputPath*(job, DocumentCollection);

FileOutputFormat.*setOutputPath*(job, PostingList);

//等待结束

job.waitForCompletion(**true**);

//将新聚类中心赋给旧聚类中心

*resetCenter*();

*iteration\_number*--;

run\_count++;

}

System.***out***.println("本次聚类共迭代"+run\_count+"次");

System.*exit*(1);

}

//设置初始聚类中心

**private** **static** **void** setCenter() {

// **TODO** Auto-generated method stub

**int** count = 0;

**int** flag = 0;

Random rdm = **new** Random(System.*currentTimeMillis*());

**int** r = Math.*abs*(rdm.nextInt());

**while**(count<*K*)

{

flag = 0;

*center*[count] = r % *drp*.getCount();

r = (*center*[count]+1)\*157;

**for**(**int** i = 0;i<count;i++)

**if**(*center*[count]==*center*[i])

flag = 1;

**if**(flag==0)

count++;

}

**for**(**int** i=0;i<*K*;i++)

*oldCenter*.set(i, *drp*.getUrlCode("p"+*center*[i]+""));

}

//删除文件夹

**private** **static** **void** DeleteFile(File file) {

// **TODO** Auto-generated method stub

**if**(file.exists()){

**if**(file.isFile())

file.delete();

**else** **if**(file.isDirectory()){

File files[] = file.listFiles();

**for**(**int** i=0;i<files.length;i++)

*DeleteFile*(files[i]);

file.delete();

}

}

}

//用于获取聚类中心坐标

**public** **static** **void** configure() {

*drp* = **new** getFileCoord();

**try** {

*drp*.initialize(**new** File("src/cluster.txt"));

} **catch** (IOException e) {

**throw** **new** RuntimeException(e);

}

}

//重设聚类中心位置

**private** **static** **void** resetCenter() {

**for**(**int** i=0;i<*K*;i++)

*oldCenter*.set(i, *newCenter*.get(i));

}

}