数据挖掘

基于java实现kmeans算法进行图像分割

# kmeans算法概述

## 基本简介

k-means 算法接受输入量 k ；

然后将n个数据对象划分为 k个聚类以便使得所获得的聚类满足：同一聚类中的对象相似度较高；

而不同聚类中的对象相似度较小。

聚类相似度是利用各聚类中对象的均值所获得一个“中心对象”（引力中心）来进行计算的。

## 处理流程

k-means算法的处理流程说明如下：

1.首先从n个数据对象任意选择 k 个对象作为初始聚类中心；

2.而对于所剩下其它对象，则根据它们与这些聚类中心的相似度（距离），分别将它们分配给与其最相似的（聚类中心所代表的）聚类；

3.然后再计算每个所获新聚类的聚类中心（该聚类中所有对象的均值）；

4.不断重复这一过程直到标准测度函数开始收敛为止。一般都采用均方差作为标准测度函数。

k个聚类具有以下特点：各聚类本身尽可能的紧凑，而各聚类之间尽可能的分开。

## 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

# 图示算法

K-Means算法主要解决的问题如下图所示。我们可以看到，在图的左边有一些点，我们用肉眼可以看出来有四个点群，但是我们怎么通过计算机程序找出这几个点群来呢？于是就出现了我们的K-Means算法





**A，B，C，D，E是五个在图中点。而灰色的点是我们的种子点，也就是我们用来找点群的点**。有两个种子点，所以K=2。

然后，K-Means的算法如下：

随机在图中取K（这里K=2）个种子点。

然后对图中的所有点求到这K个种子点的距离，假如点Pi离种子点Si最近，那么Pi属于Si点群。（我们可以看到A，B属于上面的种子点，C，D，E属于下面中部的种子点）

接下来，我们要移动种子点到属于他的“点群”的中心。（见图上的第三步）

然后重复第2和第3步，直到，种子点没有移动（我们可以看到图中的第四步上面的种子点聚合了A，B，C，下面的种子点聚合了D，E）。

# 求点群中心的算法

1.Minkowski Distance公式



λ可以随意取值，可以是负数，也可以是正数，或是无穷大。

2. Euclidean Distance公式



也就是第一个公式λ=2的情况

3. CityBlock Distance公式



也就是第一个公式λ=1的情况

这三个公式的求中心点有一些不一样的地方，三个图表示他们是怎么个逼近中心的，第一个图以星形的方式，第二个图以同心圆的方式，第三个图以菱形的方式。

 1 2 3

# 编程思路

一个像素点，在一幅图片中存储需要5个不同的数据，分别是[x,y,r,g,b]，这个x和y分别对应着在图片中的位置，第x行第y列，然后rgb这三个就是电脑显示的三原色（红绿蓝），也就是说，不论什么颜色，都是由这三个三原色组成显示的。

其实xy这两个维度和聚类无关，我们图像分割主要就是想要保证一个颜色的在一起，所以只要进行RGB的三个维度聚类就可以。

把每个像素点作为一个类，这个类包含着RGB三个成员变量，再把图片读取成二维数组的形式，就可以保持X和Y不变。

使用欧几里得距离：

二维空间的公式：d=sqrt( (x1-x2)^2+(y1-y2)^2 )

三维空间的公式：d=√( (x1-x2)^2+(y1-y2)^2+(z1-z2)^2 )

n维空间的公式：ρ(**A**，**B**) =√ [ ∑( a[i] - b[i] )^2 ] (i = 1，2，…，n)

# 研究成果与应用

原图。注意：迭代次数均为10

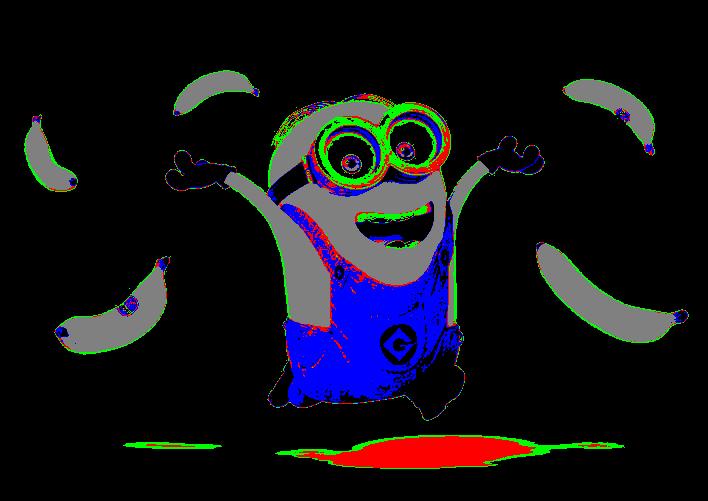


当聚类中心k=2时，图像分割后

当聚类中心k=4时，图像分割后



当聚类中心k=6时，图像分割后



# 总结

## 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 。

## 缺点

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。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。

# 代码

我是在Macbook上做的测试，文件路径写法与Windows不一样。

package kmeansimg;

import java.awt.Color;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

public class ImageCluster {

//主要功能就是读取一副图像，再对图像进行分割

//需要分类的簇数

private int k;

//迭代次数

private int m;

//数据集合

private dataItem[][] source;

//中心集合

private dataItem[] center;

//统计每个簇的各项数据的总和，用于计算新的点数

private dataItem[] centerSum;

//读取指定目录的图片数据，并且写入数组，这个数据要继续处理

private int[][] getImageData(String path){

BufferedImage bi=null;

try{

bi=ImageIO.read(new File(path));

}catch (IOException e){

e.printStackTrace();

}

int width=bi.getWidth();

int height=bi.getHeight();

int [][] data=new int[width][height];

for(int i=0;i<width;i++)

for(int j=0;j<height;j++)

data[i][j]=bi.getRGB(i, j);

return data;

}

//用来处理获取的像素数据，提取我们需要的写入dataItem数组

private dataItem[][] InitData(int [][] data){

dataItem[][] dataitems=new dataItem[data.length][data[0].length];

for(int i=0;i<data.length;i++){

for(int j=0;j<data[0].length;j++){

dataItem di=new dataItem();

Color c=new Color(data[i][j]);

di.r=(double)c.getRed();

di.g=(double)c.getGreen();

di.b=(double)c.getBlue();

di.group=1;

dataitems[i][j]=di;

}

}

return dataitems;

}

//生成随机的初始中心

private void initCenters(int k){

center =new dataItem[k];

centerSum=new dataItem[k];//用来统计每个聚类里面的RGB分别之和，方便计算均值

int width,height;

for(int i=0;i<k;i++){

//boolean flag=true;

dataItem cent=new dataItem();

dataItem cent2=new dataItem();

width=(int)(Math.random()\*source.length);

height=(int)(Math.random()\*source[0].length);

cent.group=i;

cent.r=(double)source[width][height].r;

cent.g=(double)source[width][height].g;

cent.b=(double)source[width][height].b;

center[i]=cent;

cent2.r=cent.r;

cent2.g=cent.g;

cent2.b=cent.b;

cent2.group=0;

centerSum[i]=cent2;

width=0;height=0;

}

System.out.println("初始"+k+"个中心");

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

{

System.out.println("("+center[i].r+","+center[i].g+","+center[i].b+")");

}

}

//计算两个像素之间的欧式距离，用RGB作为三维坐标

private double distance(dataItem first,dataItem second){

double distance=0;

distance=Math.sqrt(Math.pow((first.r-second.r),2)+Math.pow((first.g-second.g),2)+

Math.pow((first.b-second.b),2));

return distance;

}

//返回一个数组中最小的坐标

private int minDistance(double[] distance){

double minDistance=distance[0];

int minLocation=0;

for(int i=0;i<distance.length;i++){

if(distance[i]<minDistance){

minDistance=distance[i];

minLocation=i;

}else if(distance[i]==minDistance){

if((Math.random()\*10)<5){

minLocation=i;

}

}

}

return minLocation;

}

//每个点进行分类

private void clusterSet(){

int group=-1;

double distance[]=new double[k];

for(int i=0;i<source.length;i++){

for(int j=0;j<source[0].length;j++){

//求出距离中心点最短的中心

for(int q=0;q<center.length;q++){

distance[q]=distance(center[q],source[i][j]);

}

group=minDistance(distance);//寻找该点最近的中心

source[i][j].group=group;//把该点进行分类

centerSum[group].r+=source[i][j].r;//分类完求出该类的RGB和

centerSum[group].g+=source[i][j].g;

centerSum[group].b+=source[i][j].b;

centerSum[group].group+=1;//这个就是用来统计聚类里有几个点

group=-1;

}

}

}

//设置新的中心

public void setNewCenter(){

for(int i=0;i<centerSum.length;i++){

System.out.println(i+":"+centerSum[i].group+":"+centerSum[i].r+":"+centerSum[i].g+":"+centerSum[i].b);

//取平均值为新的中心

center[i].r=(int)(centerSum[i].r/centerSum[i].group);

center[i].g=(int)(centerSum[i].g/centerSum[i].group);

center[i].b=(int)(centerSum[i].b/centerSum[i].group);

//重置之前的求和结果

centerSum[i].r=center[i].r;

centerSum[i].g=center[i].g;

centerSum[i].b=center[i].b;

centerSum[i].group=0;

}

}

//输出聚类好的数据

private void ImagedataOut(String path){

Color c0=new Color(255,0,0);

Color c1=new Color(0,255,0);

Color c2=new Color(0,0,255);

Color c3=new Color(128,128,128);

BufferedImage nbi=new BufferedImage(source.length,source[0].length,BufferedImage.TYPE\_INT\_RGB);

for(int i=0;i<source.length;i++){

for(int j=0;j<source[0].length;j++){

if(source[i][j].group==0)

nbi.setRGB(i, j, c0.getRGB());

else if(source[i][j].group==1)

nbi.setRGB(i, j, c1.getRGB());

else if(source[i][j].group==2)

nbi.setRGB(i, j, c2.getRGB());

else if (source[i][j].group==3)

nbi.setRGB(i, j, c3.getRGB());

//Color c=new Color((int)center[source[i][j].group].r,

// (int)center[source[i][j].group].g,(int)center[source[i][j].group].b);

//nbi.setRGB(i, j, c.getRGB());

}

}

try{

ImageIO.write(nbi, "jpg", new File(path));

}catch(IOException e){

e.printStackTrace();

}

}

//进行kmeans计算的核心函数

public void kmeans(String path,int k,int m){

source=InitData(getImageData(path));

/\*测试输出

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

for(int j=0;j<source[0].length;j++)

System.out.println("("+source[i][j].x+","+source[i][j].y+","+source[i][j].r+","+source[i][j].g+","+source[i][j].b+")");

\*/

this.k=k;

this.m=m;

//初始化聚类中心

initCenters(k);

/\*测试输出

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

System.out.println("("+center[i].x+","+center[i].y+","+center[i].r+","+center[i].g+","+center[i].b+")");

\*/

//进行m次聚类

for(int level=0;level<m;level++){

clusterSet();

setNewCenter();

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

{

System.out.println("("+center[i].r+","+center[i].g+","+center[i].b+")");

}

}

clusterSet();

System.out.println("第"+m+"次迭代完成，聚类中心为：");

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

{

System.out.println("("+center[i].r+","+center[i].g+","+center[i].b+")");

}

System.out.println("迭代总次数："+m);//进行图像输出，这个随意改

ImagedataOut("/Users/mac/Desktop/kmeansimg.jpg");//输出文件

}

}

再写个测试类

package kmeansimg;

public class Test {

public static void main(String[] args){

ImageCluster ic=new ImageCluster();

ic.kmeans("/Users/mac/Desktop/xhr.jpeg",4,10);

}

}