K中心点算法（K-medoids）

前面介绍了k-means算法，并列举了该算法的缺点。而K中心点算法（K-medoids）正好能解决k-means算法中的“噪声”敏感这个问题。

如何解决的呢？

首先，我们得介绍下k-means算法为什么会对“噪声”敏感。还记得K-means寻找质点的过程吗？对某类簇中所有的样本点维度求平均值，即获得该类簇质点的维度。当聚类的样本点中有“噪声”（离群点）时，在计算类簇质点的过程中会受到噪声异常维度的干扰，造成所得质点和实际质点位置偏差过大，从而使类簇发生“畸变”。

Eg: 类簇C1中已经包含点A(1,1)、B(2,2)、 C(1,2)、 D(2,1)， 假设N(100,100)为异常点，当它纳入类簇C1时，计算质点Centroid((1+2+1+2+100)/5,(1+2+2+1+100)/5)=centroid(21,21),此时可能造成了类簇C1质点的偏移，在下一轮迭代重新划分样本点的时候，将大量不属于类簇C1的样本点纳入，因此得到不准确的聚类结果。

为了解决该问题，K中心点算法（K-medoids）提出了新的质点选取方式，而不是简单像k-means算法采用均值计算法。在K中心点算法中，每次迭代后的质点都是从聚类的样本点中选取，而选取的标准就是当该样本点成为新的质点后能提高类簇的聚类质量，使得类簇更紧凑。该算法使用绝对误差标准来定义一个类簇的紧凑程度。

[聚类分析（三） <wbr>K中心点算法（k-mediods）](http://photo.blog.sina.com.cn/showpic.html#blogid=4882f26d0100pitr&url=http://s15.sinaimg.cn/orignal/4882f26dg9e444eade41e)  (p是空间中的样本点，Oj是类簇Cj的质点)

如果某样本点成为质点后，绝对误差能小于原质点所造成的绝对误差，那么K中心点算法认为该样本点是可以取代原质点的，在一次迭代重计算类簇质点的时候，我们选择绝对误差最小的那个样本点成为新的质点。

Eg：样本点A –>E1=10

样本点B –>E2=11

样本点C –>E3=12

原质点O–>E4=13，那我们选举A作为类簇的新质点。

与K-means算法一样，K-medoids也是采用欧几里得距离来衡量某个样本点到底是属于哪个类簇。终止条件是，当所有的类簇的质点都不在发生变化时，即认为聚类结束。

该算法除了改善K-means的“噪声”敏感以后，其他缺点和K-means一致，并且由于采用新的质点计算规则，也使得算法的时间复杂度上升：O（k(n-k)2）

Java实现代码如下：

package com.kmedoids;  
import java.util.ArrayList;  
public class Cluster {  
    private String clusterName; // 类簇名  
    private Medoid medoid; // 类簇的质点  
    private ArrayList<DataPoint> dataPoints; // 类簇中各样本点  
  
    public Cluster(String clusterName) {  
        this.clusterName = clusterName;  
        this.medoid = null; // will be set by calling setCentroid()  
        dataPoints = new ArrayList<DataPoint>();  
    }  
  
    public void setMedoid(Medoid c) {  
        medoid = c;  
    }  
  
    public Medoid getMedoid() {  
        return medoid;  
    }  
  
     
    public void addDataPoint(DataPoint dp) { // called from CAInstance  
        dp.setCluster(this);// 标注该类簇属于某点,计算欧式距离  
        this.dataPoints.add(dp);  
    }  
  
    public void removeDataPoint(DataPoint dp) {  
        this.dataPoints.remove(dp);  
    }  
  
    public int getNumDataPoints() {  
        return this.dataPoints.size();  
    }  
  
    public DataPoint getDataPoint(int pos) {  
        return (DataPoint) this.dataPoints.get(pos);  
    }  
  
  
    public String getName() {  
        return this.clusterName;  
    }  
  
    public ArrayList<DataPoint> getDataPoints() {  
        return this.dataPoints;  
    }  
}

------------------------------------

package com.kmedoids;  
import java.util.ArrayList;  
  
public class DataPoint {  
    private double dimension[]; //样本点的维度  
    private String pointName; //样本点名字  
    private Cluster cluster; //类簇  
    private double euDt;//样本点到质点的距离  
  
    public DataPoint(double dimension[], String pointName) {  
        this.dimension = dimension;  
        this.pointName = pointName;  
        this.cluster = null;  
    }  
  
    public void setCluster(Cluster cluster) {  
        this.cluster = cluster;  
    }  
  
  
     
    public double calEuclideanDistanceSum() {  
        double sum=0.0;  
        Cluster cluster=this.getCluster();  
        ArrayList<DataPoint> dataPoints=cluster.getDataPoints();  
  
        for(int i=0;i<dataPoints.size();i++){  
            double[] dims=dataPoints.get(i).getDimensioin();  
            for(int j=0;j<dims.length;j++){  
                 double temp=Math.pow((dims[j]-this.dimension[j]),2);  
                 sum=sum+temp;  
            }  
        }  
  
        return Math.sqrt(sum);  
    }  
  
     
    public double testEuclideanDistance(Medoid c) {  
        double sum=0.0;  
        double[] cDim=c.getDimensioin();  
  
        for(int i=0;i<dimension.length;i++){  
           double temp=Math.pow((dimension[i]-cDim[i]),2);  
           sum=sum+temp;  
        }  
  
        return Math.sqrt(sum);  
    }  
  
    public double[] getDimensioin() {  
        return this.dimension;  
    }  
  
    public Cluster getCluster() {  
        return this.cluster;  
    }  
  
    public double getCurrentEuDt() {  
        return this.euDt;  
    }  
  
    public String getPointName() {  
        return this.pointName;  
    }  
}  
-------------------------------

package com.kmedoids;  
import java.util.ArrayList;  
  
public class Medoid{  
  
    private double dimension[]; // 质点的维度  
    private Cluster cluster; //所属类簇  
    private double etdDisSum;//Medoid到本类簇中所有的欧式距离之和  
  
  
    public Medoid(double dimension[]) {  
        this.dimension = dimension;  
    }  
  
    public void setCluster(Cluster c) {  
        this.cluster = c;  
    }  
  
    public double[] getDimensioin() {  
        return this.dimension;  
    }  
  
    public Cluster getCluster() {  
        return this.cluster;  
    }  
  
    public void calcMedoid() {// 取代价最小的点  
        calcEtdDisSum();  
        double minEucDisSum = this.etdDisSum;  
        ArrayList<DataPoint> dps = this.cluster.getDataPoints();  
        for (int i = 0; i < dps.size(); i++) {  
            double tempeucDisSum = dps.get(i).calEuclideanDistanceSum();  
            if (tempeucDisSum < minEucDisSum) {  
                dimension = dps.get(i).getDimensioin();  
                minEucDisSum=tempeucDisSum;  
            }  
        }  
    }  
  
    // 计算该Medoid到同类簇所有样本点的欧斯距离和  
    private void calcEtdDisSum() {  
        double sum=0.0;  
        Cluster cluster=this.getCluster();  
        ArrayList<DataPoint> dataPoints=cluster.getDataPoints();  
  
        for(int i=0;i<dataPoints.size();i++){  
            double[] dims=dataPoints.get(i).getDimensioin();  
            for(int j=0;j<dims.length;j++){  
                 double temp=Math.abs(dims[j]-this.dimension[j]);  
                 sum=sum+temp;  
            }  
        }  
        etdDisSum= sum;  
    }  
}

--------------------------

package com.kmedoids;  
  
import java.util.ArrayList;  
  
public class ClusterAnalysis {  
  
    private Cluster[] clusters;// 所有类簇  
    private int miter;// 迭代次数  
    private ArrayList<DataPoint> dataPoints = new ArrayList<DataPoint>();// 所有样本点  
    private int dimNum;//维度  
  
    public ClusterAnalysis(int k, int iter, ArrayList<DataPoint> dataPoints,int dimNum) {  
        clusters = new Cluster[k];// 类簇种类数  
        for (int i = 0; i < k; i++) {  
            clusters[i] = new Cluster("Cluster:" + i);  
        }  
        this.miter = iter;  
        this.dataPoints = dataPoints;  
        this.dimNum=dimNum;  
    }  
  
    public int getIterations() {  
        return miter;  
    }  
  
    public ArrayList<DataPoint>[] getClusterOutput() {  
        ArrayList<DataPoint> v[] = new ArrayList[clusters.length];  
        for (int i = 0; i < clusters.length; i++) {  
            v[i] = clusters[i].getDataPoints();  
        }  
        return v;  
    }  
  
     
    public void startAnalysis(double[][] medoids) {  
  
        setInitialMedoids(medoids);  
  
        double[][] newMedoids=medoids;  
        double[][] oldMedoids=new double[medoids.length][this.dimNum];  
  
        while(!isEqual(oldMedoids,newMedoids)){  
            for(int m = 0; m < clusters.length; m++){//每次迭代开始情况各类簇的点  
                clusters[m].getDataPoints().clear();  
            }  
            for (int j = 0; j < dataPoints.size(); j++) {  
                int clusterIndex=0;  
                double minDistance=Double.MAX\_VALUE;  
  
                for (int k = 0; k < clusters.length; k++) {//判断样本点属于哪个类簇  
                    double eucDistance=dataPoints.get(j).testEuclideanDistance(clusters[k].getMedoid());  
                    if(eucDistance<minDistance){  
                        minDistance=eucDistance;  
                        clusterIndex=k;  
                    }  
                }  
  
               //将该样本点添加到该类簇  
                clusters[clusterIndex].addDataPoint(dataPoints.get(j));  
  
            }  
  
            for(int m = 0; m < clusters.length; m++){  
                clusters[m].getMedoid().calcMedoid();//重新计算各类簇的质点  
            }  
  
            for(int i=0;i<medoids.length;i++){  
                for(int j=0;j<this.dimNum;j++){  
                    oldMedoids[i][j]=newMedoids[i][j];  
                }  
            }  
  
  
            for(int n=0;n<clusters.length;n++){  
                newMedoids[n]=clusters[n].getMedoid().getDimensioin();  
            }  
  
            this.miter++;  
        }  
  
  
    }  
  
    private void setInitialMedoids(double[][] medoids) {  
        for (int n = 0; n < clusters.length; n++) {  
            Medoid medoid = new Medoid(medoids[n]);  
            clusters[n].setMedoid(medoid);  
            medoid.setCluster(clusters[n]);  
        }  
    }  
  
     
    private boolean isEqual(double[][] oldMedoids,double[][] newMedoids){  
        boolean flag=false;  
        for(int i=0;i<oldMedoids.length;i++){  
            for(int j=0;j<oldMedoids[i].length;j++){  
                if(oldMedoids[i][j]!=newMedoids[i][j]){  
                    return flag;  
                }  
            }  
        }  
        flag=true;  
        return flag;  
    }  
}  
--------------------------------------------

package com.kmedoids;  
  
import java.util.ArrayList;  
import java.util.Iterator;  
  
public class TestMain {  
    public static void main (String args[]){  
        ArrayList<DataPoint> dataPoints = new ArrayList<DataPoint>();  
  
         
        double[] a={2,3};  
        double[] b={2,4};  
        double[] c={1,4};  
        double[] d={1,3};  
        double[] e={2,2};  
        double[] f={3,2};  
  
        double[] g={8,7};  
        double[] h={8,6};  
        double[] i={7,7};  
        double[] j={7,6};  
        double[] k={8,5};  
  
        double[] l={100,2};//孤立点  
  
        double[] m={8,20};  
        double[] n={8,19};  
        double[] o={7,18};  
        double[] p={7,17};  
        double[] q={7,20};  
  
        dataPoints.add(new DataPoint(a,"a"));  
        dataPoints.add(new DataPoint(b,"b"));  
        dataPoints.add(new DataPoint(c,"c"));  
        dataPoints.add(new DataPoint(d,"d"));  
        dataPoints.add(new DataPoint(e,"e"));  
        dataPoints.add(new DataPoint(f,"f"));  
  
        dataPoints.add(new DataPoint(g,"g"));  
        dataPoints.add(new DataPoint(h,"h"));  
        dataPoints.add(new DataPoint(i,"i"));  
        dataPoints.add(new DataPoint(j,"j"));  
        dataPoints.add(new DataPoint(k,"k"));  
  
        dataPoints.add(new DataPoint(l,"l"));  
  
        dataPoints.add(new DataPoint(m,"m"));  
        dataPoints.add(new DataPoint(n,"n"));  
        dataPoints.add(new DataPoint(o,"o"));  
        dataPoints.add(new DataPoint(p,"p"));  
        dataPoints.add(new DataPoint(q,"q"));

        ClusterAnalysis ca=new ClusterAnalysis(3,0,dataPoints,2);  
       double[][] cen={{8,7},{8,6},{7,7}};  
       ca.startAnalysis(cen);

       ArrayList<DataPoint>[] v = ca.getClusterOutput();  
        for (int ii=0; ii<v.length; ii++){  
            ArrayList tempV = v[ii];  
            System.out.println("-----------Cluster"+ii+"---------");  
            Iterator iter = tempV.iterator();  
            while(iter.hasNext()){  
                DataPoint dpTemp = (DataPoint)iter.next();  
                System.out.println(dpTemp.getPointName());  
            }  
        }  
    }  
  
}