层次聚类算法

http://simg.sinajs.cn/blog7style/images/common/sg_trans.gif (2011-03-17 17:53:47)

[[http://simg.sinajs.cn/blog7style/images/common/sg_trans.gif](javascript:;)转载▼](javascript:;)

|  |  |
| --- | --- |
| 标签：  [杂谈](http://uni.sina.com.cn/c.php?t=blog&k=%D4%D3%CC%B8&ts=bpost&stype=tag) | 分类： [数据挖掘](http://blog.sina.com.cn/s/articlelist_1216541293_1_1.html) |

层次聚类算法：

前面介绍的K-means算法和K中心点算法都属于划分式（partitional）聚类算法。层次聚类算法是将所有的样本点自底向上合并组成一棵树或者自顶向下分裂成一棵树的过程，这两种方式分别称为凝聚和分裂。

凝聚层次算法:

初始阶段，将每个样本点分别当做其类簇，然后合并这些原子类簇直至达到预期的类簇数或者其他终止条件。

分裂层次算法:

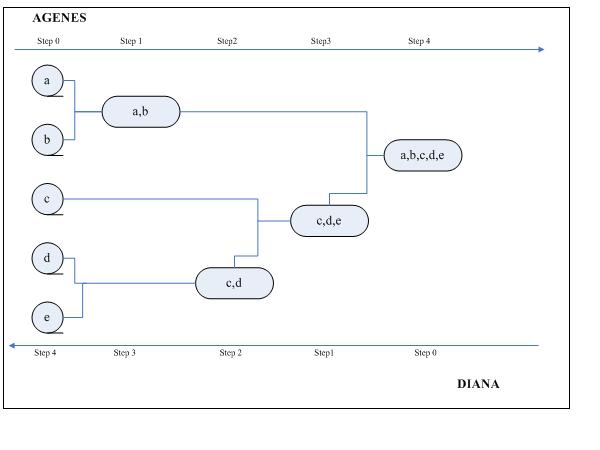
初始阶段，将所有的样本点当做同一类簇，然后分裂这个大类簇直至达到预期的类簇数或者其他终止条件。

两种算法的代表：

传统的凝聚层次聚类算法有AGENES，初始时，AGENES将每个样本点自为一簇，然后这些簇根据某种准则逐渐合并，例如，如果簇C1中的一个样本点和簇C2中的一个样本点之间的距离是所有不同类簇的样本点间欧几里得距离最近的，则认为簇C1和簇C2是相似可合并的。

传统的分裂层次聚类算法有DIANA，初始时DIANA将所有样本点归为同一类簇，然后根据某种准则进行逐渐分裂，例如类簇C中两个样本点A和B之间的距离是类簇C中所有样本点间距离最远的一对，那么样本点A和B将分裂成两个簇C1和C2，并且先前类簇C中其他样本点根据与A和B之间的距离，分别纳入到簇C1和C2中,例如，类簇C中样本点O与样本点A的欧几里得距离为2，与样本点B的欧几里得距离为4，因为Distance(A，O)<Distance(B,O)那么O将纳入到类簇C1中。

如图所示：

[](http://photo.blog.sina.com.cn/showpic.html#blogid=4882f26d0100pnrr&url=http://s12.sinaimg.cn/orignal/4882f26dg9eaaca6a335b)

算法：AGENES。传统凝聚层次聚类算法

输入：K：目标类簇数 D：样本点集合

输出：K个类簇集合

方法：1) 将D中每个样本点当做其类簇；

      2) repeat

      3)    找到分属两个不同类簇，且距离最近的样本点对；

      4)    将两个类簇合并；

      5) util 类簇数=K

算法：DIANA。传统分裂层次聚类算法

输入：K：目标类簇数 D：样本点集合

输出：K个类簇集合

方法：1) 将D中所有样本点归并成类簇；

      2) repeat

      3)    在同类簇中找到距离最远的样本点对；

      4)    以该样本点对为代表，将原类簇中的样本点重新分属到新类簇

      5) util 类簇数=K

缺点：

传统的层次聚类算法的效率比较低O(tn2) t:迭代次数 n:样本点数，最明显的一个缺点是不具有再分配能力，即如果样本点A在某次迭代过程中已经划分给类簇C1，那么在后面的迭代过程中A将永远属于类簇C1，这将影响聚类结果的准确性。

改进：

一般情况下，层次聚类通常和划分式聚类算法组合，这样既可以解决算法效率的问题，又能解决样本点再分配的问题，在后面将介绍BIRCH算法。首先把邻近样本点划分到微簇(microcluseters)中，然后对这些微簇使用K-means算法。

----------------贴上本人实现的AGENES算法，大家有兴趣可以把DIANA算法自己实现下---------------

package com.agenes;  
  
public class DataPoint {  
    String dataPointName; // 样本点名  
    Cluster cluster; // 样本点所属类簇  
    private double dimensioin[]; // 样本点的维度  
  
    public DataPoint(){  
  
    }  
  
    public DataPoint(double[] dimensioin,String dataPointName){  
         this.dataPointName=dataPointName;  
         this.dimensioin=dimensioin;  
    }  
  
    public double[] getDimensioin() {  
        return dimensioin;  
    }  
  
    public void setDimensioin(double[] dimensioin) {  
        this.dimensioin = dimensioin;  
    }  
  
    public Cluster getCluster() {  
        return cluster;  
    }  
  
    public void setCluster(Cluster cluster) {  
        this.cluster = cluster;  
    }  
  
    public String getDataPointName() {  
        return dataPointName;  
    }  
  
    public void setDataPointName(String dataPointName) {  
        this.dataPointName = dataPointName;  
    }  
}

package com.agenes;  
  
import java.util.ArrayList;  
import java.util.List;  
  
  
public class Cluster {  
    private List<DataPoint> dataPoints = new ArrayList<DataPoint>(); // 类簇中的样本点  
    private String clusterName;  
  
    public List<DataPoint> getDataPoints() {  
        return dataPoints;  
    }  
  
    public void setDataPoints(List<DataPoint> dataPoints) {  
        this.dataPoints = dataPoints;  
    }  
  
    public String getClusterName() {  
        return clusterName;  
    }  
  
    public void setClusterName(String clusterName) {  
        this.clusterName = clusterName;  
    }  
  
}

package com.agenes;  
  
import java.util.ArrayList;  
import java.util.List;  
  
  
public class ClusterAnalysis {  
   public List<Cluster> startAnalysis(List<DataPoint> dataPoints,int ClusterNum){  
      List<Cluster> finalClusters=new ArrayList<Cluster>();  
       
      List<Cluster> originalClusters=initialCluster(dataPoints);  
      finalClusters=originalClusters;  
      while(finalClusters.size()>ClusterNum){  
          double min=Double.MAX\_VALUE;  
          int mergeIndexA=0;  
          int mergeIndexB=0;  
          for(int i=0;i<finalClusters.size();i++){  
              for(int j=0;j<finalClusters.size();j++){  
                  if(i!=j){  
                      Cluster clusterA=finalClusters.get(i);  
                      Cluster clusterB=finalClusters.get(j);  
  
                      List<DataPoint> dataPointsA=clusterA.getDataPoints();  
                      List<DataPoint> dataPointsB=clusterB.getDataPoints();  
  
                      for(int m=0;m<dataPointsA.size();m++){  
                          for(int n=0;n<dataPointsB.size();n++){  
                              double tempDis=getDistance(dataPointsA.get(m),dataPointsB.get(n));  
                              if(tempDis<min){  
                                  min=tempDis;  
                                  mergeIndexA=i;  
                                  mergeIndexB=j;  
                              }  
                          }  
                      }  
                  }  
              } //end for j  
          }// end for i  
          //合并cluster[mergeIndexA]和cluster[mergeIndexB]  
          finalClusters=mergeCluster(finalClusters,mergeIndexA,mergeIndexB);  
      }//end while  
  
      return finalClusters;  
   }  
  
   private List<Cluster> mergeCluster(List<Cluster> clusters,int mergeIndexA,int mergeIndexB){  
        if (mergeIndexA != mergeIndexB) {  
            // 将cluster[mergeIndexB]中的DataPoint加入到 cluster[mergeIndexA]  
            Cluster clusterA = clusters.get(mergeIndexA);  
            Cluster clusterB = clusters.get(mergeIndexB);  
  
            List<DataPoint> dpA = clusterA.getDataPoints();  
            List<DataPoint> dpB = clusterB.getDataPoints();  
  
            for (DataPoint dp : dpB) {  
                DataPoint tempDp = new DataPoint();  
                tempDp.setDataPointName(dp.getDataPointName());  
                tempDp.setDimensioin(dp.getDimensioin());  
                tempDp.setCluster(clusterA);  
                dpA.add(tempDp);  
            }  
  
            clusterA.setDataPoints(dpA);  
  
            // List<Cluster> clusters中移除cluster[mergeIndexB]  
            clusters.remove(mergeIndexB);  
        }  
  
        return clusters;  
   }  
  
   // 初始化类簇  
   private List<Cluster> initialCluster(List<DataPoint> dataPoints){  
       List<Cluster> originalClusters=new ArrayList<Cluster>();  
       for(int i=0;i<dataPoints.size();i++){  
           DataPoint tempDataPoint=dataPoints.get(i);  
           List<DataPoint> tempDataPoints=new ArrayList<DataPoint>();  
           tempDataPoints.add(tempDataPoint);  
  
           Cluster tempCluster=new Cluster();  
           tempCluster.setClusterName("Cluster "+String.valueOf(i));  
           tempCluster.setDataPoints(tempDataPoints);  
  
           tempDataPoint.setCluster(tempCluster);  
           originalClusters.add(tempCluster);  
       }  
  
       return originalClusters;  
   }  
  
   //计算两个样本点之间的欧几里得距离  
   private double getDistance(DataPoint dpA,DataPoint dpB){  
        double distance=0;  
        double[] dimA = dpA.getDimensioin();  
        double[] dimB = dpB.getDimensioin();  
  
        if (dimA.length == dimB.length) {  
            for (int i = 0; i < dimA.length; i++) {  
                 double temp=Math.pow((dimA[i]-dimB[i]),2);  
                 distance=distance+temp;  
            }  
            distance=Math.pow(distance, 0.5);  
        }  
  
       return distance;  
   }  
  
   public static void main(String[] args){  
       ArrayList<DataPoint> dpoints = 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={8,20};  
  
       dpoints.add(new DataPoint(a,"a"));  
       dpoints.add(new DataPoint(b,"b"));  
       dpoints.add(new DataPoint(c,"c"));  
       dpoints.add(new DataPoint(d,"d"));  
       dpoints.add(new DataPoint(e,"e"));  
       dpoints.add(new DataPoint(f,"f"));  
  
       dpoints.add(new DataPoint(g,"g"));  
       dpoints.add(new DataPoint(h,"h"));  
       dpoints.add(new DataPoint(i,"i"));  
       dpoints.add(new DataPoint(j,"j"));  
       dpoints.add(new DataPoint(k,"k"));  
  
//       dataPoints.add(new DataPoint(l,"l"));  
  
       dpoints.add(new DataPoint(m,"m"));  
       dpoints.add(new DataPoint(n,"n"));  
       dpoints.add(new DataPoint(o,"o"));  
       dpoints.add(new DataPoint(p,"p"));  
       dpoints.add(new DataPoint(q,"q"));  
  
       int clusterNum=3; //类簇数  
  
       ClusterAnalysis ca=new ClusterAnalysis();  
       List<Cluster> clusters=ca.startAnalysis(dpoints, clusterNum);  
  
       for(Cluster cl:clusters){  
           System.out.println("------"+cl.getClusterName()+"------");  
           List<DataPoint> tempDps=cl.getDataPoints();  
           for(DataPoint tempdp:tempDps){  
               System.out.println(tempdp.getDataPointName());  
           }  
       }  
  
   }  
}