山东大学 软件 学院

模式识别技术 课程实验报告

|  |  |  |
| --- | --- | --- |
| 学号：201500301125 | 姓名： 马嫣然 | 班级： 15级1班 |
| 实验题目：K-means实验——算法设计与实现 | | |
| 实验目的：   * 编程实现K-means算法，并在红酒数据集上运行。 * 设置不同K值，不同初始中心，在红酒数据集上进行实验比较。 * 分析k-means的优缺点，并对其中一个或几个缺点进行改进。 * 演示实验并提交代码，统计分析实验结果并上交实验报告； | | |
| 实验步骤与内容：   1. 代码来源   代码是在参考开源代码的基础上做出的改进  Kmeans算法java代码 CSDN博客  <http://blog.csdn.net/jshayzf/article/details/22067855>   1. 算法设计说明 2. 实验环境：   硬件环境 个人笔记本电脑  软件环境 Java Eclipse  2.所用语言： Java   1. 实验数据分析：   **红酒数据集（Wine Data Set）<http://archive.ics.uci.edu/ml/datasets/Wine>**  共178个数据，每个数据特征为13维  13个特征分别为：（13个化学成分，每个成分取值为实数）  1) Alcohol  2) Malic acid  3) Ash  4) Alcalinity of ash  5) Magnesium  6) Total phenols  7) Flavanoids  8) Nonflavanoid phenols  9) Proanthocyanins  10) Color intensity  11) Hue  12) OD280/OD315 of diluted wines  13) Proline    给定的数据集有十四列，第一列是类别，应该排除掉，使用后十三列作为属性   1. 算法设计：   **（1）思路**  K-means算法的基本思想是：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。通过迭代的方法，逐次更新各聚类中心的值，直至得到最好的聚类结果。  假设要把样本集分为c个类别，算法描述如下：  （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（2）（3）的迭代法更新后，值保持不变，则迭代结束，否则继续迭代。   1. **具体实现** 2. **加载数据**   BufferedReader br=**new** BufferedReader(**new** InputStreamReader(**new** FileInputStream("src/K\_means/Wine dataset.txt")));  String data = **null**;  List<ArrayList<Double>> dataList = **new** ArrayList<ArrayList<Double>>();  **while**((data=br.readLine())!=**null**){  //System.out.println(data);  String []fields = data.split(",");  List<Double> tmpList = **new** ArrayList<Double>();  **for**(**int** i=0; i<fields.length;i++)  tmpList.add(Double.*parseDouble*(fields[i]));  dataList.add((ArrayList<Double>) tmpList);  }  br.close();   1. **随机确定K个初始聚类中心**   Random rd = **new** Random();  **int** k=3;  **int** [] initIndex={59,71,48};  **int** [] helpIndex = {0,59,130};  **int** [] givenIndex = {0,1,2};  System.***out***.println("random centers' index");  **for**(**int** i=0;i<k;i++){  **int** index = rd.nextInt(initIndex[i]) + helpIndex[i];  //int index = givenIndex[i];  System.***out***.println("index "+index);  centers.add(dataList.get(index));  helpCenterList.add(**new** ArrayList<ArrayList<Double>>());  }   1. **把每个样本归入距离最短的中心所在的类**   **for**(**int** i=0;i<dataList.size();i++){//标注每一条记录所属于的中心  **double** minDistance=99999999;  **int** centerIndex=-1;  **for**(**int** j=0;j<k;j++){//离0~k之间哪个中心最近  **double** currentDistance=0;  **for**(**int** t=1;t<centers.get(0).size();t++){//计算两点之间的欧式距离  currentDistance += ((centers.get(j).get(t)-dataList.get(i).get(t))/(centers.get(j).get(t)+dataList.get(i).get(t))) \* ((centers.get(j).get(t)-dataList.get(i).get(t))/(centers.get(j).get(t)+dataList.get(i).get(t)));  }  **if**(minDistance>currentDistance){  minDistance=currentDistance;  centerIndex=j;  }  }  helpCenterList.get(centerIndex).add(dataList.get(i));  }   1. **计算新的k个聚类中心并更新值**   **for**(**int** i=0;i<k;i++){    ArrayList<Double> tmp = **new** ArrayList<Double>();    **for**(**int** j=0;j<centers.get(0).size();j++){  **double** sum=0;  **for**(**int** t=0;t<helpCenterList.get(i).size();t++)  sum+=helpCenterList.get(i).get(t).get(j);  tmp.add(sum/helpCenterList.get(i).size());  }    newCenters.add(tmp);    }   1. **重复cd后如果值不变，则迭代结束**   //计算新旧中心之间的距离，当距离小于阈值时，聚类算法结束  **double** distance=0;    **for**(**int** i=0;i<k;i++){  **for**(**int** j=1;j<centers.get(0).size();j++){//计算两点之间的欧式距离  distance += ((centers.get(i).get(j)-newCenters.get(i).get(j))/(centers.get(i).get(j)+newCenters.get(i).get(j))) \* ((centers.get(i).get(j)-newCenters.get(i).get(j))/(centers.get(i).get(j)+newCenters.get(i).get(j)));  }  //System.out.println(i+" "+distance);  }  System.***out***.println("\ndistance: "+distance+"\n\n");  **if**(distance==0)//小于阈值时，结束循环  **break**;  **else**//否则，新的中心来代替旧的中心，进行下一轮迭代  {  centers = **new** ArrayList<ArrayList<Double>>(newCenters);  //System.out.println(newCenters);  newCenters = **new** ArrayList<ArrayList<Double>>();  helpCenterList = **new** ArrayList<ArrayList<ArrayList<Double>>>();  helpCenterList=*initHelpCenterList*(helpCenterList,k);  }  **（3）改进**  添加计算准确率的输出  a.**int** match=0,total=0;  **float** rate=0;  b.读入数据集时，计算total  c.若预测结果与数据第一列相同，则增加match，即估算正确数  **if**(helpCenterList.get(i).get(j).get(0)==(i+1))  match++;  d.输出概率  rate=(**float**)match/total;  rate=rate\*10000/100;  System.***out***.println("总测试数："+total);  System.***out***.println("正确数："+match);  System.***out***.println("正确率："+rate+ "%");   1. 实验结果：   初始中心  **int** [] initIndex={59,71,48,66,71};  **int** [] helpIndex = {0,59,130,53,43};  **int** index = rd.nextInt(initIndex[i]) + helpIndex[i];  要求：改变K的值，比较结果。   |  |  |  | | --- | --- | --- | | K值 | 迭代次数 | 正确率 | | 1 | 2 | 33.14607% | | 2 | 12 | 35.955055% | | 3 | 5 | 93.25843% | | 4 | 8 | 82.02247% | | 5 | 7 | 61.79775% |   同一个K值可能运行结果、迭代次数、准确率也有不同，有时候还会无限迭代。   1. 实验结果分析：   a.初始质心是随机选取的，这样簇的质量往往会很差。  b.有可能导致算法收敛很慢  c.K值为3的时候正确率最高，因原数据集就分成了3类  d.K-means的优缺点  参考网络 Kmeans算法的优缺点——CSDN博客  <http://blog.csdn.net/gaobellen/article/details/45024663>  优点：1.先，算法能根据较少的已知聚类样本的类别对树进行剪枝确定部分样本的分类；  2.为克服少量样本聚类的不准确性，该算法本身具有优化迭代功能，在已经求得的聚类上再次进行迭代修正剪枝确定部分样本的聚类，优化了初始监督学习样本分类不合理的地方；  3.由于只是针对部分小样本可以降低总的聚类时间复杂度。  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；  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，一旦初始值选择的不好，可能无法得到有效的聚类结果；  3.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。  e.改进  1.优化初始化随机质心的方法  2.减少不必要的距离的计算，不要每次都计算所有样本到所有质心的距离  3）实验结果截图  1  2  3  4  5 | | |