《数据仓库与数据挖掘》课程实验报告

题目： **K-medoids聚类算法程序实现**

姓名： 徐哲然

学号： **1352334**

专业： 信息安全

一.实验背景

k-medoids聚类算法，即k-中心聚类算法，它是基于k-means聚类算法的改进。我们知道，k-means算法执行过程，首先需要随机选择初始质心，只有第一次随机选择的初始质心才是实际待聚类点集中的点，而后续将非质心点指派到对应的质心点后，重新计算得到的质心并非是待聚类点集中的点，而且如果某些非质心点是离群点的话，导致重新计算得到的质心可能偏离整个簇，为了解决这个问题，提出了改进的k-medoids聚类算法。

二、实验原理

k-medoids聚类算法也是通过划分的方式来计算得到聚类结果，它使用绝对差值和（Sum of Absolute Differences，SAD）的度量来衡量聚类结果的优劣，在n维欧几里德空间中，计算SAD的公式如下所示：

$$SAD-\sum_{m=1}^{k}\sum_{p_i\in C_i} dist(p_i,o_i)-\sum_{m=1}^{k}\sum_{p_i\in C_i}\sqrt{\sum_{j=1}^{n_{C_i}}(p_{ij}-o_{ij})^2}$$

围绕中心点划分（Partitioning Around Medoids，PAM）的方法是比较常用的，使用PAM方法进行处理，可以指定一个最大迭代次数的参数，在迭代过程中基于贪心策略来选择使得聚类的质量最高的划分。使用PAM的方法处理，每次交换一个中心点和非中心点，然后执行将非中心点指派到最近的中心点，计算得到的SAD值越小，则聚类质量越好，如此不断地迭代，直到找到一个最好的划分。

维基百科上给出的基于PAM方法计算聚类的过程，描述如下：

从待聚类的数据点集中随机选择k个点，作为初始中心点；

将待聚类的数据点集中的点，指派到最近的中心点；

进入迭代，直到聚类的质量满足指定的阈值（可以通过计算SAD），使总代价减少：

对每一个中心点o，对每一个非中心点p，执行如下计算步骤：

交换点o和p，重新计算交换后的该划分所生成的代价值；

如果本次交换造成代价增加，则取消交换。

上面算法描述，应该是按顺序的取遍中心点集合中的点，也从非中心点集合中取遍所有非中心点，分别计算生成的新划分的代价。由于待聚类的点集可大可小，我们可以考虑，每次取点的时候，采用随机取点的策略，随机性越强越好，只要满足最终迭代终止的条件即可。通常，如果能够迭代所有情况，那么最终得到的划分一定是最优的划分，即聚类结果最好，这通常适用于聚类比较小的点的集合。但是如果待聚类的点的集合比较大，则需要通过限制迭代次数来终止迭代计算，从而得到一个能够满足实际精度需要的聚类结果。我们在下面实现k-medoids聚类算法，分别随机选择中心点和非中心点，对他们进行交换，通过设置允许最大迭代次数（maxIterations）这个参数值，来使聚类计算最后停止。

聚类算法实现

首先，为了便于理解后面的代码实现，我们描述一下代码实现聚类过程的基本步骤，如下所示：

输入待聚类点集，以及参数k、maxIterations、parallism；

同k-means算法一样，随机选择初始中心点集合；

启动parallism个线程，用来将非中心点指派给最近的中心点；

开始执行迭代，使得聚类结果对应的划分的SAD值最小：

将非中心点，基于Round-Robin策略，分配给多个线程，并行指派：将非中心点指派给距离其最近的中心点；

将多个线程指派的局部结果进行合并，得到一个全局的指派结果；

根据指派结果计算SAD值：如果是第一次进行指派，直接计算其SAD值，保存在previousSAD变量中，该变量保存的是最小的SAD值，第一次初始化第一次指派结果计算得到的SAD值；如果不是第一次进行指派，也计算SAD值，将SAD值保存在变量currentSAD中，继续执行步骤8；

随机选择一个非中心点；

创建一个ClusterHolder对象，该对象保存了该轮迭代指派结果，根据随机选择的非中心点修改ClusterHolder对象中的结果，将随机选择非中心点和对应的中心点进行交换，为下一轮指派过程准备数据；

最后，判断是否达到指定的最大迭代次数，如果达到则终止计算，处理最终聚类结果，否则执行下一轮迭代计算，转步骤5。

我们实现的k-medoids聚类算法，需要指定2个聚类相关参数，另外一个参数是程序计算并行度，可以通过构造方法看到。

三.代码实现

1.实验环境：windows 8 64x. Visual studio 2013

实验语言：C++

2.核心代码如下所示：

头文件：

```
#ifndef _MEDOIDES_H_
#define _MEDOIDES_H_

#include <iostream>
#include <vector>
#include <string>
#include <set>
using namespace std;

class medoids
{
public:
	void initial(string fileName, int k);
	void distribute();
	bool findNewCenter();
	void print();
private:
	void getData(string fileName);
	int closet(int num);
	double calCost(set<double>& s);
	vector<double> data;
	int clusterNum;
	vector<double> center;
	vector<set<double> > cluster;

};


#endif
```

源文件：

```
#include<fstream>
#include<cmath>
#include"medoids.h"
#include"stdafx.h"
using namespace std;

void medoids::initial(string fileName, int k)
```

```
{
	getData(fileName);
	if (data.size()<k)
		clusterNum = data.size();
	else
		clusterNum = k;
	for (int i = 0; i<clusterNum; i++)
		center.push_back(data[i]);
	cluster.resize(clusterNum);
	int s = cluster.size();
}

void medoids::getData(string file)
{
	ifstream in(file.c_str(), ios::in);
	if (!in.is_open())
	{
		cout << "open file failed!";
		exit(1);
	}
	int size;
	in >> size;
	for (int i = 0; i<size; i++)
	{
		double temp;
		in >> temp;
		data.push_back(temp);
	}
	in.close();
}

void medoids::distribute()
{
	cluster.clear();
	cluster.resize(clusterNum);
	int s = cluster.size();
	for (int i = 0; i<data.size(); i++)
	{
		int pos = closet(data[i]);
		cluster[pos].insert(data[i]);
	}
}

int medoids::closet(int num)
{
	int pos = -1;
	double min = 9.9e+20;
	for (int i = 0; i<center.size(); i++)
		if (min>fabs(center[i] - num))
		{
			min = fabs(center[i] - num);
			pos = i;
		}
	return pos;
}

bool medoids::findNewCenter()
{
	bool again = false;
	for (int i = 0; i<cluster.size(); i++)
	{
		double newCenter = calCost(cluster[i]);
```

```
            if (newCenter != center[i])
            {
                center[i] = newCenter;
                again = true;
            }
        }
        return again;
}

double medoids::calCost(set<double>& s)
{
        set<double>::iterator ite = s.begin();
        int minCost = 9.9e+20;
        double newCenter = -1;
        while (ite != s.end())
        {
            set<double>::iterator ite1 = s.begin();
            double total = 0;

            while (ite1 != s.end())
            {
                double delta = (*ite1) - (*ite);
                total = total + delta*delta;
                ite1++;
            }

            if (total<minCost)
            {
                minCost = total;
                newCenter = (*ite);
            }
            ite++;
        }
        return newCenter;
}

void medoids::print()
{
        ofstream cout("medoids.txt", ios::out);
        set<double>::iterator ite;
        for (int i = 0; i<clusterNum; i++)
        {
            cout << "cluster " << i << endl;
            cout << "medoids " << center[i] << endl;
            ite = cluster[i].begin();
            while (ite != cluster[i].end())
            {
                cout << (*ite) << ' ';
                ite++;
            }
            cout << endl << endl << endl;
        }
        //cout.close();
}
```