

实验三：Canopy 和 KMeans 聚类实验

1. Canopy 算法聚类实验

1.1 Canopy 算法原理及目标

通过计算数据集中两两数据项的相似度，将相似度满足一定阈值（通常取相似度的平均值）的数据项聚合到一起，形成类，最后所有的数据项聚合成若干类，类内的对象之间相似度高，而类之间的对象相似度低。类内相似度越高，类间相似度越低，则聚类效果越好。

例如：给出如图 1 所示的一个新闻标题文本的数据集，每个数据项为一个标题，本实验的实验目标就是将这些标题聚类。

习近平出席纪念红军长征胜利80周年大会
长征胜利80周年 习近平讲话
中国游客在日本顺走马桶盖 当事人向单位提出辞职
李克强会见菲律宾总统杜特尔特
李克强会见杜特尔特：南海问题不是中菲关系全部
最高检:凡暴力伤医案一律列为重大敏感案件
黑龙江省长在北京会见刘士余：希望得到证监会支持
赴日游客带走酒店马桶盖引热议
李克强会见全国先进个体工商户代表并讲话
外交部回应安倍将访珍珠港：中方有很多场所可开放
安倍将访珍珠港 外交部:中国有很多场所可供日方凭吊
.....

如何将这些标题聚类？

图 1 聚类目标图

聚类的结果如图 2 所示，把相似度高的标题文本聚到一起形成一类，最后形成若干个类。

习近平出席纪念红军长征胜利80周年大会
长征胜利80周年 习近平讲话

中国游客在日本顺走马桶盖 当事人向单位提出辞职
赴日游客带走酒店马桶盖引热议

李克强会见菲律宾总统杜特尔特
李克强会见杜特尔特：南海问题不是中菲关系全部

李克强会见全国先进个体工商户代表并讲话

外交部回应安倍将访珍珠港：中方有很多场所可开放
安倍将访珍珠港 外交部:中国有很多场所可供日方凭吊

.....

图 2 聚类结果图

1.2 Canopy 算法流程

经典 Canopy 算法流程图如下：

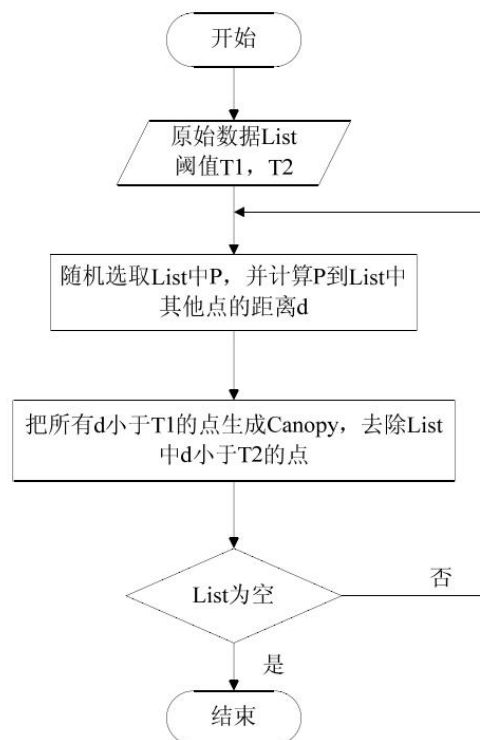


图 3 Canopy 算法流程图

本实验是经典 **Canopy** 算法的简化版，只采用一个阈值 T ，而两点之间的距离计算用两个数据项的相似度计算来表示。相似度越高的两个数据项代表距离越近。实验具体过程如下：

- (1) 初始化：输入文本集合，文本向量化后的向量集合；
- (2) 根据向量集合计算两两向量的相似度的平均值，作为相似度阈值 T ；
- (3) 取向量集合第一个向量，创建一个类，将该向量对应索引值加入该类，并将该类加入类集合；
- (4) 取一个向量集合中的向量 v （从第二个开始）；
- (5) 计算向量 v 与类集合中所有类的相似度，若相似度大于阈值 T ，则将向量 v 对应索引值加入该类，并退出，转到 (4)；
- (6) 当向量 v 与类集合中所有类的相似度都不大于阈值 T ，则创建一个新类，将 v 对应索引值加入该类，并将该类加入类集合；
- (7) 重复 (4) (5) (6) 步，直到向量集合所有向量都被取到为止，聚类结束；
- (8) 根据类集合中存放的文本索引值生成对应的文本类集合，并将各个不同的类中的数据写入文件保存，或者显示数据。

1.3 Canopy 算法实现

算法实现语言：Java 版本：jdk1.8

开发工具：Eclipse 版本： Mars.2 Release (4.5.2)

算法主要代码如下图所示：

```

/**
 * Canopy聚类算法
 * @author tankai
 *
 */
public class Canopy {
    //原始文本对应向量
    private List<double[]> vectors;

    //聚类结果对应的下标
    private List<List<Integer>> resultIndex;

    //Canopy初始阈值
    private double T = 0f;

    //聚类结果类别数量
    private int canopy = 0;

    //用于相似度计算的类
    private CosDistance cosDistance;
}

```

上图所示为 Canopy 类以及其中的属性,编写代码时注意自行生成所有属性的 get() 和 set()方法。

```

//canopy算法
public void cluster(){
    //初始化cosDistance类
    cosDistance = new CosDistance(vectors);
    setT(cosDistance.getThreshold());
    //初始化聚类结果集
    resultIndex = new ArrayList<List<Integer>>();
    List<Integer> tmpIndex = null;
    //遍历向量集合
    for(int i = 0 ; i < vectors.size() ; i++){
        //i = 0 一个类都没有时，直接添加进resultIndex。
        if(i == 0){
            //把第1个向量的索引添加到tmpIndex，作为第一个类
            tmpIndex = new ArrayList<Integer>();
            tmpIndex.add(i);
            //把第一个类加入到resultIndex
            resultIndex.add(tmpIndex);
            continue;
        }
        //找到符合相似度要求的类的标志
        boolean isFind = false;

        for(int j = 0 ; j < resultIndex.size() ; j++){
            //计算向量与已分的类的向量平均值是否大于阈值，大于则添加到该类
            if(cosDistance.getDistance(i, resultIndex.get(j)) > T){
                //获取待比较的那个类的所有元素的索引，存放在tmpIndex
                tmpIndex = resultIndex.get(j);

                //把i(对应向量的索引值)加入到tmpIndex
                tmpIndex.add(i);

                //从resultIndex里删除旧的待比较的那个类
                resultIndex.remove(j);

                //添加加入了i的新类到resultIndex
                resultIndex.add(tmpIndex);
                //已加入相似类，退出当前循环
                isFind = true;
                break;
            }
        }
        //与前面的类相似度都不符合要求则新建一个类
        if(!isFind){
            tmpIndex = new ArrayList<Integer>();
            tmpIndex.add(i);
            resultIndex.add(tmpIndex);
        }
    }
    //获取聚类的数量
    canopy = resultIndex.size();
}

```

上图所示为 Canopy 类中实现聚类功能的 cluster()函数的代码。可根据 1.1 和 1.2 节 Canopy 算法原理和流程自行编写聚类程序，或者在以上代码基础上进行优化。

```

public class CanopyTest{

    public static void main(String[] args) {

        //读取原始数据
        List<String> dataList = ExcelReader.read("E:\\四川项目\\测试数据\\qianchengwuyou.xlsx",1);

        // ClusterUtil.showDataList(dataList);
        //分词
        AnsjSegmentation ansj = new AnsjSegmentation();
        ansj.setWordList(dataList);
        ansj.segment();

        //得到分词后的List集合
        List<String[]> seglist = ansj.getSegList();
        // ClusterUtil.showSeglist(seglist);

        //向量转换
        TFIDFConverter converter = new TFIDFConverter(seglist);
        List<double[]> vectors = converter.getVector();
        Canopy canopy = new Canopy();
        canopy.setVectors(vectors);

        //进行Canopy聚类
        canopy.cluster();

        //查看canopy阈值
        System.out.println("计算的平均阈值: "+canopy.getT());

        //聚类结果显示到控制台
        ClusterUtil.showResult(canopy.getResultIndex(), dataList);
        System.out.println("聚类个数: "+canopy.getCanopy());

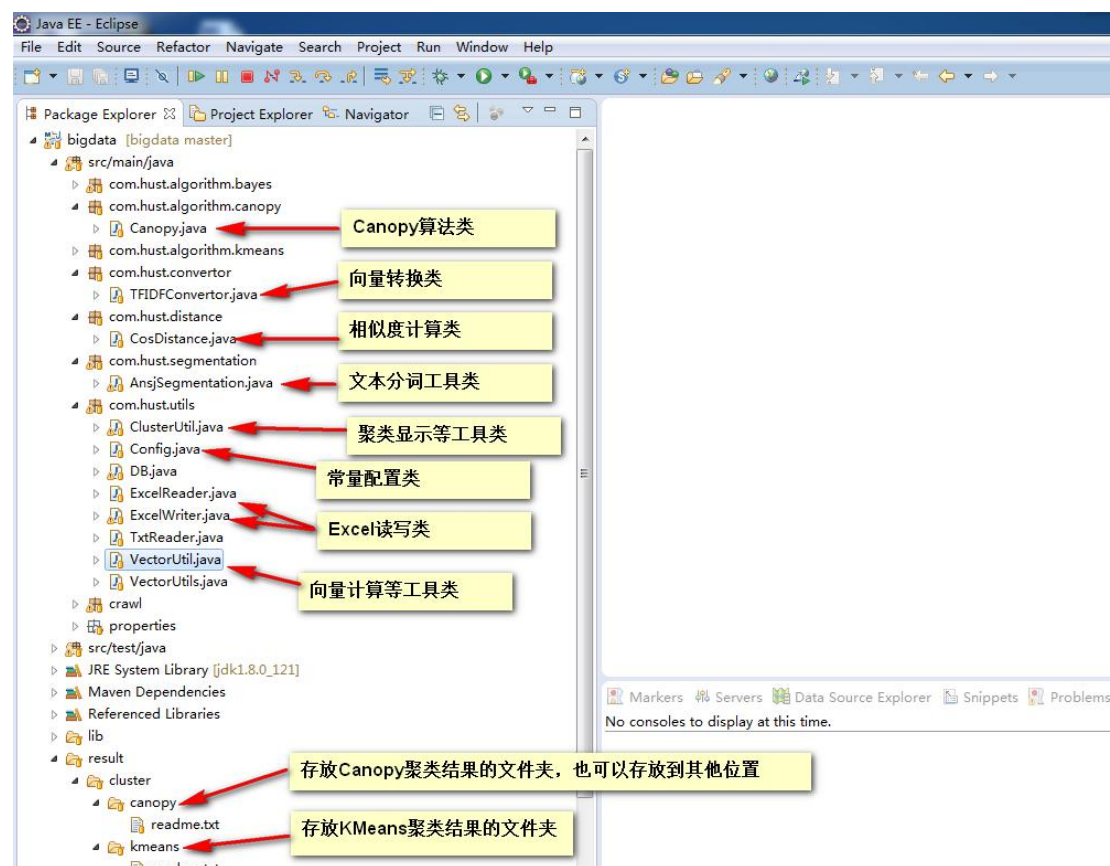
        //聚类结果写入到文件
        List<List<String>> clusterlist = ClusterUtil.getClusters(canopy.getResultIndex(), dataList);
        //把每个类的结果输出到一个Excel文件
        for(int i = 0 ; i < clusterlist.size() ; i++){
            //
            ExcelWriter.collistToExcel(Config.CANOPY_RESULT_PATH+
                clusterlist.get(i).get(0)+".xlsx", clusterlist.get(i));
        }
    }
}

```

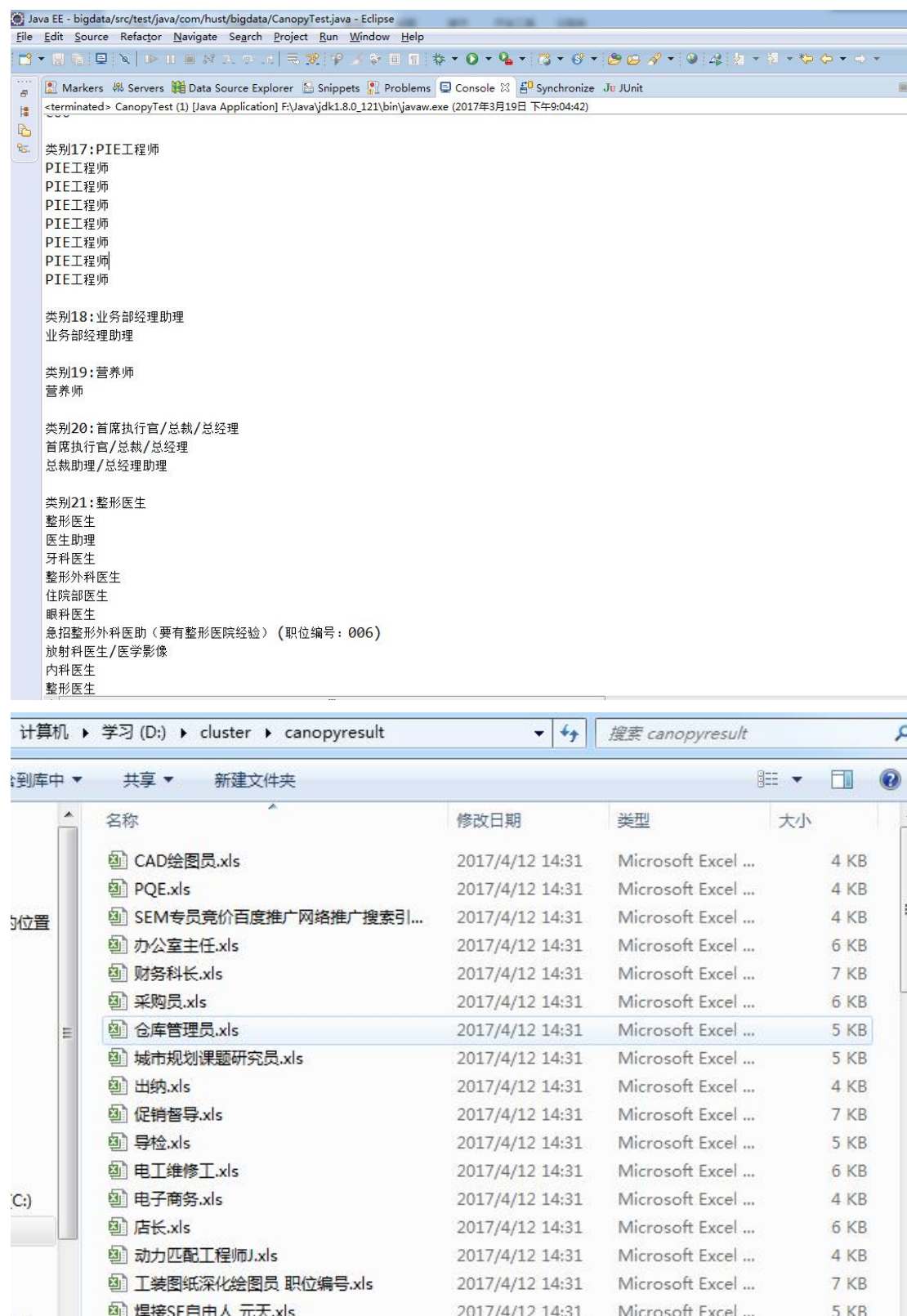
上图所示为 Canopy 算法测试程序，包括如下几步：

- (1) 从 Excel 文件读取待聚类的数据，保存到 List<String> dataList;
- (2) 将 List<String>文本集合进行分词，保存到 List<String[][]> seglist;
- (3) 对分词后的文本进行向量转换，保存到 List<double[]> vectors;
- (4) 初始化 Canopy 聚类参数，进行聚类；
- (5) 将聚类结果保存到 Excel 文件。

注：程序用到的工具类在实验前会由助教给出，其名字和功能如下图所示：



Canopy 算法运行结果打印到控制台，和写入到 Excel 文件效果图如下图所示：



1.4 Canopy 实验要求

- (1) 理解 Canopy 算法原理，根据算法流程图和示例代码自己编写 Canopy 程序；

- (2) 思考自动确定 Canopy 相似度阈值的方法，写出计算方法的代码并运用到 Canopy 聚类里；
- (3) 编写 Canopy 的测试程序，回顾前面的实验，实现读取实验一爬取的数据、进行分词、去停用词、向量转化、Canopy 聚类等过程；
- (4) 运行 Canopy 测试程序，能在控制台输出聚类好的数据；
- (5) 将聚类后的数据保存到 Excel 文件。

2. KMeans 算法聚类

2.1 KMeans 算法原理及目标

K-means 算法是最为经典的基于划分的聚类方法，是十大经典数据挖掘算法之一。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。KMeans 算法目标是在 Canopy 算法聚类的个数基础上继续更精确地聚类。

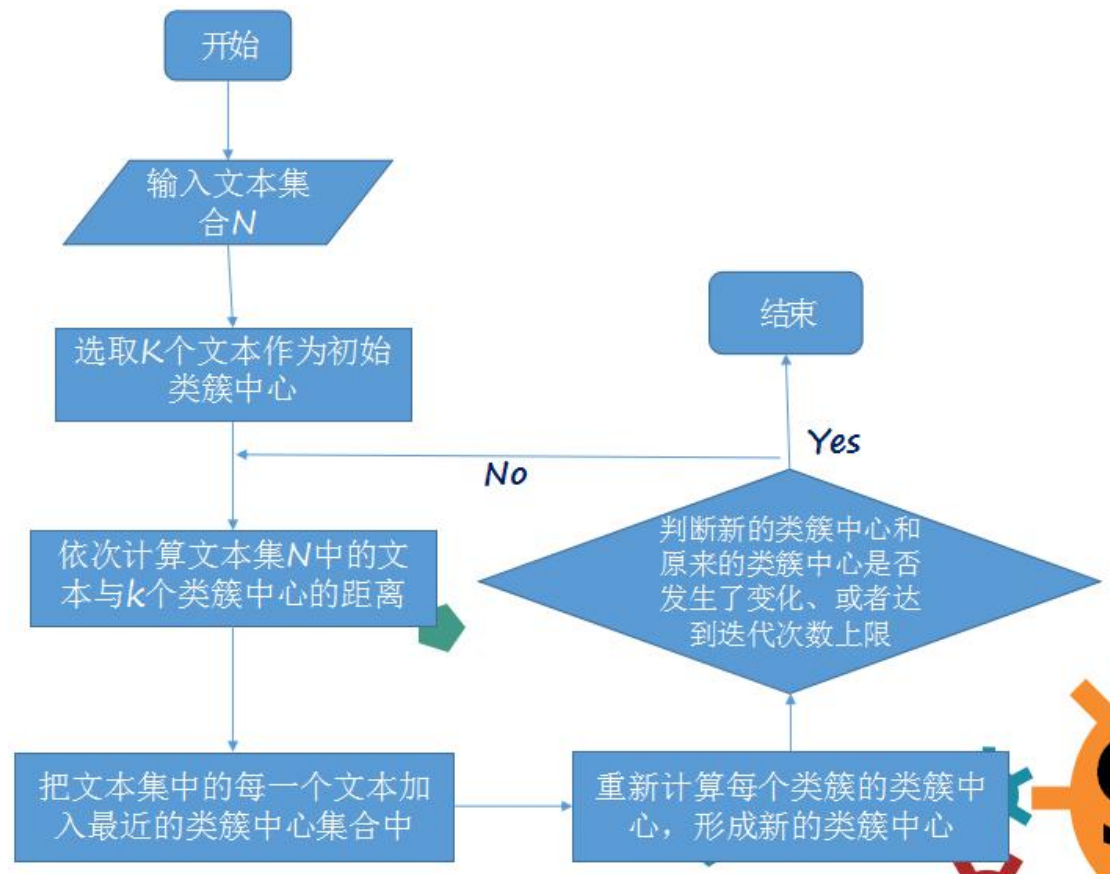
假设要把样本集分为 c 个类别，算法描述如下：

- (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。

该算法的最大优势在于简洁和快速。算法的关键在于初始中心的选择和距离公式。

2.2 KMeans 算法流程

本实验是基于 Canopy 的 KMeans 算法，Canopy 的聚类结果数作为 KMeans 的 K 值，Canopy 流程参照上一节。KMeans 流程图如下所示：



2.3 KMeans 算法实现

算法实现语言：Java 版本：jdk1.8

开发工具：Eclipse 版本： Mars.2 Release (4.5.2)

算法主要代码如下图所示：

```

public class KMeans {
    //原始文本对应向量
    private List<double[]> vectors;

    //聚类结果对应的下标
    private List<List<Integer>> resultIndex;

    //K个聚类中心点
    private List<double[]> centers;

    private List<double[]> newcenters;

    //KMeans初始K值
    private int K = 0;

    //迭代次数
    private int iterTimes = 10;

    //用于相似度计算的类
    private CosDistance cosDistance;

    //构造函数，进行初始化
    public KMeans(int k , List<double[]> vectors, int times){
        //初始化K
        setK(k);
        setVectors(vectors);
        setIterTimes(times);

        centers = new ArrayList<>();
        resultIndex = new ArrayList<>();

        cosDistance = new CosDistance(vectors);
    }
}

```

上图所示为 KMeans 类中的属性及构造函数，编写代码时注意自行生成所有属性的 get()和 set()方法。

```

//初始化K个初始聚类中心点--随机选择K个向量
public void initClusters(){
    //得到随机数
    Vector<Integer> vecIndex = random(K,vectors.size());
    for(int i = 0 ; i < K ; i++){
        //生成的第i个随机数对应的向量作为第i个中心
        centers.add(vectors.get(vecIndex.get(i)));
        //存放下标
        List<Integer> cluster = new ArrayList<>();
        resultIndex.add(cluster);
    }
}

```

上图所示为 KMeans 类中的初始化聚类中心点函数。

```

/**
 * KMeans聚类
 */
public void cluster(){
    //初始化聚类中心点、结果集
    initClusters();

    //记录最大的相似度
    double maxSim = 0f;
    int tmpIndex = 0;
    //新中心点
    newcenters = centers;
    //开始迭代
    while(iterTimes > 0){
        iterTimes--;
        //遍历所有向量,
        for(int i = 0 ; i < vectors.size() ; i++){
            double[] vector = vectors.get(i);
            //相似度置0
            maxSim = 0;
            //依次计算文本向量集中的向量与k个类簇中心的距离
            for(int j = 0 ; j < K ; j++){
                if(cosDistance.calculate(vector, newcenters.get(j)) > maxSim){
                    maxSim = cosDistance.calculate(vector, newcenters.get(j));
                    tmpIndex = j;
                }
            }
            //把文本向量集中的每一个向量对应的索引加入最近的类簇中心集合中
            resultIndex.get(tmpIndex).add(i);
        }
        //重新计算每个类簇的类簇中心
        for(int i = 0 ; i < resultIndex.size() ; i++){
            List<Integer> cluster = resultIndex.get(i);
            double[] sum = new double[vectors.get(0).length];
            for(int j : cluster){
                sum = VectorUtil.add(sum, vectors.get(j));
            }
            newcenters.set(i, VectorUtil.center(sum,cluster.size()));
        }
        //判断新簇中心相对于旧中心移动的距离是否在条件内
        if(!centerMove(centers, newcenters)){
            centers = newcenters;
            break;
        }
        //形成新的类簇中心
        centers = newcenters;
        //清除掉结果集里的数据,继续迭代
        resultIndex.clear();
    }
}

```

上图所示为 KMeans 类中的关键代码 cluter()聚类函数。可根据 KMeans 算法原理和流程自行编写聚类程序，或者在以上代码基础上进行优化。

```

/**
 * 判断聚类中心点是否移动
 * 新中心和旧中心相似度小于T=95%. return true
 * 新中心和旧中心相似度大于等于T=95%. return false
 * @param centers
 * @param newcenters
 * @return true--移动
 */
private boolean centerMove(List<double[]> centers, List<double[]> newcenters) {
    // TODO Auto-generated method stub
    double T = 0.95f;

    double sum = 0f;
    for(int i = 0 ; i < K ; i++){
        sum += cosDistance.caculate(centers.get(i), newcenters.get(i));
    }
    if(sum/centers.size() < T){
        System.out.println(sum/centers.size());
        return true;
    }
    return false;
}

/**
 * 产生一组随机数作为K个聚类中心的下标
 * @param K
 * @param size 向量总数
 * @return
 */
private Vector<Integer> random(int K, int size) {
    //创建一个产生随机数的对象
    Random r = new Random();
    //创建一个存储随机数的集合
    Vector<Integer> v = new Vector<Integer>();
    //定义一个统计变量
    int count = 0;
    while(count < K){
        int number = r.nextInt(size);
        //判断number是否在集合中存在
        if(!v.contains(number)){
            //不在集合中,就添加
            v.add(number);
            count++;
        }
    }
    return v;
}

```

上图所示 centerMove()函数为 KMeans 类中判断中心点是否发生移动的函数，当中心点不发生移动时或迭代次数达到要求则完成聚类；

random()函数为随机生成 K 个随机数的函数。

```

public class KMeansTest{

    public static void main(String[] args) {

        //读取原始数据
        List<String> dataList = ExcelReader.read("E:\\四川项目\\测试数据\\qianchengwuyou.xlsx",1);

        // ClusterUtil.showDataList(dataList);
        //分词
        AnsjSegmentation ansj = new AnsjSegmentation();
        ansj.setWordList(dataList);
        ansj.segment();

        //得到分词后的List集合
        List<String[]> seglist = ansj.getSegList();
        // ClusterUtil.showSeglist(seglist);

        //向量转换
        TFIDFConvertor convertor = new TFIDFConvertor(seglist);
        List<double[]> vectors = convertor.getVector();
        Canopy canopy = new Canopy();
        canopy.setVectors(vectors);

        //进行Canopy聚类
        canopy.cluster();

        //查看canopy阈值
        System.out.println("计算的平均阈值: "+canopy.getT());

        //聚类结果显示到控制台
        ClusterUtil.showResult(canopy.getResultIndex(), dataList);
        System.out.println("聚类个数: "+canopy.getCanopy());

        //初始化KMeans聚类参数 (K值--Canopy聚类的个数, 向量集合, 迭代次数)
        KMeans kmeans = new KMeans(canopy.getCanopy(), vectors, 10);

        //进行KMeans聚类
        kmeans.cluster();
        //聚类结果显示到控制台
        ClusterUtil.showResult(kmeans.getResultIndex(), dataList);

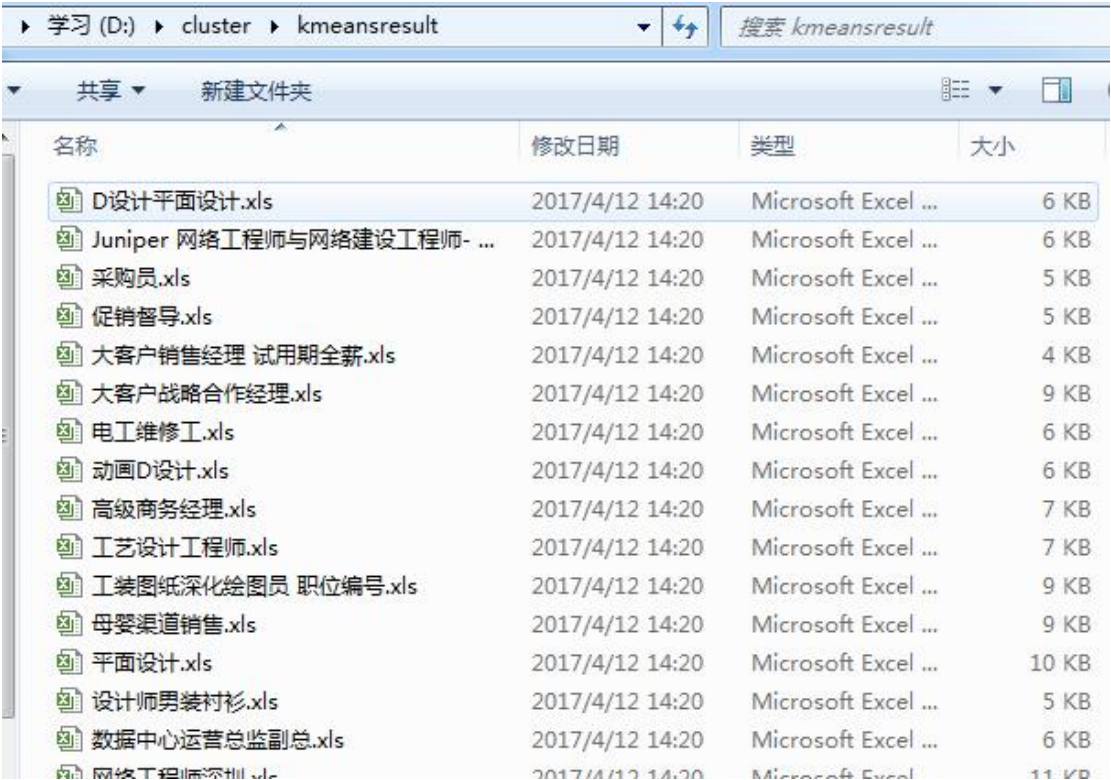
        //聚类结果写入到文件
        List<List<String>> clusterlist = ClusterUtil.getClusters(kmeans.getResultIndex(), dataList);
        //把每个类的结果输出到一个Excel文件
        for(int i = 0 ; i < clusterlist.size() ; i++){
            //
            ExcelWriter.colListToExcel(Config.KMEANS_RESULT_PATH+
                clusterlist.get(i).get(0)+".xlsx", clusterlist.get(i));
        }
    }
}

```

上图所示为 KMeans 算法测试程序，包括如下几步：

- (1) 从 Excel 文件读取待聚类的数据，保存到 List<String> dataList;
- (2) 将 List<String>文本集合进行分词，保存到 List<String[][]> seglist;
- (3) 对分词后的文本进行向量转换，保存到 List<double[]> vectors;
- (4) 初始化 Canopy 聚类的向量参数，进行 Canopy 聚类;
- (5) 初始化 KMeans 的 K 值、向量等参数，进行 KMeans 聚类;
- (6) 将 KMeans 聚类结果保存到 Excel 文件。

KMeans 算法运行结果打印到控制台，和写入到 Excel 文件效果图如下图所示：



名称	修改日期	类型	大小
D设计平面设计.xls	2017/4/12 14:20	Microsoft Excel ...	6 KB
Juniper 网络工程师与网络建设工程师- ...	2017/4/12 14:20	Microsoft Excel ...	6 KB
采购员.xls	2017/4/12 14:20	Microsoft Excel ...	5 KB
促销督导.xls	2017/4/12 14:20	Microsoft Excel ...	5 KB
大客户销售经理 试用期全薪.xls	2017/4/12 14:20	Microsoft Excel ...	4 KB
大客户战略合作经理.xls	2017/4/12 14:20	Microsoft Excel ...	9 KB
电工维修工.xls	2017/4/12 14:20	Microsoft Excel ...	6 KB
动画D设计.xls	2017/4/12 14:20	Microsoft Excel ...	6 KB
高级商务经理.xls	2017/4/12 14:20	Microsoft Excel ...	7 KB
工艺设计工程师.xls	2017/4/12 14:20	Microsoft Excel ...	7 KB
工装图纸深化绘图员 职位编号.xls	2017/4/12 14:20	Microsoft Excel ...	9 KB
母婴渠道销售.xls	2017/4/12 14:20	Microsoft Excel ...	9 KB
平面设计.xls	2017/4/12 14:20	Microsoft Excel ...	10 KB
设计师男装衬衫.xls	2017/4/12 14:20	Microsoft Excel ...	5 KB
数据中心运营总监副总.xls	2017/4/12 14:20	Microsoft Excel ...	6 KB
网络工程培训.xls	2017/4/12 14:20	Microsoft Excel ...	11 KB

2.4 KMeans 实验要求

- (1) 理解 KMeans 算法原理，根据算法流程图和示例代码自己编写 KMeans 程序；
- (2) 编写 KMeans 的测试程序，回顾前面的实验，实现读取实验一爬取的数据、进行分词、去停用词、向量转化、Canopy 聚类、KMeans 聚类等过程；
- (3) 将 Canopy 聚类结果的类别数作为 KMeans 算法的初始 K 值进行 KMeans 聚类，并运行 KMeans 测试程序，能在控制台输出聚类好的数据；
- (4) 根据聚类结果合理调整 K 值和迭代次数，然后进行聚类，并不断优化代码使得聚类效果更好，最后将 K 值设为 10 的聚类结果上传；
- (5) 编写程序，功能是统计每个类中出现次数最多的一条文本作为 Excel 文件名，并将该类全部项保存到该 Excel 文件，输入参数为聚类效果最好的一次聚类后的数据。

3. 作业要求

登录大数据教学平台，将实验所有作业文件上传。作业文件包括：

- （1）原始数据，以“原始数据.xls”表格形式提交；
- （2）Canopy 及 KMeans 程序源代码压缩包（整个项目导出，压缩），以“bigdata.zip”压缩包形式提交；
- （3）通过程序自动计算 Canopy 聚类的阈值（提示：阈值可取相似度平均值，所给源代码中有计算阈值的方法，直接调用即可），聚类好的数据，每个类的数据保存到一个 Excel 文件，将所有 Excel 文件打包，以“canopyresult.zip”压缩包形式提交；
- （4）将聚类的初始 K 值设为 10，聚类好的数据，每个类的数据保存到一个 Excel 文件，将所有 Excel 文件打包，以“kmeansresult.zip”压缩包形式提交。