Homework3: Write a k-means clustering algorithm on Hadoop

组员：王宇琪、胡晨旭、张智为

数据预处理

Email dataset 读入

依次读入每一封邮件，整个邮件读入成一个字符串，然后加入到 all_ham_text 和 all_spam_text 中

```
all_ham_text = []
all_spam_text = []
for index in ham_train_files:
    file = os.path.join(ham_train_root,index)
    with open(file, 'r',encoding='utf-8') as f:
        text = f.read()
    all_ham_text.append(text)

for index in spam_train_files:
    file = os.path.join(spam_train_root,index)
    with open(file, 'r', errors='ignore',encoding='utf-8') as f:
        text = f.read()
    all_spam_text.append(text)
```

使用 Python 库 nltk 进行以下的处理

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
from nltk.stem.porter import PorterStemmer
```

1. 分词

```
tokenizer = RegexpTokenizer(r'\w+')
tok = tokenizer.tokenize(all_ham_text[i].lower())
```

tokenizer.tokenize 会返回一个 list，其中就是每个`element`就是一个单词。它会自动识别标点符号，然后将其忽略掉。

结果例子展示：

```
In [6]: text
        executed in 20ms, finished 17:19:21 2019-10-16
Out[6]: 'Subject: california update 7 / 25 / 01 - - - - meeting w / assembly on its latest\nversion of edison mou\nadding to kristin \' s update , here \' s the summary of a meeting that just occured with assembly leadership staff and industry on the assembly \' s latest version of an " edison mou " :\nthe speaker \' s staffer said that if he had to bet at this time , he would bet that the assembly would not convene on friday .\nhe also said that , if it doesn \' t happen on friday , nothing will happen until they get back from recess on the 20 th of august .\nthe assembly will let folks know by tonight or first thing tomorrow whether they intend to move forward with a vote on friday or recess .\nfrom the most recent proposal released today by the assembly leadership , it appears that the centrist democrats are prevailing . the proposal :\nretains direct access ( though they said that they are continuing to be pressured by the treasurer , who wants da suspended due to the bond issuance )\nplaces 80 % of edison \' s past debt with large customers ( leaving 20 % with small customers , which business customers or strenuously opposing ) .'

In [8]: tokenizer.tokenize(text.lower())
        executed in 11ms, finished 17:20:27 2019-10-16
Out[8]: ['subject',
         'california',
         'update',
         '7',
         '25',
         '01',
         'meeting',
         'w',
         'assembly',
         'on',
         'its',
         'latest',
         'version',
         'of',
         'edison',
         'mou',
         'adding',
         'to',
         'kristin',
```

2. 去除 stop words

nltk 中有 English 的 stopwords。这是一个 list，其中有英文的 stopwords。`

```
print(stopwords.words('english'))
executed in 8ms, finished 17:23:45 2019-10-16
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
"you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',
'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o',
're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't",
'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn',
"wouldn't"]
```

```
# 从分词结果中去除掉 stopwords
words = [w for w in tok if w not in stopwords.words('english')]
```

3. 提取词干

使用 stemmer = PorterStemmer()，会把每个词转化成它的词干。

使用方法 stemmer.stem(w)，返回单词 w 的词干

示例如下：

```
In [18]: [stemmer.stem(w) for w in ['happy','company', 'study', 'playing'] ]
         executed in 12ms, finished 17:47:04 2019-10-16
Out[18]: ['happi', 'compani', 'studi', 'play']
```

将以上三步组合进行处理：

```
tokenizer = RegexpTokenizer(r'\w+')
stemmer = PorterStemmer()
words_ham_split = []

for i in range(len(all_ham_text)):
    # 分词
    tok = tokenizer.tokenize(all_ham_text[i].lower())
    # 去除 stopwords
    words = [w for w in tok if w not in stopwords.words('english')]
    # 去除短的字符串
    de_small = [w for w in words if len(w) > 2]
    # 提取词干
    stem_res = [stemmer.stem(w) for w in de_small]
    words_ham_split.extend(stem_res)
```

Email document embedding

在 ham/spam train data 中选择出现次数最多的 1000 个单词。ham/spam 各 500 个单词。因为整个训练集当中，出现的单词种类有 20 多万种，所以如果用每个 email 的各个单词的词频来表示文本的话，会导致每个 document 的 vector 过大，无法进行训练。所以选择 spam 和 ham 里出现最多的 1028 个词的词频来做 document embedding。

```
from collections import Counter
ham_count = Counter(words_ham_split)
# 按出现次数进行排序
sorted_ham_count = ham_count.most_common()
# len(sorted_ham_count) == 48056
ham_begin = 50
```

```
ham_500 = sorted_ham_count[ham_begin:700+ham_begin]
ham_500 = [word[0] for word in ham_500]
# 同理对 spam 进行计数
spam_count = Counter(words_spam_split)
sorted_spam_count = spam_count.most_common()
spam_500 = sorted_spam_count[:700]
spam_500 = [w[0] for w in spam_500]
keyword_100 = set(ham_500+spam_500)
```

对每个文档进行清洗，只保留这 1028 个词

```
new_all_text = []
for mail in (all_ham_text+all_spam_text):
    # 只有在这 1028 个词里的词才会被保留
    new_all_text.append([w for w in mail.split(" ") if w in keyword_1000])
corpus = [" ".join(w) for w in new_all_text]
```

利用 sklearn 里面的 CountVectorizer 对预处理过后的文本进行 embedding。它会把每个文章转化为一个 1028 维的向量，其中第 i 维上的数值，表示了单词w_i在此文本出现的次数。

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
# X is a 29991x1028 sparse matrix, which is all documents embedding
new_keyword_1000 = vectorizer.get_feature_names()
# new_keyword_1000 is a list of 1028 words, where new_keyword_1000[i] is wi
```

把这个稀疏矩阵 X 以文本形式输出，以方便 Java 进行数据的读入。注意：numpy 的数据格式.npz，Java 无法读取。

```
X_dense = X.todense()
# 在每个向量前面加一维表示文本的序号，便于 JAVA 处理
a = np.zeros((X_dense.shape[0],1),dtype=np.int64)
for i in range(a.shape[0]):
    a[i,0] = i
X_dense_index = np.hstack((a,X_dense))
np.savetxt('all-index-data.txt', X_dense_index, delimiter=' ',fmt='%d',newline='\n')
```

之后就可以用输出的 txt 文本，利用 JAVA 在 Hadoop 进行处理。

KMeans 算法

通过之前对邮件的预处理我们把每个邮件样本输出成 1029（1028+1）维的向量，其中包含一维 index 不参与 Kmeans 的运算，但对后续的准确率测试提供帮助。

Kmeans 流程：

（1）初始化：选择 k 个初始中心点，c[0]=data[0],…c[k-1]=data[k-1]，写到 centers.txt 作为输入参数；

（2） Map：对于 data[0]….data[n]，分别与 c[0]…c[k-1]比较，假定与 c[i]差值最少，就标记为 i；

（3） Reduce：对于所有标记为 i 点，重新计算 c[i]={ 所有标记为 i 的 data[j]之和}/标记为 i 的个数；

（4） 重复(2)(3),直到所有 c[i]值的变化小于给定阈值。

代码理解：

KMeansMapper.java：在 map 的过程中，对读入的一条邮件数据计算其到所有中心的距离，计算出最短距离中心的 index，并以 index 和邮件的特征形成一个新的键值对。

```
public class KMeansMapper extends Mapper<Object, Text, IntWritable, Text> {
    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException{
        String line = value.toString();
        String[] fields = line.split(" ");
        List<ArrayList<Float>> centers = Assistance.getCenters(context.getConfiguration().get("centerpath"));
        int k = Integer.parseInt(context.getConfiguration().get("kpath"));
        float minDist = Float.MAX_VALUE;
        int centerIndex = k;
        //计算样本点到各个中心的距离，并把样本聚类到距离最近的中心点所属的类
        for (int i = 0; i < k; ++i){
            float currentDist = 0;
            for (int j = 0; j+1 < fields.length; ++j){
                float tmp = Math.abs(centers.get(i).get(j + 2) - Float.parseFloat(fields[j+1]));
                currentDist += Math.pow(tmp, 2);
            }
            if (minDist > currentDist){
                minDist = currentDist;
                centerIndex = i;
            }
        }
        context.write(new IntWritable(centerIndex), new Text(value));
    }
}
```

KMeansReducer.java：Reduce 时，将文本的类型的数据转化为浮点计算为平均值，得到新的聚类中心，然后再转成文本写到输出中。

```java
public class KMeansReducer extends Reducer<IntWritable, Text, IntWritable, Text> {
    public void reduce(IntWritable key, Iterable<Text> value, Context context)
            throws IOException, InterruptedException{
        List<ArrayList<Float>> assistList = new ArrayList<ArrayList<Float>>();
        String tmpResult = "";
        for (Text val : value){
            String line = val.toString();
            String[] fields = line.split(" ");
            List<Float> tmpList = new ArrayList<Float>();
            for (int i = 0; i < fields.length; ++i){
                tmpList.add(Float.parseFloat(fields[i]));
            }
            assistList.add((ArrayList<Float>) tmpList);
        }
        //计算新的聚类中心
        for (int i = 0; i < assistList.get(0).size(); ++i){
            float sum = 0;
            for (int j = 0; j < assistList.size(); ++j){
                sum += assistList.get(j).get(i);
            }
            float tmp = sum / assistList.size();
            if (i == 0){
                tmpResult += tmp;
            }
            else{
                tmpResult += " " + tmp;
            }
        }
        Text result = new Text(tmpResult);
        context.write(key, result);
    }
}
```

KMeansDriver.java：定义了 main 类，在 main 中包含一个循环，每一次循环中执行一次 Hadoop 任务，实现一次 Kmeans 的迭代，如果满足条件，则终止迭代。

Assistance.java：提供一些辅助函数，如 getCenters 函数根据路径读取当前和前一次的聚类中心；deleteLastResult 函数删除上一次 Hadoop 作业的输出；isFinished 函数根据前后两次聚类中心的距离来判断是否终止迭代。

具体代码附在附录中。

准备工作

按照代码的输入参数需要，我们需要在 Hadoop 中提供：初始的聚类中心，输出入和输出文件。

- **初始的聚类中心**

为了让算法更快地收敛,我们选取了spam和ham中各一封邮件的特征作为初始的聚类中心，将两者的特征记录在一个文件centers.txt中，并用 *hadoop fs -put centers.txt /user/kmeans/*指令上传文件

- **输入文件**

将处理过的数据存在文件all-index-data.txt中，然后用 *hdfs dfs -mkdir -p /user/kmeans/input* 创建input文件夹

*hadoop fs -put all-index-data.txt /user/kmeans/inpu*t 上传数据至文件夹

- **输出文件**

hdfs dfs -mkdir -p /user/kmeans/out 创建out文件夹,用作Hadoop的输出路径。

在完成相应文件配置后，编译Java源文件

cd src

javac -encoding utf8 ./.java*

*jar -cvf KMeans.jar *.class*

输入指定参数，开始运行，我们设定k=2，threshold=10。

*hadoop jar *.jar * input output centerspath newcenters k threshold*

实际调用格式如下，我们设定k=2，threshold=10

hadoop jar KMeans.jar KMeansDriver /user/kmeans/input /user/kmeans/out /user/kmeans/centers.txt /user/kmeans/out/part-r-00000 2 10

实际测试

我们利用5个节点，取k=2，threashold=1时，运行Kmeans程序，并记录每一次迭代的程序输出如下：

1. Distance = 2330.2766 Threshold = 1.0

2. Distance = 683.9401 Threshold = 1.0

3. Distance = 6747.721 Threshold = 1.0

4. Distance = 8772.65 Threshold = 1.0

5. Distance = 1109.9999 Threshold = 1.0

6. Distance = 263.95486 Threshold = 1.0

7. Distance = 264.68964 Threshold = 1.0

8. Distance = 36.71771 Threshold = 1.0

9. Distance = 9.949009 Threshold = 1.0

10. Distance = 11.145945 Threshold = 1.0

11. Distance = 0.0 Threshold = 1.0

并且记录下总共运行的时间为 75 分钟，程序大多时间花在了 map 上，reduce 上较少。

Distance 随着迭代的进行逐渐减小，即 Kmeans 的聚类中心调整的距离缩短，聚类结果趋于稳定。

当选取不同的节点数量运行该程序时，每一轮的输出结果不变。

结果与思考

1. 运行结果

当 threshold=10 时，一共有 29991 个样本需要预测，其中有 17205 个垃圾邮件，16786 个正常邮件。我们将所有邮件分成了两类，一类具有 56 个，另一类具有 29936 个邮件。根据每一类的 id，我们将聚类结果分为 spam 和 ham，并计算了以下参数反映分类的结果。

迭代次数	准确率	精确度	召回率	F1
10	0.442	0.441	0.767	0.560

从以上数据可以看出，Kmeans 的结果不那么理想。

2.关键词选取

选取每一类中出现平均次数最高的 10 个词，作为每一类的关键词，计算结果如下（第一列 id，第二列是平均词频）：

Ham：

```
184      128.535714
808      104.392857
283       58.178571
887       52.428571
948       50.410714
122       49.910714
304       49.535714
839       46.678571
567       42.642857
619       41.625000
```

对应的词（经过预处理，只保留了词汇主干）分别是：compani said dynegi stock trade billion energi share market new

Spam：

```
896          1.417805
175          0.694070
699          0.625255
184          0.603374
464          0.438316
939          0.419643
720          0.415701
1020         0.394121
9            0.383598
619          0.376048
```

对应的词分别是：subject com pleas compani inform time price would 2000 new

可见正常邮件中，出现最多的词汇的频率比垃圾邮件高，即正常邮件中有一些词汇经常出现，而垃圾邮件中各种各样的词汇都有少量出现，这是符合认知的。如果看一下这些关键词，也可以发现 spam 中的词一般出现在广告中，ham 中的词汇更贴近于生活工作中的交流需要。

3. 多个节点运行时间比较

我们分别测试了 2,3,4,5 台服务器时，程序运行的时间，threshold 都选取为 10。

服务器个数	2	3	4	5
运行时间/min	63	77	69	75

节点的数量改变没有明显的提升运行的时间，我们认为有以下几点原因：

1.我们只选取了 k=2，当进行 reduce 时，只有两个节点的工作，因此时间差不多。

2.网络波动造成运算差异。在各个节点通讯时的不稳定，造成了运行时间的差异，但整体任务的运算时间是相近的。

3.在数据传输上花费了主要的时间。可能此次数据量还不大，数据处理上的时间不多，每个节点处理数据的时间都是类似的，时间耗费主要在传输上，而传输的时间是相对固定的，所以总时间相似。

遇到的问题

1. Hadoop 自动在 key 和 value 之间增加制表符，导致数据格式不正确。

解决方法：在读取数据时，如果读到了制表符，就把它转化成空格处理。

2. 数据维度不一致，导致程序运行报错。

解决方法：centers 含有标签，所以维数比数据多一维，在进行数据处理时，需要注意循环的起始和终止条件。

3.对于所有实验，运行 map 任务时，完成至 67%后会直接跳到 100%。

尚未能较好地解释该现象。猜测是最后 33%的 map 任务在子节点上已经完成。

4.分类的结果不够理想。

可能是我们选取的特征较为简单，如有时间将尝试效果更好地特征。

重回正轨

在 10 月 16 日晚上的上机课上，我们和助教和老师进行了一些交流，才发现我们理解错了题目的意思

. You are asked to do clustering for both ham and spam emails.

意思是给垃圾邮件和正常邮件**分别**做聚类分析。而我们之前的理解是把所有的邮件分为垃圾邮件和正常邮件两类，显然，对于 Kmeans 这种无监督学习，分成特定的类别并与原来的标签比较是的意义是不明确的，应该按照数据本身的性质进行分类。

我们依旧使用之前处理的数据，把 ham 和 spam 的数据分别上传至服务器。我们先运行了 ham 的数据，选取 k=10，threshold=1。

经过了 110 分钟，我们的程序达到了最大迭代次数 20 次，最终 Distance=54.356606 结束了迭代。选取每一类出现频率最大的 10 个词作为关键词。

0	compani	said	dynegi	energi	stock	trade	billion	share	new	deal
1	com	net	home	yahoo	org	edu	usa	mail	michael	inc
2	compani	said	share	stock	energi	billion	jone	dow	trade	new
3	presid	manag	busi	ceo	oper	new	chairman	risk	global	year
4	subject	pleas	would	2000	thank	time	need	meet	work	forward
5	market	power	price	energi	state	ga	compani	said	california	would
6	com	subject	mail	messag	pleas	would	2000	kaminsk	sent	energi
7	deal	subject	2000	ga	meter	000	term	daren	volum	pleas
8	compani	said	stock	billion	trade	energi	rate	share	market	new
9	subject	pleas	thank	2000	com	know	data	attach	data	deal

每个类的邮件个数分别是：

类别	0	1	2	3	4	5	6	7	8	9
个数	16	1	43	72	3516	194	596	444	29	9875

由于时间原因，我们只跑了 ham 的数据，可以看到 k=10 并非完全合理，如有时间可以尝试更小的 k。但从关键词来说，每一类都比较有特色，聚类效果较为明显。

组员分工

胡晨旭：数据预处理，及此部分的报告

王宇琪：运行 Kmeans，后续结果分析

张智为：运行 Kmeans，报告整合

附录

KMeansDriver.java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

import java.io.IOException;

public class KMeansDriver {
    public static void main(String[] args) throws Exception {
        int repeated = 0;
        /*
        不断提交 MapReduce 作业指导相邻两次迭代聚类中心的距离小于阈值或到达设定的迭代次
数
        */
        do {
            Configuration conf = new Configuration();
            String[] otherArgs = new GenericOptionsParser(conf, args).getRemain
ingArgs();
            if (otherArgs.length != 6) {
                System.err.println("Usage: <int> <out> <oldcenters> <newcenters
> <k> <threshold>");
                System.exit(2);
            }
            conf.set("centerpath", otherArgs[2]);
            conf.set("kpath", otherArgs[4]);
            Job job = new Job(conf, "KMeansCluster");//新建 MapReduce 作业
            job.setJarByClass(KMeansDriver.class);//设置作业启动类

            Path in = new Path(otherArgs[0]);
            Path out = new Path(otherArgs[1]);
            FileInputFormat.addInputPath(job, in);//设置输入路径
            FileSystem fs = FileSystem.get(conf);
            if (fs.exists(out)) {//如果输出路径存在，则先删除之
                fs.delete(out, true);
            }
```

```
            FileOutputFormat.setOutputPath(job, out);//设置输出路径

            job.setMapperClass(KMeansMapper.class);//设置 Map 类
            job.setReducerClass(KMeansReducer.class);//设置 Reduce 类

            job.setOutputKeyClass(IntWritable.class);//设置输出键的类
            job.setOutputValueClass(Text.class);//设置输出值的类

            job.waitForCompletion(true);//启动作业

            ++repeated;
            System.out.println("We have repeated " + repeated + " times.");
        }
        while (repeated < 20 && (Assistance.isFinished(args[2], args[3], Intege
r.parseInt(args[4]), Float.parseFloat(args[5])) == false));
        //根据最终得到的聚类中心对数据集进行聚类
        Cluster(args);
    }

    public static void Cluster(String[] args)
            throws IOException, InterruptedException, ClassNotFoundException {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingA
rgs();
        if (otherArgs.length != 6) {
            System.err.println("Usage: <int> <out> <oldcenters> <newcenters> <k
> <threshold>");
            System.exit(2);
        }
        conf.set("centerpath", otherArgs[2]);
        conf.set("kpath", otherArgs[4]);
        Job job = new Job(conf, "KMeansCluster");
        job.setJarByClass(KMeansDriver.class);

        Path in = new Path(otherArgs[0]);
        Path out = new Path(otherArgs[1]);
        FileInputFormat.addInputPath(job, in);
        FileSystem fs = FileSystem.get(conf);
        if (fs.exists(out)) {
            fs.delete(out, true);
        }
        FileOutputFormat.setOutputPath(job, out);

        //因为只是将样本点聚类，不需要 reduce 操作，故不设置 Reduce 类
```

```
        job.setMapperClass(KMeansMapper.class);

        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(Text.class);

        job.waitForCompletion(true);
    }
}
```

KMeansMapper.java

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class KMeansMapper extends Mapper<Object, Text, IntWritable, Text> {
    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException{
        String line = value.toString();
        String[] fields = line.split(" ");
        List<ArrayList<Float>> centers = Assistance.getCenters(context.getConfi
guration().get("centerpath"));
        int k = Integer.parseInt(context.getConfiguration().get("kpath"));
        float minDist = Float.MAX_VALUE;
        int centerIndex = k;
        //计算样本点到各个中心的距离，并把样本聚类到距离最近的中心点所属的类
        for (int i = 0; i < k; ++i){
            float currentDist = 0;
            for (int j = 0; j+1 < fields.length; ++j){
                float tmp = Math.abs(centers.get(i).get(j + 2) - Float.parseFlo
at(fields[j+1]));
                currentDist += Math.pow(tmp, 2);
            }
            if (minDist > currentDist){
                minDist = currentDist;
                centerIndex = i;
            }
        }
        context.write(new IntWritable(centerIndex), new Text(value));
    }
}
```

KMeansReducer.java

```java
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class KMeansReducer extends Reducer<IntWritable, Text, IntWritable, Text
> {
    public void reduce(IntWritable key, Iterable<Text> value, Context context)
            throws IOException, InterruptedException{
        List<ArrayList<Float>> assistList = new ArrayList<ArrayList<Float>>();
        String tmpResult = "";
        for (Text val : value){
            String line = val.toString();
            String[] fields = line.split(" ");
            List<Float> tmpList = new ArrayList<Float>();
            for (int i = 0; i < fields.length; ++i){
                tmpList.add(Float.parseFloat(fields[i]));
            }
            assistList.add((ArrayList<Float>) tmpList);
        }
        //计算新的聚类中心
        for (int i = 0; i < assistList.get(0).size(); ++i){
            float sum = 0;
            for (int j = 0; j < assistList.size(); ++j){
                sum += assistList.get(j).get(i);
            }
            float tmp = sum / assistList.size();
            if (i == 0){
                tmpResult += tmp;
            }
            else{
                tmpResult += " " + tmp;
            }
        }
        Text result = new Text(tmpResult);
        context.write(key, result);
    }
}
```

Assistance.java

```java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.util.LineReader;

import java.io.IOException;
import java.util.*;

public class Assistance {
    //读取聚类中心点信息：聚类中心 ID、聚类中心点
    public static List<ArrayList<Float>> getCenters(String inputpath) {
        List<ArrayList<Float>> result = new ArrayList<ArrayList<Float>>();
        Configuration conf = new Configuration();
        try {
            FileSystem hdfs = FileSystem.get(conf);
            Path in = new Path(inputpath);
            FSDataInputStream fsIn = hdfs.open(in);
            LineReader lineIn = new LineReader(fsIn, conf);
            Text line = new Text();
            while (lineIn.readLine(line) > 0) {
                String record = line.toString();
                /*
                因为 Hadoop 输出键值对时会在键跟值之间添加制表符，
                所以用空格代替之。
                */
                String[] fields = record.replace("\t", " ").split(" ");
                List<Float> tmplist = new ArrayList<Float>();
                for (int i = 0; i < fields.length; ++i) {
                    tmplist.add(Float.parseFloat(fields[i]));
                }
                result.add((ArrayList<Float>) tmplist);
            }
            fsIn.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return result;
    }

    //删除上一次 MapReduce 作业的结果
```

```
    public static void deleteLastResult(String path) {
        Configuration conf = new Configuration();
        try {
            FileSystem hdfs = FileSystem.get(conf);
            Path path1 = new Path(path);
            hdfs.delete(path1, true);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //计算相邻两次迭代结果的聚类中心的距离，判断是否满足终止条件
    public static boolean isFinished(String oldpath, String newpath, int k, flo
at threshold)
            throws IOException {
        List<ArrayList<Float>> oldcenters = Assistance.getCenters(oldpath);
        List<ArrayList<Float>> newcenters = Assistance.getCenters(newpath);
        float distance = 0;
        for (int i = 0; i < k; ++i) {
            for (int j = 2; j < oldcenters.get(i).size(); ++j) {
                float tmp = Math.abs(oldcenters.get(i).get(j) - newcenters.get(
i).get(j));
                distance += Math.pow(tmp, 2);
            }
        }
        System.out.println("Distance = " + distance + " Threshold = " + thresho
ld);
        if (distance < threshold)
            return true;
        /*
        如果不满足终止条件，则用本次迭代的聚类中心更新聚类中心
        */
        Assistance.deleteLastResult(oldpath);
        Configuration conf = new Configuration();
        FileSystem hdfs = FileSystem.get(conf);
        hdfs.copyToLocalFile(new Path(newpath), new Path("/home/hadoop/class/ol
dcenter.data"));
        hdfs.delete(new Path(oldpath), true);
        hdfs.moveFromLocalFile(new Path("/home/hadoop/class/oldcenter.data"), n
ew Path(oldpath));
        return false;
    }
}
```