# 一 Project 内容

1：用 MapReduce 算法实现贝叶斯分类器的训练过程，并输出训练模型；

2：用输出的模型对测试集文档进行分类测试。测试过程可基于单机 Java 程序，也可以是 MapReduce 程序。输出每个测试文档的分类结果；

3：利用测试文档的真实类别，计算分类模型的 Precision，Recall 和 F1 值。

# 二 贝叶斯分类器理论介绍

## 2.1 朴素贝叶斯方法介绍

问题：给定一个类标签集合 $C = c_1,c_2,...,c_j$ 以及一个文档 d，给文档 d 分配一个最合适的类别标签

基本思想：对于类标签集合 C 中的每个类标签 $c_i(i = 1,...,j)$, 计算条件概率 $p(ci|d)$，使条件概率 $p(ci|d)$ 最大的类别作为文档 d 最终的类别。

- 基于概率的思想，给定一个文档 d，看该文档属于哪个类别的可能性最大，就认为该文档属于哪个类别。
- 类似于赌博游戏，要你猜硬币的正、反面，猜对了有奖励。假如我告诉你出现正面的几率是 60%，现负面的几率是 40%，你会怎么选择？当然会选择正面，因为这样你有更高的几率赢钱。

因此 Naïve Bayes 是一个基于概率的分类器。Naïve Bayes 是一个简单、速度快的分类器，效果也非常好。经常作为分类器性能比较的基准（Base line）。

## 2.2 Bayes 公式

上面我们提到了通过计算每个类的条件概率来选择最大的概率的类别作为文档 d 的最终类别，那么我们该如何计算条件概率呢。这里我们要利用 Bayes 公式：

$$p(c_i|d) = \frac{p(d|c_i)p(c_i)}{p(d)}$$

其中：

- $p(c_i|d)$ 为后验概率或条件概率
- $p(c_i)$ 为先验概率
- $p(d|c_i)$ 为似然概率
- $p(d)$ 为证据

Bayes 公式的意义：

1、当观察到 evidence $p(d)$ 时，后验概率 $p(ci|d)$ 取决于似然概率 $p(d|ci)$ 和先验概率 $p(ci)$。因为当 evidence $p(d)$ 已知时，$p(d)$ 成为常量，Bayes 公式变成：

$$p(c_i|d) = \frac{p(d|c_i)p(c_i)}{p(d)} \propto p(d|c_i)p(c_i)$$

2、当先验概率 $p(c1) = p(c2) = ... = p(cj)$时，公式变为：

$$p(c_i|d) \propto p(d|c_i)$$

这时给定文档 d，该文档属于类别 $c_i$ 的概率 $p(c_i|d)$ 取决于似然概率 $p(d|c_i)$：

$p(d|c_i)$ 的涵义：给定文档类别 $c_i$，由类别 $c_i$ 产生文档 d 的可能性（likelihood）。

如果类别 $c_i$ 产生文档 d 的可能性 $p(d|ci)$ 最大，则文档 d 属于类别 $c_i$ 的概率 $p(c_i|d)$ 最大。这叫最大似然估计（Maximum Likelihood Estimation，MLE）。

### 2.3 朴素贝叶斯参数推导

现在再回到 Naïve Bayes 分类器

$$p(c_i|d) = \frac{p(d|c_i)p(c_i)}{p(d)} \propto p(d|c_i)p(c_i)$$

对于类标签集合 C 中的每个类标签 $c_i(i = 1,\ldots,j)$，计算条件概率 $p(c_i|d)$，使条件概率 $p(c_i|d)$最大的类别作为文档 d 最终的类别。

$$c_d = \underset{c_i \in C}{argmax} p(c_i|d) = \underset{c_i \in C}{argmax} p(d|c_i)p(c_i)$$

根据 Bayes 公式， $c_d$ =使得 $p(d|c_i)p(c_i)$ 值最大的类型。剩下的问题是如何得到 $p(d|c_i)p(c_i)$ ？对于 Naïve Bayes，用训练集对机器进行训练就是为了算出这两个参数，训练的过程就是参数估计的过程。

参数估计的过程为：

假设类别标签集合 $C = c1,c2,\ldots,cj$ 。假设训练集 D 包含 N 个文档，其中每个文档都被标上了类别标签。

1、首先估计先验概率 $p(c_i)(i = 1,\ldots,j)$

$$p(c_i) = \frac{\text{类型为}c_i\text{的文档个数}}{\text{训练集中文档总数 } N}$$

2、估计似然概率 $p(d|c_i)(i = 1,\ldots,j)$

为了估计 $p(d|c_i)$，需要一个假设：Term 独立性假设，即文档中每个 term 的出现是彼此独立的 。基于这个假设，似然概率 $p(d|c_i)$ 的估计方法如下，假设文档 d 包含 nd 个 term：t1, t2, …, tnd：

$$p(d|c_i) = p(t_1,t_2,\ldots,t_{n_d}|c_i) = p(t_1|c)p(t_2|c)\ldots p(t_{n_d}|c) = \prod_{1 \le k \le n_d} p(t_k|c_i)$$

因此，估计 $p(d|c_i)$ 就需要估计 $p(t_k|c_i)$ ：

$$p(t_k|c_i) = \frac{t_k\text{在类型为}c_i\text{的文档中出现的次数}}{\text{在类型为}c_i\text{的文档中出现的 } term \text{ 的总数}}$$

### 2.4 朴素贝叶斯分类器总结

对于类标签集合 C 中的每个类标签 $c_i(i = 1,\ldots,j)$，计算条件概率 $p(c_i|d)$，使条件概率 $p(c_i|d)$ 最大的类别作为文档 d 最终的类别，即：

$$c_d = \underset{c_i \in C}{argmax}\, p(c_i|d) = \underset{c_i \in C}{argmax}\, p(d|c_i)p(c_i) = \underset{c_i \in C}{argmax} \prod_{1 \le k \le n_d} p(t_k|c_i)\, p(c_i)$$

其中参数 $p(ci)(i = 1,\ldots,j)$ 通过训练集来估计

$$p(c_i) = \frac{\text{类型为}c_i\text{的文档个数}}{\text{训练集中文档总数 } N}$$

参数 $p(t_k|c_i)$ 通过训练集来估计

$$p(t_k|c_i) = \frac{t_k\text{在类型为}c_i\text{的文档中出现的次数}}{\text{在类型为}c_i\text{的文档中出现的 } term \text{ 的总数}}$$

## 三 贝叶斯分类器训练的 MapReduce 算法设计

基于 MapReduce 的朴素贝叶斯文档分类器算法主要包含 InitSequenceFileJob.java 、 InitSequenceFileInputFormat.java 、 GetDocCountFromDocTypeJob.java、GetSingleWordCountFromDocTypeJob.java、GetTotalWordCountFromDocTypeJob.java 、 GetNaiveBayesResultJob.java 和 Evaluation.java7 个程序组成，其中前面为 MapReduce 程序，Evaluation 是单机程序，用来评估分类结果。

### 3.1 InitSequenceFileJob

InitSequenceFileJob 中将.txt 格式的输入文件转换成 SequenceFile，其中的 Map 后的 Key 是 “文档类型@文件名”，Value 为输入的 txt 文件的内容，通过重写 FileInputFormat 和 RecordReader 将输入的文件内容处理成 byte 数组进行进行存储。Reduce 不做任何操作，因此所有的.txt 文件输出为一个序列文件，里面的 Key-Value 对即为 <文档类型@文件名，文档内容 >。

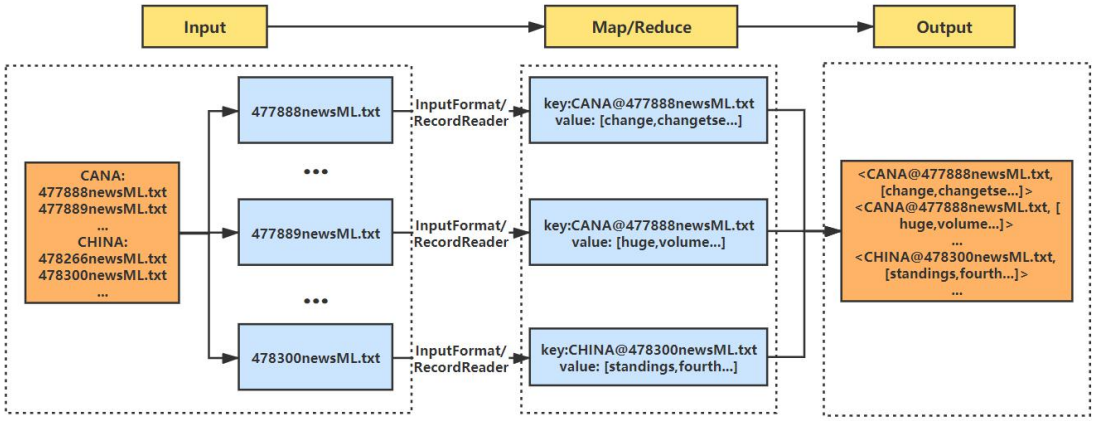


图 1 InitSequenceFileJob 中 MapReduce 程序的 Dataflow 图

### 3.2 GetDocCountFromDocTypeJob

GetDocCountFromDocTypeJob.java 中根据 InitSequenceFileJob 输出的 sequence_file 统计每个 DocType 有多少个文档。具体来说：

1. Map 输入为 Key-Value 对为 <文档类型@文件名，文档内容 > 的 SequenceFile，Map 之后的 key 为文档类型，如：CANA 或 CHINA，value 为 1，即每个文档对应一个键值对 <文档类型，1>

2. Map 输出的键值对传给 Combine，将相同 key 的键值对中的 value 合并为一个数组，此时的 key-value 对为：<文档类型，[1,1,...]>
3. Combine 的输出交给 Reducer，在 Reduce 中对 value 数组进行求和，这样就得到了每个文档类型如：CANA 和 CHINA 的文档总数，计算出来的文档总数将用于训练后续的先验概率。

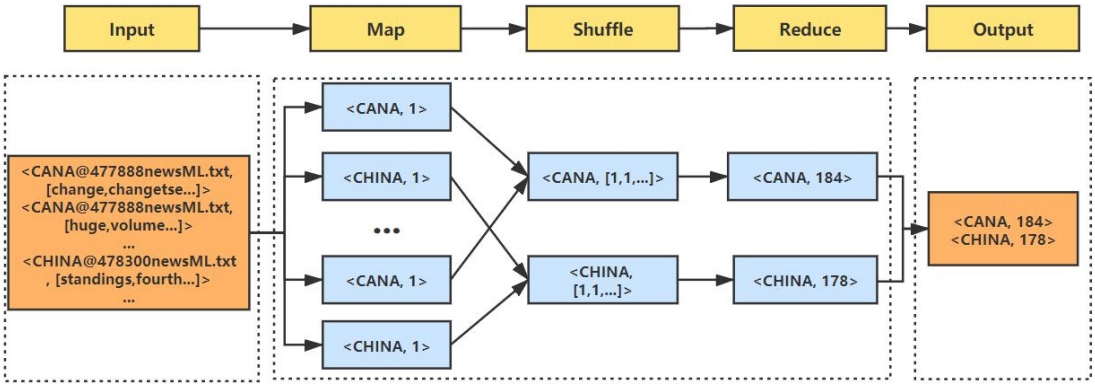


图 2 GetDocCountFromDocTypeJob 中 MapReduce 的 DataFlow 图

### 3.3 GetSingleWordCountFromDocTypeJob

GetSingleWordCountFromDocTypeJob.java 程序根据 InitSequenceFileJob 输出的 SequenceFile 统计每个单词在每个文档类别中出现的次数。具体来说：

1. Map 的输入为 Key-Value 对为 <文档类型@文件名，文档内容 > 的 SequenceFile，Map 之后的 key 为文档类型@单词，如：CANA@change，value 为 1，即每个文档中的单个单词对应一个键值对 <文档类型@单词，1>
2. Map 输出的键值对传给 Combine，将相同 key 的键值对中的 value 合并为一个数组，此时的 key-value 对为：<文档类型@单词，[1,1,...]>
3. Combine 的输出交给 Reducer，在 Reduce 中对 value 数组进行求和，这样就得到了每个文档类型中所有单个单词的数量，如：change 在 CANA 类别中的总数，计算出来的单词总数将用于训练后续的条件概率，此时输出的 key-value 对为：<文档类型@单词，单词总数> 。

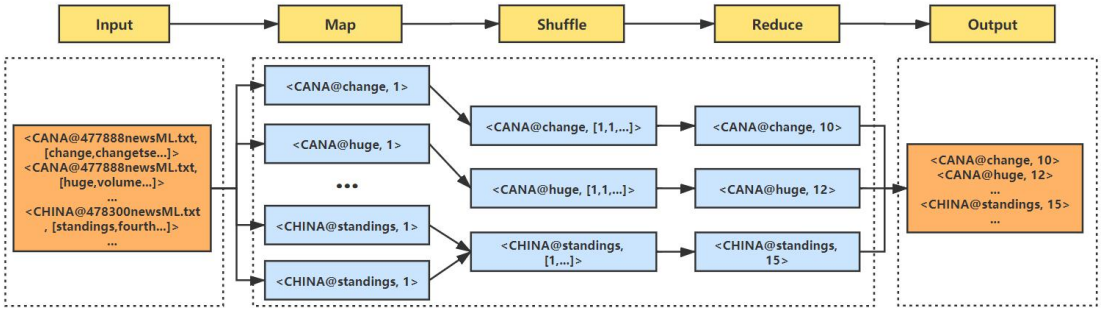


图 3 GetSingleWordCountFromDocTypeJob 中的 MapReduce 的 DataFlow 图

### 3.4 GetTotalWordCountFromDocTypeJob

GetTotalWordCountFromDocTypeJob.java 程 序 根 据

GetSingleWordCountFromDocTypeJob 输出的 sequence_file 统计每个文档类别的总单词数。具体来说：

1. Map 的输入为 Key-Value 对为 <文档类型@单词，单词数> 的 SequenceFile，Map 之后的 key 为文档类型，如：CANA，value 为每个单词在该文档类别中的总数，即每个文档中的单个单词对应一个键值对 <文档类型，某个单词总数>
2. Map 输出的键值对传给 Combine，将相同 key 的键值对中的 value 合并为一个数组，此时的 key-value 对为：<文档类型，[Count1,Count2,...]>
3. Combine 的输出交给 Reducer，在 Reduce 中对 value 数组进行求和，这样就得到了每个文档类型中所有单词的总数量，如：CANA 类型中的单词总数，计算出来的单词总数将用于训练后续的条件概率，此时输出的 key-value 对为：<文档类型，单词总数> 。

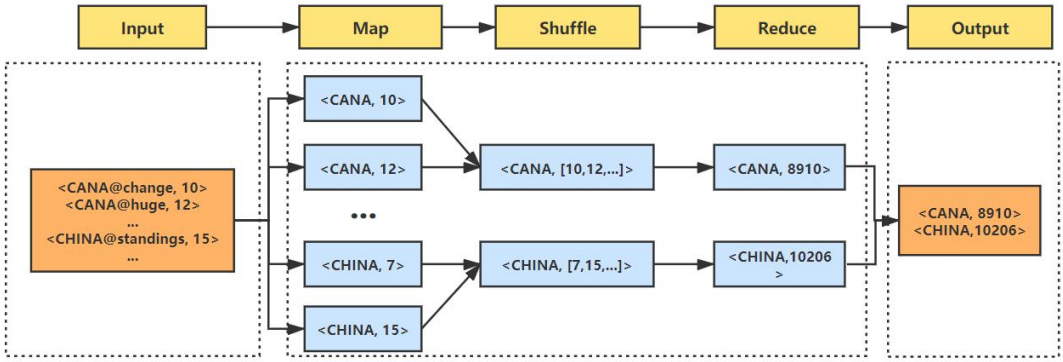


图 4 GetTotalWordCountFromDocTypeJob 中的 MapReduce 的 DataFlow 图

### 3.5 GetNaiveBayesResultJob

GetNaiveBayesResultJob.java 程序根据 InitSequenceFileJob 输出的 sequence_file 统计每个文档类别的总单词数。具体来说：

1. 在 Setup 中通过前面计算出的文档总数，单词总数，单个单词在文档中出现总数，计算训练集的先验概率、条件概率。
2. Map 的输入为 Key-Value 对为 <文档类型@文件名，文件内容> 的 SequenceFile，Map 之后的 key 为文档类型@文件名，如：CANA@477888newsML.txt，value 为该文档属于每个类别的概率，即每个文档对应一个键值对 <文档类型@文件名，每个类别对应的概率> 。
3. Map 的输出交给 Reducer，在 Reduce 中计算属于每个文档概率的最大值作为 value，这样就得到了每个文档属于最大概率的类别，此时输出的 key-value 对为：<文档类型@文件名，文档类型@最大条件概率> 。

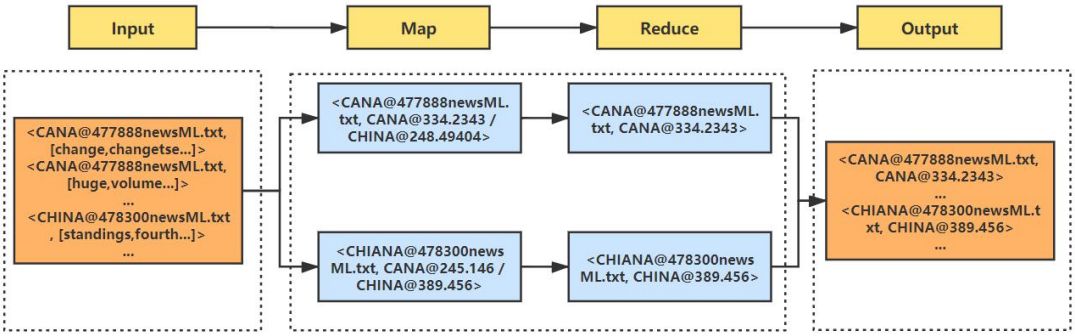


图 5 GetNaiveBayesResultJob 中的 MapReduce 的 DataFlow 图

### 3.6 Evaluation

`Evaluation.java` 程序对各文档的贝叶斯分类结果进行评估，计算各文档 FP、TP、FN、TN、Precision、Recall、F1 以及整体的宏平均、微平均。注：该程序为单机程序而非 MapReduce 程序。

## 四：源代码清单

### 4.1 InitSequenceFileJob

```
package job;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

import java.io.IOException;
import java.util.Arrays;

/**
 * 将.txt 格式的输入文件转换成 SequenceFile
```

```
 * 将"文档类型@文件名"作为 SequenceFile 的 Key 进行保存
 */
public class InitSequenceFileJob extends Configured implements Tool {

    /**
     * 重写 Mapper 函数
     */
    public static class InitSequenceFileMapper extends Mapper<NullWritable, BytesWritable,
Text, BytesWritable> {
        private Text docKey = new Text();

        /**
         * 在 map 之前运行,将 map 的 key 映射成：文档类型@文件名
         * 例如 CANA@477888newsML.txt
         */
        @Override
        protected void setup(Context context) throws IOException, InterruptedException {
//                System.out.println("start InitSequenceFileMapper's setup()");
            InputSplit split = context.getInputSplit();
            String docName = ((FileSplit) split).getPath().getName();
            String docType = ((FileSplit) split).getPath().getParent().getName();
            docKey.set(docType + "@" + docName);
//                System.out.println("将 map 的 key 映射成：" + docType + "@" + docName);
        }

        /**
         * 重写 map 函数，返回 key-value 对为 <文档类型@文件名，文件内容>
         */
        @Override
        protected void map(NullWritable key, BytesWritable value, Context context) throws
IOException, InterruptedException {
            // key: CANA@487557newsML.txt
            // value: 487557newsML.txt 的文件内容
//                System.out.println("start InitSequenceFileMapper's map()");
            context.write(this.docKey, value);
        }
    }

    @Override
    public int run(String[] strings) throws Exception {
        System.out.println("开始对 InitSequenceFileJob 进行配置");
        Configuration conf = getConf();
        Path inputPath = new Path(conf.get("INPUT_PATH"));
        Path outputPath = new Path(conf.get("OUTPUT_PATH"));
```

```
        FileSystem fs = outputPath.getFileSystem(conf);
        if (fs.exists(outputPath)) {
            fs.delete(outputPath, true);
        }

        fs = inputPath.getFileSystem(conf);
        //获取到的是 hdfs://master:8020/目录下的文件信息，即每个数据集文件夹的信息
        FileStatus[] inputFileStatusList = fs.listStatus(inputPath);
        String[] inputFilePathList = new String[inputFileStatusList.length];
        for (int i = 0; i < inputFilePathList.length; i++) {
            //获取每个数据集文件目录的路径，例如：hdfs://master:8020/CANA
            inputFilePathList[i] = inputFileStatusList[i].getPath().toString();
        }

        Job job = Job.getInstance(conf, "InitSequenceFileJob");

        job.setJarByClass(InitSequenceFileJob.class);
        job.setMapperClass(InitSequenceFileMapper.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(BytesWritable.class);
        job.setInputFormatClass(InitSequenceFileInputFormat.class);
        job.setOutputFormatClass(SequenceFileOutputFormat.class);

        // 将每个数据集文件夹导入到 InitSequenceFileInputFormat 中
        for (String path : inputFilePathList) {
            InitSequenceFileInputFormat.addInputPath(job, new Path(path));
        }
        SequenceFileOutputFormat.setOutputPath(job, outputPath);

        System.out.println("完成配置，开始执行 InitSequenceFileJob");
        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new Configuration(), new InitSequenceFileJob(), args);
        System.out.println("InitSequenceFileJob 运行结束");
        System.exit(exitCode);
    }
}
```

### 4.2 InitSequenceFileInputFormat

```
package job;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;

import java.io.IOException;
import java.util.Arrays;

/**
 * 重写 FileInputFormat 和 RecordReader 函数
 * 将 input 的数据处理成 bytes 数组，作为 InitSequenceFileJob 中 map 的 value
 */
public class InitSequenceFileInputFormat extends FileInputFormat<NullWritable, BytesWritable>
{
    static {
        System.out.println("开始处理 InitSequenceFileJob 的 InputFormat");
    }

    @Override
    public RecordReader<NullWritable, BytesWritable> createRecordReader(InputSplit inputSplit,
TaskAttemptContext taskAttemptContext) throws IOException, InterruptedException {
        InitSequenceFileRecordReader reader = new InitSequenceFileRecordReader();
//          System.out.println("start InitSequenceFileInputFormat's initialize() ");
        reader.initialize(inputSplit, taskAttemptContext);
        return reader;
    }
}

class InitSequenceFileRecordReader extends RecordReader<NullWritable, BytesWritable> {

    private FileSplit fileSplit;
    private Configuration conf;
```

```
    private BytesWritable value = new BytesWritable();
    private boolean processed = false;

    @Override
    public void initialize(InputSplit inputSplit, TaskAttemptContext taskAttemptContext) throws IOException, InterruptedException {
        this.fileSplit = (FileSplit) inputSplit;
        this.conf = taskAttemptContext.getConfiguration();
        // initialize fileSplit:hdfs://master:8020/input/CANA/478888newsML.txt
    }

    /**
     * 重写 nextKeyValue()函数，该函数会在 Mapper 中的 map 函数中赋值 value 的时候被调用
     */
    @Override
    public boolean nextKeyValue() throws IOException, InterruptedException {
        if (!processed) {
            byte[] contents = new byte[(int) fileSplit.getLength()];
            Path filePath = fileSplit.getPath();
            FileSystem fs = filePath.getFileSystem(conf);
            FSDataInputStream stream = null;
            try {
                stream = fs.open(filePath);
                // 将 file 文件中的内容放入 contents 数组中。
                // 使用了 IOUtils 实用类的 readFully 方法，将 in 流中得内容放入 contents 字节数组中。
                IOUtils.readFully(stream, contents, 0, contents.length);
                value.set(contents, 0, contents.length);
//                  System.out.println("next value 完成：" + Arrays.toString(contents));
            } finally {
                IOUtils.closeStream(stream);
            }
            this.processed = true;
            return true;
        }
        return false;
    }

    @Override
    public NullWritable getCurrentKey() throws IOException, InterruptedException {
        return NullWritable.get();
    }
```

```
    @Override
    public BytesWritable getCurrentValue() throws IOException, InterruptedException {
        return value;
    }

    @Override
    public float getProgress() throws IOException, InterruptedException {
        return processed ? 1.0f : 0.0f;
    }

    @Override
    public void close() throws IOException {

    }
}
```

### 4.3 GetDocCountFromDocTypeJob

```
package job;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.SequenceFileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import utils.Const;

import java.io.IOException;

/**
 * 根据 InitSequenceFileJob 输出的 sequence_file 统计每个 DocType 有多少个文档
 */
public class GetDocCountFromDocTypeJob extends Configured implements Tool {
```

```
    public static class GetDocCountMapper extends Mapper<Text, BytesWritable, Text, IntWritable> {
        private Text docTypeName = new Text();
        private IntWritable docCount = new IntWritable(1);

        /*
         * 重写 map 函数，输入为<文档类型@文件名，文件内容>，输出为<文档类型，1>
         */
        @Override
        protected void map(Text key, BytesWritable value, Context context) throws IOException, InterruptedException {
            // key: CANA@487557newsML.txt
            // value: 487557newsML.txt 的文件内容
            // 这里只取 key 的信息用来计算每个文档种类有多少个文档，value 不用管
            String[] keyName = key.toString().split("@");
            this.docTypeName.set(keyName[0]);
            this.docCount.set(1);
            context.write(this.docTypeName, docCount);
        }
    }

    public static class GetDocCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        private Text docTypeName = new Text();
        private IntWritable totalDocCount = new IntWritable(1);

        /*
         * 重写 reduce 方法，输入为<文档类型，[1,1,...]>，输出为<文档类型，该类型的文件总数>
         */
        @Override
        protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
            // key: CANA
            // values: [1,1,1,1,1.....,1,1,1]
            // 因为设置了 job.setCombinerClass(GetDocCountReducer.class);
            // 相同的 key 的 value 会合并成一个数组，数组的和就是改文档种类对应的文档总数
            int totalDocCount = 0;
            for (IntWritable docCount : values) {
                totalDocCount += docCount.get();
            }
```

```
                this.docTypeName.set(key);
                this.totalDocCount.set(totalDocCount);
                context.write(this.docTypeName, this.totalDocCount);
            }
        }

        @Override
        public int run(String[] strings) throws Exception {
            System.out.println("开始对 GetDocCountFromDocTypeJob 进行配置");

            Configuration conf = new Configuration();

            // 如果输出目录存在，则先删除输出目录
            Path outputPath = new
Path(Const.GET_DOC_COUNT_FROM_DOC_TYPE_JOB_OUTPUT_PATH);
            FileSystem fs = outputPath.getFileSystem(conf);
            if (fs.exists(outputPath)) {
                fs.delete(outputPath, true);
            }

            Job job = Job.getInstance(conf, "GetDocCountFromDocTypeJob");

            job.setJarByClass(GetDocCountFromDocTypeJob.class);
            job.setMapperClass(GetDocCountMapper.class);
            job.setCombinerClass(GetDocCountReducer.class);
            job.setReducerClass(GetDocCountReducer.class);

            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(IntWritable.class);
            job.setInputFormatClass(SequenceFileInputFormat.class);
            job.setOutputFormatClass(SequenceFileOutputFormat.class);

            FileInputFormat.addInputPath(job, new
Path(Const.TRAIN_DATA_SEQUENCE_FILE_PATH));
            FileOutputFormat.setOutputPath(job, new
Path(Const.GET_DOC_COUNT_FROM_DOC_TYPE_JOB_OUTPUT_PATH));

            System.out.println("完成配置，开始执行 GetDocCountFromDocTypeJob");
            return job.waitForCompletion(true) ? 0 : 1;
        }

        public static void main(String[] args) throws Exception {
            int res = ToolRunner.run(new Configuration(), new GetDocCountFromDocTypeJob(),
args);
```

```
            System.out.println("GetDocCountFromDocTypeJob 运行结束");
            System.exit(res);
        }
}
```

### 4.4 GetSingleWordCountFromDocTypeJob

```
package job;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.SequenceFileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import utils.Const;

import java.io.IOException;
import java.util.regex.Pattern;

/**
 * 根据 InitSequenceFileJob 输出的 SequenceFile 统计每个单词在每个文档类别中出现的次数
 */
public class GetSingleWordCountFromDocTypeJob extends Configured implements Tool {

    public static final Logger log = LoggerFactory.getLogger(GetSingleWordCountFromDocTypeJob.class);

    public static class GetSingleWordCountFromDocTypeMapper extends Mapper<Text, BytesWritable, Text, IntWritable> {
        // 声明单词的 Text
        private Text word = new Text();
```

```
        // 单词出现次数
        private IntWritable singleWordCountInEachDoc = new IntWritable(1);
        private static final Pattern ENGLISH_WORD_REGEX = Pattern.compile("^[A-Za-z]{2,}$");

        /*
         * 重写 map 函数，此时输入为<文档类型@文件名，文件内容>，输出为<文档类型@单词，1>
         */
        @Override
        protected void map(Text key, BytesWritable value, Context context) throws IOException, InterruptedException {
            // key: CANA@487557newsML.txt
            // value: 487557newsML.txt 的文件内容
            String docTypeName = key.toString().split("@")[0];
            // 将 sequence_file 中的 bytes 读成字符串
            String content = new String(value.getBytes());

            String[] wordList = content.split("\\s+");
            for (String word : wordList) {
                if (ENGLISH_WORD_REGEX.matcher(word).find() && !Const.STOP_WORDS_LIST.contains(word)) {
                    this.word.set(docTypeName + "@" + word);
                    context.write(this.word, this.singleWordCountInEachDoc);
                }
                // 处理训练集中出现的特殊字符对单词的影响
                else if (word.contains(".")) {
                    for (String maybeWord : word.split(".")) {
                        if (ENGLISH_WORD_REGEX.matcher(word).find()) {
                            this.word.set(docTypeName + "@" + maybeWord);
                            context.write(this.word, this.singleWordCountInEachDoc);
                        }
                    }
                } else if (word.contains("-")) {
                    for (String maybeWord : word.split("-")) {
                        if (ENGLISH_WORD_REGEX.matcher(word).find()) {
                            this.word.set(docTypeName + "@" + maybeWord);
                            context.write(this.word, this.singleWordCountInEachDoc);
                        }
                    }
                }
                else {
                    log.debug("过滤无用词：" + word);
                }
            }
```

```
        }
    }

    public static class GetSingleWordCountFromDocTypeJobReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        // 单词出现总次数
        private IntWritable singleWordCountInEachDocType = new IntWritable(1);

        /*
         * 重写 reduce 函数，输入为<文档类型@单词，[1,1,...]>，输出为<文档类型@单词，该类型中该单词的总数>
         */
        @Override
        protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
            // key: CANA@hello
            // value: [1,1,1,1,1....,1,1,1]
            int totalWordCount = 0;
            for (IntWritable wordCount : values) {
                totalWordCount += wordCount.get();
            }
            this.singleWordCountInEachDocType.set(totalWordCount);
            context.write(key, this.singleWordCountInEachDocType);
        }
    }

    @Override
    public int run(String[] strings) throws Exception {
        System.out.println("开始对 GetSingleWordCountFromDocTypeJob 进行配置");

        Configuration conf = new Configuration();

        // 如果输出目录存在，则先删除输出目录
        Path outputPath = new Path(Const.GET_SINGLE_WORD_COUNT_FROM_DOC_TYPE_JOB_OUTPUT_PATH);
        FileSystem fs = outputPath.getFileSystem(conf);
        if (fs.exists(outputPath)) {
            fs.delete(outputPath, true);
        }

        Job job = Job.getInstance(conf, "GetSingleWordCountFromDocTypeJob");

        job.setJarByClass(GetSingleWordCountFromDocTypeJob.class);
```

```
job.setMapperClass(GetSingleWordCountFromDocTypeJob.GetSingleWordCountFromDocTypeMapper.class);

job.setCombinerClass(GetSingleWordCountFromDocTypeJob.GetSingleWordCountFromDocTypeJobReducer.class);

job.setReducerClass(GetSingleWordCountFromDocTypeJob.GetSingleWordCountFromDocTypeJobReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setInputFormatClass(SequenceFileInputFormat.class);
        job.setOutputFormatClass(SequenceFileOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(Const.TRAIN_DATA_SEQUENCE_FILE_PATH));
        FileOutputFormat.setOutputPath(job, new Path(Const.GET_SINGLE_WORD_COUNT_FROM_DOC_TYPE_JOB_OUTPUT_PATH));

        System.out.println("完成配置，开始执行 GetSingleWordCountFromDocTypeJob");
        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(), new GetSingleWordCountFromDocTypeJob(), args);
        System.out.println("GetSingleWordCountFromDocTypeJob 运行结束，已计算每个文档类型中每个单词出现的次数");
        System.exit(res);
    }
}
```

### 4.5 GetTotalWordCountFromDocTypeJob

```
package job;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
```

```
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.SequenceFileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import utils.Const;

import java.io.IOException;

/**
 * 根据 GetSingleWordCountFromDocTypeJob 输出的 sequence_file 统计每个文档类别的总单词数
 */
public class GetTotalWordCountFromDocTypeJob extends Configured implements Tool {

    public static class GetTotalWordCountFromDocTypeMapper extends Mapper<Text, IntWritable, Text, IntWritable> {
        private Text docTypeName = new Text();
        // 该文档中每个单词出现的总次数
        private IntWritable wordCount = new IntWritable(0);

        /*
         * 重写 map 函数，输入为 GetSingleWordCountFromDocTypeJob 输出的 SequenceFile，输出为<文档类型，单词数量>
         */
        @Override
        protected void map(Text key, IntWritable value, Context context) throws IOException, InterruptedException {
            // key: CANA@hello
            // value: 13  表示 hello 在 CANA 文档类别中出现了 13 次
            String docTypeName = key.toString().split("@")[0];
            this.docTypeName.set(docTypeName);
            this.wordCount.set(value.get());
            context.write(this.docTypeName, this.wordCount);
        }
    }

    public static class GetTotalWordCountFromDocTypeReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        // 每个文档类别的单词总词数
        private IntWritable totalWordCount = new IntWritable(0);
```

```
        /*
         * 重写 reduce 函数，输入为<文档类型，[count1,count2,...]>，输出为<文档类型，该类型中的单词总数>
         */
        @Override
        protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
            // key: CANA
            // values: [13,1,1,24,3,7....12,3,6]
            // values 是中该类别中每个单词出现的次数组成的数组，数组求和即是每个文档类别的单词总词数
            int totalWordCount = 0;
            for (IntWritable wordCount : values) {
                totalWordCount += wordCount.get();
            }
            this.totalWordCount.set(totalWordCount);
            System.out.println(key.toString() + this.totalWordCount);
            context.write(key, this.totalWordCount);
        }
    }

    @Override
    public int run(String[] strings) throws Exception {
        System.out.println("开始对 GetTotalWordCountFromDocTypeJob 进行配置");

        Configuration conf = new Configuration();

        // 如果输出目录存在，则先删除输出目录
        Path outputPath = new Path(Const.GET_TOTAL_WORD_COUNT_FROM_DOC_TYPE_JOB_OUTPUT_PATH);
        FileSystem fs = outputPath.getFileSystem(conf);
        if (fs.exists(outputPath)) {
            fs.delete(outputPath, true);
        }

        Job job = Job.getInstance(conf, "GetTotalWordCountFromDocTypeJob");

        job.setJarByClass(GetTotalWordCountFromDocTypeJob.class);

job.setMapperClass(GetTotalWordCountFromDocTypeJob.GetTotalWordCountFromDocTypeMapper.class);

job.setCombinerClass(GetTotalWordCountFromDocTypeJob.GetTotalWordCountFromDocTypeRe
```

```
ducer.class);

job.setReducerClass(GetTotalWordCountFromDocTypeJob.GetTotalWordCountFromDocTypeReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setInputFormatClass(SequenceFileInputFormat.class);
        job.setOutputFormatClass(SequenceFileOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(Const.GET_SINGLE_WORD_COUNT_FROM_DOC_TYPE_JOB_OUTPUT_PATH));
        FileOutputFormat.setOutputPath(job, new Path(Const.GET_TOTAL_WORD_COUNT_FROM_DOC_TYPE_JOB_OUTPUT_PATH));

        System.out.println("完成配置，开始执行 GetTotalWordCountFromDocTypeJob");
        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(), new GetTotalWordCountFromDocTypeJob(), args);
        System.out.println("GetTotalWordCountFromDocTypeJob 运行结束，已计算所有文档类型中所有单词出现的次数");
        System.exit(res);
    }
}
```

### 4.6 GetNaiveBayesResultJob

```
package job;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.SequenceFileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

```
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import utils.Const;

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

/**
 * 首先在 setup 中计算训练集的先验概率、条件概率，并通过 MapReduce 任务计算测试集的每个文档分成每一类的概率
 * 读取 InitSequenceFileJob 生成的测试集的 SequenceFile 计算测试集的每个文档分成每一类的概率
 */
public class GetNaiveBayesResultJob extends Configured implements Tool {

    public static final Logger log = LoggerFactory.getLogger(GetNaiveBayesResultJob.class);

    // 文档种类列表
    private static String[] docTypeList;

    // 每个类别中每个单词出现的次数
    private static Map<String, Integer> eachWordCountInDocTypeMap = new HashMap<>();

    // 每个类别中所有单词出现的次数
    private static Map<String, Integer> allWordCountInDocTypeMap = new HashMap<>();

    // 每个文档 Ci 的先验概率 P(Ci)
    private static Map<String, Double> docTypePriorProbabilityMap = new HashMap<>();

     // 每个单词 Wi 的条件概率 P(Wi|Ci)
    private static Map<String, Double> wordConditionalProbabilityMap = new HashMap<>();

    // 每个文档的预测结果
    private static Map<String, String> docPredictResultMap = new HashMap<>();

    // 单词的正则表达式
    private static final Pattern ENGLISH_WORD_REGEX = Pattern.compile("^[A-Za-z]{2,}$");


    public static class GetNaiveBayesResultMapper extends Mapper<Text, BytesWritable, Text,
```

```
Text> {

        // 测试集中单词的条件概率
        Text conditionalProbabilityValue = new Text();

        /*
         * 读取之前所有任务输出的 SequenceFile 到内存中并在 Setup 中计算训练集的先验概率、条件概率
         */
        @Override
        protected void setup(Context context) throws IOException, InterruptedException {
//              System.out.println("start GetNaiveBayesResultMapper's setup()");
            Configuration conf = context.getConfiguration();
            Path getDocCountFromDocTypePath = new Path(Const.GET_DOC_COUNT_FROM_DOC_TYPE_JOB_OUTPUT_PATH + Const.HADOOP_DEFAULT_OUTPUT_FILE_NAME);
            Path getSingleWordCountFromDocType = new Path(Const.GET_SINGLE_WORD_COUNT_FROM_DOC_TYPE_JOB_OUTPUT_PATH + Const.HADOOP_DEFAULT_OUTPUT_FILE_NAME);
            Path getTotalWordCountFromDocType = new Path(Const.GET_TOTAL_WORD_COUNT_FROM_DOC_TYPE_JOB_OUTPUT_PATH + Const.HADOOP_DEFAULT_OUTPUT_FILE_NAME);

            conf.set("INPUT_PATH", Const.TEST_DATA_INPUT_PATH);
            conf.set("OUTPUT_PATH", Const.TEST_DATA_SEQUENCE_FILE_PATH);
            conf.set("DOC_TYPE_LIST", Const.DOC_TYPE_LIST);

            docTypeList = conf.get("DOC_TYPE_LIST").split("@");

            FileSystem fs = FileSystem.get(conf);
            // 读取 sequence_file
            SequenceFile.Reader reader = null;
            double totalDocCount = 0;
            Map<String, Integer> eachDocTypeDocCountMap = new HashMap<String, Integer>(10);
            try {
                //从 sequence_file 中读取每个文档类型中的总单词数
                SequenceFile.Reader.Option option = SequenceFile.Reader.file(getDocCountFromDocTypePath);
                reader = new SequenceFile.Reader(conf, option);
                Text key = new Text(); // key: CANA
                IntWritable value = new IntWritable(); // value: 300
                while (reader.next(key, value)) {
                    eachDocTypeDocCountMap.put(key.toString(),
```

```
Integer.parseInt(value.toString()));
                    totalDocCount += value.get();
                }
            } catch (Exception ex) {
                log.error(ex.getMessage());
            } finally {
                // 确保发生异常时关闭 reader
                IOUtils.closeStream(reader);
            }

            // 计算文档 Ci 的先验概率：P(Ci)=类型 Ci 的文档数/总文档数
            double finalTotalDocCount = totalDocCount;
            eachDocTypeDocCountMap.forEach((docTypeName, docCount) -> {
                double priorProbability = docCount / finalTotalDocCount;
                docTypePriorProbabilityMap.put(docTypeName, priorProbability);
                System.out.println("文档类型 " + docTypeName + " 的先验概率 P(Ci)=" +
priorProbability);
            });

            // 取出 sequence_file 中存储的类别中每个单词出现的次数 存储到 Map 中 形
式为：CANA@hello 13
            try {
                SequenceFile.Reader.Option option =
SequenceFile.Reader.file(getSingleWordCountFromDocType);
                reader = new SequenceFile.Reader(conf, option);
                Text key = new Text(); // key: CANA@hello
                IntWritable value = new IntWritable(); // value: 13
                while (reader.next(key, value)) {
                    eachWordCountInDocTypeMap.put(key.toString(), value.get());
                }
            } catch (Exception ex) {
                log.error(ex.getMessage());
            } finally { // 确保发生异常时关闭 reader
                IOUtils.closeStream(reader);
            }

            // 取出 sequence_file 中存储的每个类别中的所有单词出现的总次数
            try {
                SequenceFile.Reader.Option option =
SequenceFile.Reader.file(getTotalWordCountFromDocType);
                reader = new SequenceFile.Reader(conf, option);
                Text key = new Text(); // key: CANA
                IntWritable value = new IntWritable(); // value: 184032
                while (reader.next(key, value)) {
```

```
                    allWordCountInDocTypeMap.put(key.toString(), value.get());
                }
            } catch (Exception ex) {
                log.error(ex.getMessage());
            } finally { // 确保发生异常时关闭 reader
                IOUtils.closeStream(reader);
            }

            // 计算每个单词的条件概率
            eachWordCountInDocTypeMap.forEach((key, value) -> {
                String docType = key.split("@")[0];
                String word = key.split("@")[1];
                double probability = (value.doubleValue() + 1.0) /
allWordCountInDocTypeMap.get(docType).doubleValue();
                wordConditionalProbabilityMap.put(key, probability);
            });
        }

        /*
         * 重写 map 函数，输出为<文档类型@文件名，文档类型@概率>
         */
        @Override
        protected void map(Text key, BytesWritable value, Context context) throws IOException,
InterruptedException {
            // key: CANA@487557newsML.txt
            // value: 487557newsML.txt 的文件内容

            //计算文档 d 为类别 Ci 的条件概率：P(d|Ci)= ∏P(Wi|Ci)
            // 将 sequence_file 中的 bytes 读成字符串
            String content = new String(value.getBytes());
            String[] wordArray = content.split("\\s+");
            for (String docTypeName : docTypeList) {
                double conditionalProbability = 0;
                for (String word : wordArray) {
                    if (ENGLISH_WORD_REGEX.matcher(word).find()
&& !Const.STOP_WORDS_LIST.contains(word)) {
                        String wordKey = docTypeName + "@" + word;
                        if (wordConditionalProbabilityMap.containsKey(wordKey)) {
                            conditionalProbability +=
Math.log10((wordConditionalProbabilityMap.get(wordKey)));
                        } else {
                            // 如果测试集出现了训练集中没有出现过的单词，那么就
把该单词在类型为 Ci 的文档中出现的次数设置为 1
                            conditionalProbability += Math.log10(1.0 /
```

```
allWordCountInDocTypeMap.get(docTypeName).doubleValue());
                    }
                } else {
                    log.debug("过滤无用词：" + word);
                }
            }
            // 再加上文档 Ci 的条件概率
            conditionalProbability += Math.log10(docTypePriorProbabilityMap.get(docTypeName));
            this.conditionalProbabilityValue.set(docTypeName + "@" + conditionalProbability);
            context.write(key, conditionalProbabilityValue);
        }
    }
}

public static class GetNaiveBayesResultReducer extends Reducer<Text, Text, Text, Text> {

    // 测试集中文档被分为 Ci 类的概率
    Text docTypeForecastResult = new Text();

    /*
     * 重写 reduce 函数，输入为<文档类型@文件名，文档类型@概率>，输出为<文档类型@文件名，文档类型@最大概率>
     */
    @Override
    // 计算文档 d 是哪一类
    protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        // key: CANA@487557newsML.txt
        // value : CANA@-334.2343 / CHINA@-484.49404

        // 最大概率默认负无穷
        double maxProbability = Double.NEGATIVE_INFINITY;
        String forecastDocType = "";
        for (Text value : values) {
            double forecastProbability = Double.parseDouble(value.toString().split("@")[1]);
            if (forecastProbability > maxProbability) {
                maxProbability = forecastProbability;
                forecastDocType = value.toString().split("@")[0];
            }
        }
        this.docTypeForecastResult.set(forecastDocType + "@" + maxProbability);
```

```
                context.write(key, docTypeForecastResult);
//                  System.out.println(key.toString() + "  预测分类为：  " + forecastDocType + "  ，
预测概率为：" + maxProbability);
            }
        }

        @Override
        public int run(String[] strings) throws Exception {
            System.out.println("开始对  GetNaiveBayesResultJob  进行配置");

            Configuration conf = new Configuration();

            // 如果输出目录存在，则先删除输出目录
            Path outputPath = new Path(Const.GET_NAIVE_BAYES_RESULT_JOB_OUTPUT_PATH);
            FileSystem fs = outputPath.getFileSystem(conf);
            if (fs.exists(outputPath)) {
                fs.delete(outputPath, true);
            }

            Job job = Job.getInstance(conf, "GetNaiveBayesResultJob");

            job.setJarByClass(GetNaiveBayesResultJob.class);
            job.setMapperClass(GetNaiveBayesResultJob.GetNaiveBayesResultMapper.class);
            job.setCombinerClass(GetNaiveBayesResultJob.GetNaiveBayesResultReducer.class);
            job.setReducerClass(GetNaiveBayesResultJob.GetNaiveBayesResultReducer.class);

            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(Text.class);
            job.setInputFormatClass(SequenceFileInputFormat.class);
            job.setOutputFormatClass(SequenceFileOutputFormat.class);

            FileInputFormat.addInputPath(job,                                                    new
Path(Const.TEST_DATA_SEQUENCE_FILE_PATH));
            FileOutputFormat.setOutputPath(job,                                                  new
Path(Const.GET_NAIVE_BAYES_RESULT_JOB_OUTPUT_PATH));

            System.out.println("完成配置，开始执行  GetNaiveBayesResultJob");
            return job.waitForCompletion(true) ? 0 : 1;
        }

        public static void main(String[] args) throws Exception {
            int res = ToolRunner.run(new Configuration(), new GetNaiveBayesResultJob(), args);
            System.out.println("GetNaiveBayesResultJob  运行结束");
            System.exit(res);
```

```
    }
}
```

### 4.7 Evaluation

```
package job;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.util.Tool;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import utils.Const;

import java.util.HashMap;
import java.util.Map;

/**
 * 对各文档的贝叶斯分类结果进行评估，计算各文档 FP、TP、FN、TN、Precision、Recall、F1 以及整体的宏平均、微平均。
 */
public class Evaluation extends Configured implements Tool {

    // 每个文档的预测结果
    private static Map<String, String> docPredictResultMap = new HashMap<>();

    // 文档种类列表
    private static String[] docTypeList;

    public static final Logger log = LoggerFactory.getLogger(Evaluation.class);

    public static void doEvaluation() {
        docTypeList = Const.DOC_TYPE_LIST.split("@");
        Path bayesResult = new Path(Const.GET_NAIVE_BAYES_RESULT_JOB_OUTPUT_PATH +
Const.HADOOP_DEFAULT_OUTPUT_FILE_NAME);
        Configuration conf = new Configuration();
        SequenceFile.Reader reader = null;
        try {
            SequenceFile.Reader.Option option = SequenceFile.Reader.file(bayesResult);
            reader = new SequenceFile.Reader(conf, option);
```

```
Text sequenceKey = new Text();
// key: CHINA@481518newsML.txt
Text sequenceValue = new Text();
// value: CANA@-1953.9381329830253
while (reader.next(sequenceKey, sequenceValue)) {
    docPredictResultMap.put(sequenceKey.toString(), sequenceValue.toString());
}
double totalPrecision = 0, totalRecall = 0, totalF1 = 0;
double totalTP = 0, totalTN = 0, totalFP = 0, totalFN = 0;
for (String c : docTypeList) {
    double TP = 0, TN = 0, FP = 0, FN = 0;
    for (String key : docPredictResultMap.keySet()) {
        String value = docPredictResultMap.get(key);
        String realDocType = key.split("@")[0];
        String predictDocType = value.split("@")[0];
        if (realDocType.equals(c) && predictDocType.equals(c)) {
            TP++;
        } else if (realDocType.equals(c)) {
            FN++;
        } else if (predictDocType.equals(c)) {
            FP++;
        } else {
            TN++;
        }
    }
    double precision = TP / (TP + FP);
    totalPrecision += precision;
    double recall = TP / (TP + FN);
    totalRecall += recall;
    double f1 = 2 * precision * recall / (precision + recall);
    totalF1 += f1;
    totalTP += TP;
    totalFN += FN;
    totalTN += TN;
    totalFP += FP;
    System.out.print(c + " TP= " + TP);
    System.out.print(" FN= " + FN);
    System.out.print(" FP= " + FP);
    System.out.println(" TN= " + TN);
    System.out.println(c + " precision: " + precision);
    System.out.println(c + " recall: " + recall);
    System.out.println(c + " f1: " + f1);
    System.out.println();
}
```

```
                double precision = totalTP / (totalTP + totalFP);
                double recall = totalTP / (totalTP + totalFN);
                double f1 = 2 * precision * recall / (precision + recall);
                System.out.print("Total TP= " + totalTP);
                System.out.print("Total FN= " + totalFN);
                System.out.print("Total FP= " + totalFP);
                System.out.println("Total TN= " + totalTN);
                System.out.println();

                System.out.println("微平均");
                System.out.println("Precision: " + precision);
                System.out.println("Recall: " + recall);
                System.out.println("F1: " + f1);
                System.out.println();
                System.out.println("宏平均");
                System.out.println("Precision: " + totalPrecision / docTypeList.length);
                System.out.println("Recall: " + totalRecall / docTypeList.length);
                System.out.println("F1: " + totalF1 / docTypeList.length);

            } catch (Exception ex) {
                log.error(ex.getMessage());
            } finally { //  确保发生异常时关闭 reader
                IOUtils.closeStream(reader);
            }
        }

        @Override
        public int run(String[] strings) throws Exception {
            doEvaluation();
            System.out.println("已计算测试集中各文档的贝叶斯分类结果");
            return 0;
        }
}
```

### 4.8 Main

```
import job.*;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.util.ToolRunner;
import utils.Const;

public class Main {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
```

```
        conf.set("INPUT_PATH", Const.TRAIN_DATA_INPUT_PATH);
        conf.set("OUTPUT_PATH", Const.TRAIN_DATA_SEQUENCE_FILE_PATH);

        //将训练集的.txt 输入文件生成一个 SequenceFile
        InitSequenceFileJob initSequenceFileJob = new InitSequenceFileJob();
        ToolRunner.run(conf, initSequenceFileJob, args);

        // 根据 InitSequenceFileJob 输出的 SequenceFile 统计每个文档类别有多少个文档
        GetDocCountFromDocTypeJob getDocCountFromDocTypeJob = new
GetDocCountFromDocTypeJob();
        ToolRunner.run(conf, getDocCountFromDocTypeJob, args);

        // 根据 InitSequenceFileJob 输出的 SequenceFile 计算
        // 每个文档类别中每个单词出现的次数
        GetSingleWordCountFromDocTypeJob getSingleWordCountFromDocTypeJob = new
GetSingleWordCountFromDocTypeJob();
        ToolRunner.run(conf, getSingleWordCountFromDocTypeJob, args);

        // 根据 GetSingleWordCountFromDocTypeJob 输出的 SequenceFile 计算
        // 每个文档类型的总单词数用于条件概率计算
        GetTotalWordCountFromDocTypeJob getTotalWordCountFromDocTypeJob = new
GetTotalWordCountFromDocTypeJob();
        ToolRunner.run(conf, getTotalWordCountFromDocTypeJob, args);



        // 运行测试集数据
        conf = new Configuration();
        conf.set("INPUT_PATH", Const.TEST_DATA_INPUT_PATH);
        conf.set("OUTPUT_PATH", Const.TEST_DATA_SEQUENCE_FILE_PATH);
        conf.set("DOC_TYPE_LIST", Const.DOC_TYPE_LIST);

        // 与训练集相同，将测试集多个文件生成一个 SequenceFile
        initSequenceFileJob = new InitSequenceFileJob();
        ToolRunner.run(conf, initSequenceFileJob, args);

        // 读取之前所有任务输出的 SequenceFile 到内存中并在 Setup 中计算训练集的先验
概率、条件概率
        // 读取 InitSequenceFileJob 生成的测试集的 SequenceFile 计算测试集的每个文档分
成每一类的概率
        GetNaiveBayesResultJob getNaiveBayesResultJob = new GetNaiveBayesResultJob();
        ToolRunner.run(conf, getNaiveBayesResultJob, args);

        // 运行 Evaluation 程序，对各文档的贝叶斯分类结果进行评估，计算各文档 FP、
TP、FN、TN、
```

```
        // Precision、Recall、F1 以及整体的宏平均、微平均。
        Evaluation evaluation = new Evaluation();
        ToolRunner.run(conf, evaluation, args);
    }
}
```

## 五：数据集说明

我在数据集中选择了 Country 文件夹下的 CHINA 和 CANA 作为本次实验的样本，其中 CHINA 类中包含 255 个文本，CANA 类中包含 263 个文本。按照 70% 与 30% 的比例选取训练集和测试集。表格如下：

表 1 实验训练集与测试集数量表

| | CHINA | CANA |
|---|---|---|
| 文档总数 | 255 | 263 |
| 训练集数 | 178 | 184 |
| 测试集数 | 77 | 79 |

## 六：程序运行说明

该项目一共要运行 6 个 Map 和 Reduce 任务，具体如下

### 6.1 InitSequenceFileJob

两个 InitSequenceFileJob 分别是对测试机和训练集文件进行序列化操作，将.txt 文件输出为 SequenceFile。

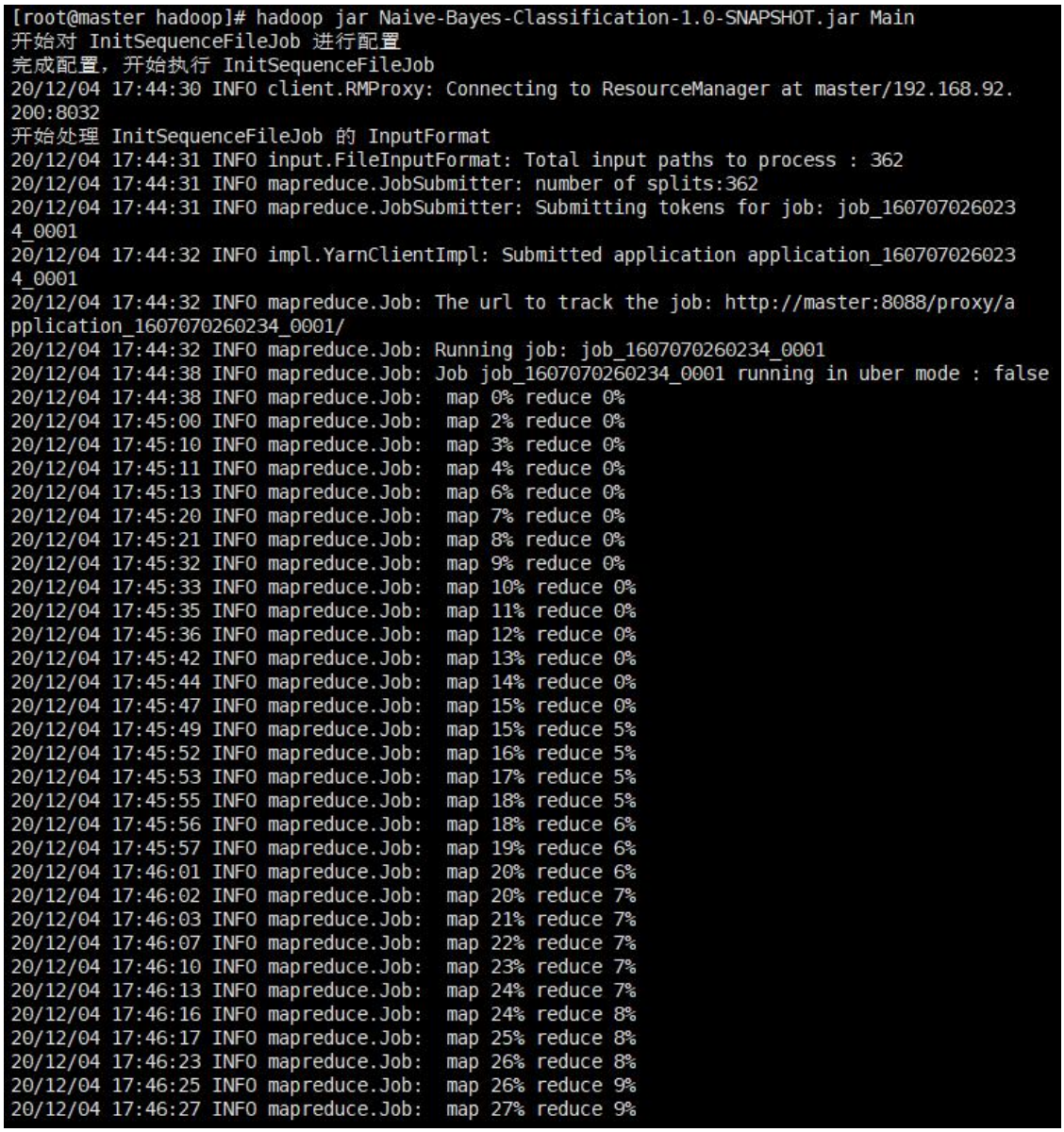


图 6 训练集的 InitSequenceFileJob 任务运行截图

### 6.2 GetDocCountFromDocTypeJob

GetDocCountFromDocTypeJob 有 1 个 Map 任务和 1 个 Reduce 任务，根据 InitSequenceFileJob 输出的 sequence_file 经过 Map 和 Reduce 后统计每个 DocType 有多少个文档。

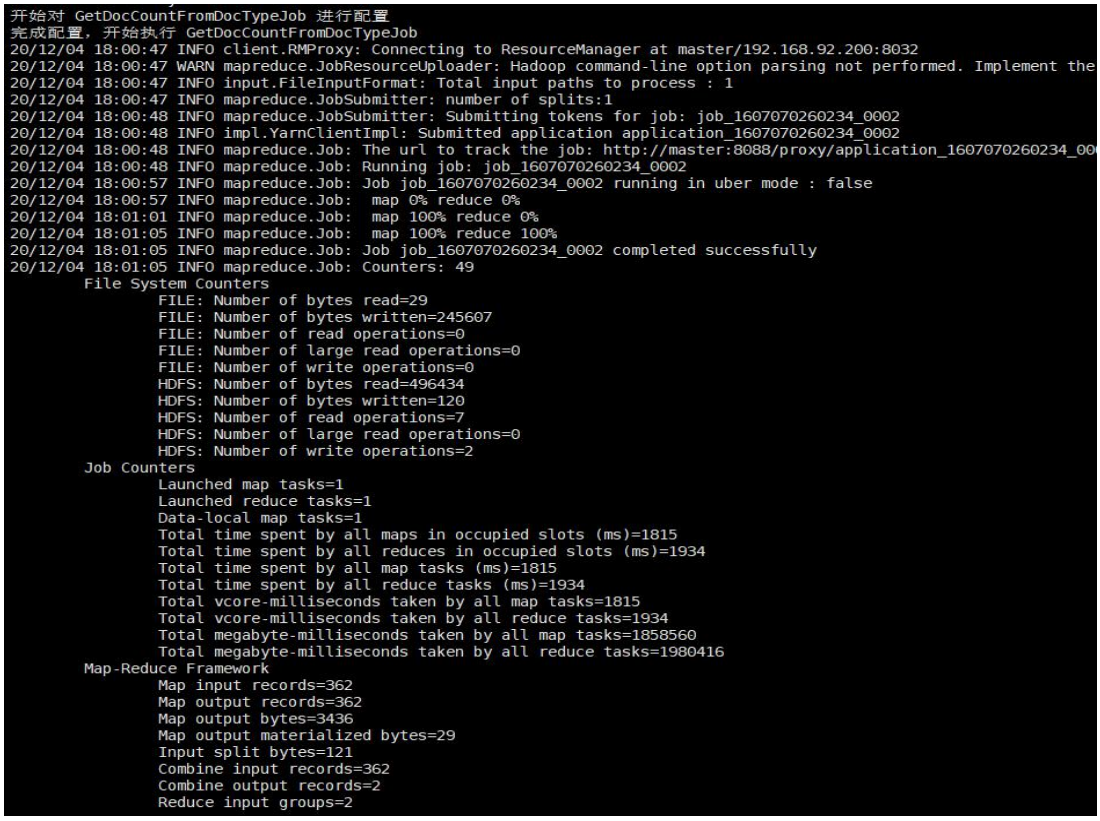


图 7 GetDocCountFromDocTypeJob 任务运行截图

### 3、GetSingleWordCountFromDocTypeJob

GetSingleWordCountFromDocTypeJob 有 1 个 Map 任务和 1 个 Reduce 任务，根据 InitSequenceFileJob 输出的 sequence_file 统计每个单词在每个文档类别中出现的次数。

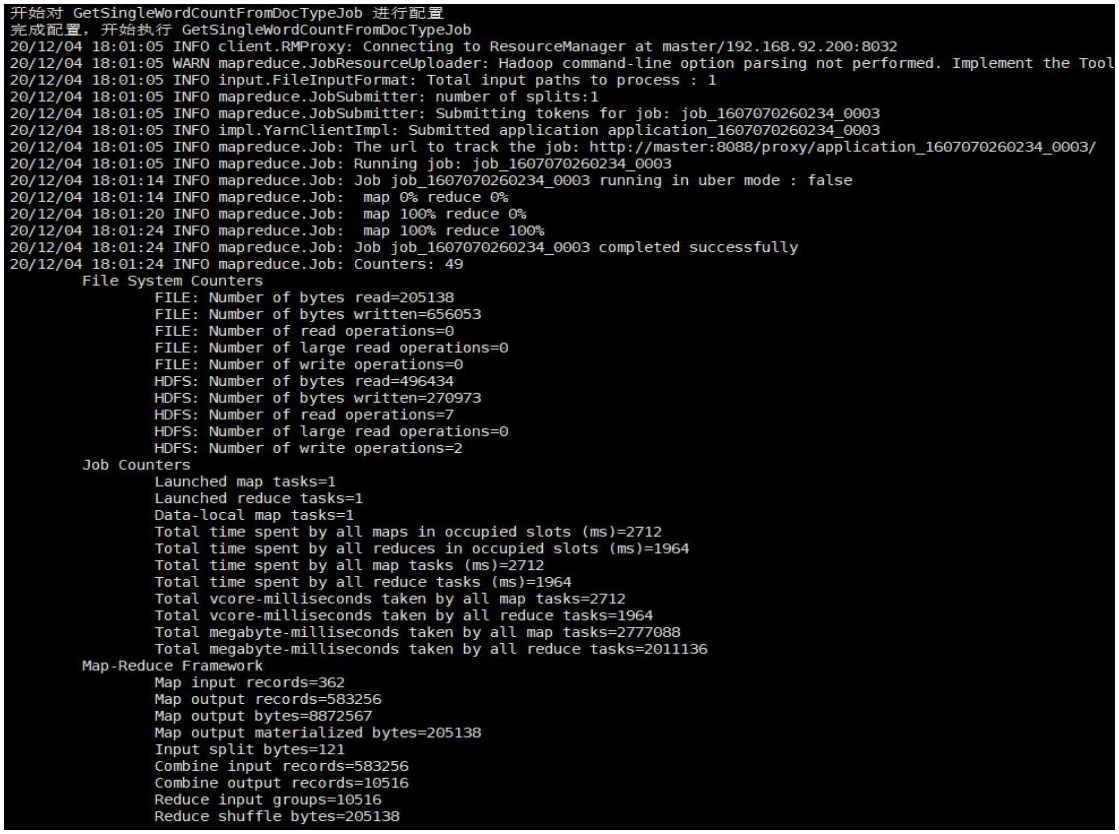


图 8 GetSingleWordCountFromTypeJob 任务运行截图

### 4、GetTotalWordCountFromDocTypeJob

GetTotalWordCountFromDocTypeJob 有 1 个 Map 任务和 1 个 Reduce 任务，根据 GetSingleWordCountFromDocTypeJob 输出的 sequence_file 统计每个文档类别的总单词数。

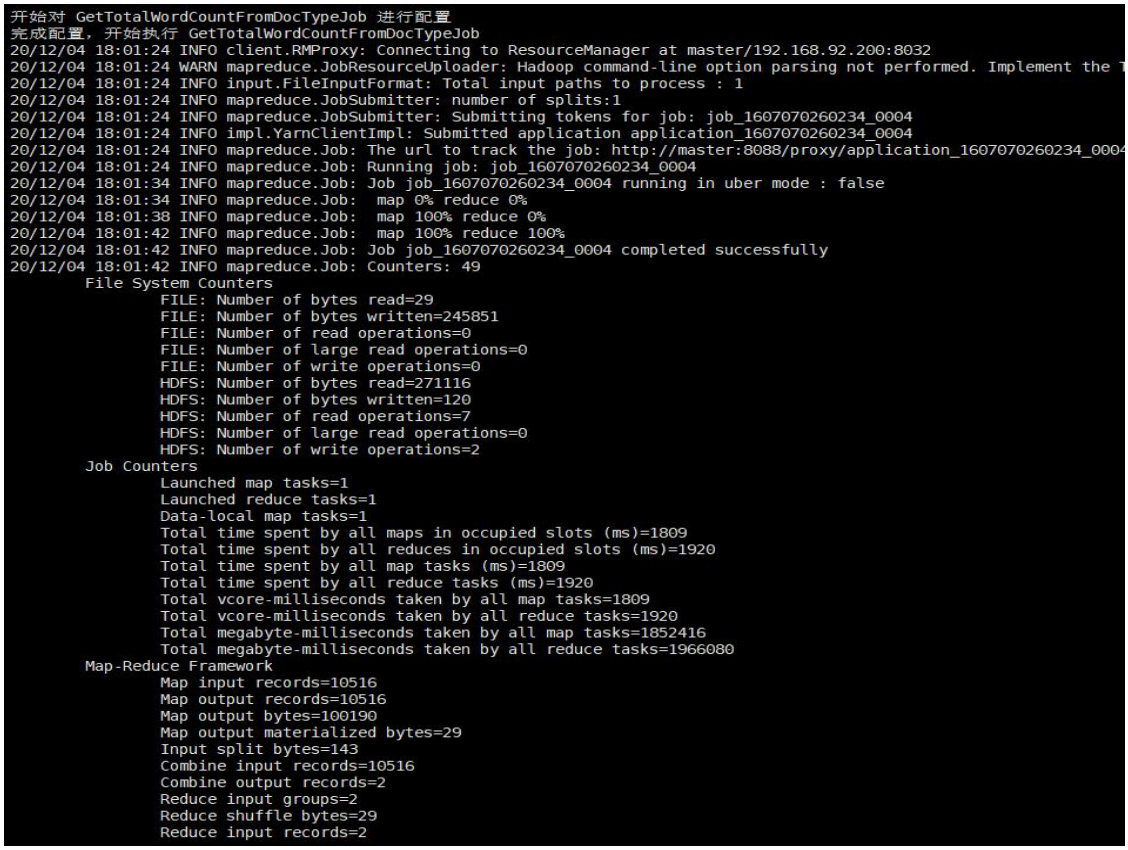


图 9 GetTotalWordCountFromDocTypeJob 任务运行截图

## 5、GetNaiveBayesResultJob

GetNaiveBayesResultJob 有 1 个 Map 任务和 1 个 Reduce 任务，读取 InitSequenceFileJob 生成的测试集的 sequence_file 计算测试集的每个文档分成每一类的概率。

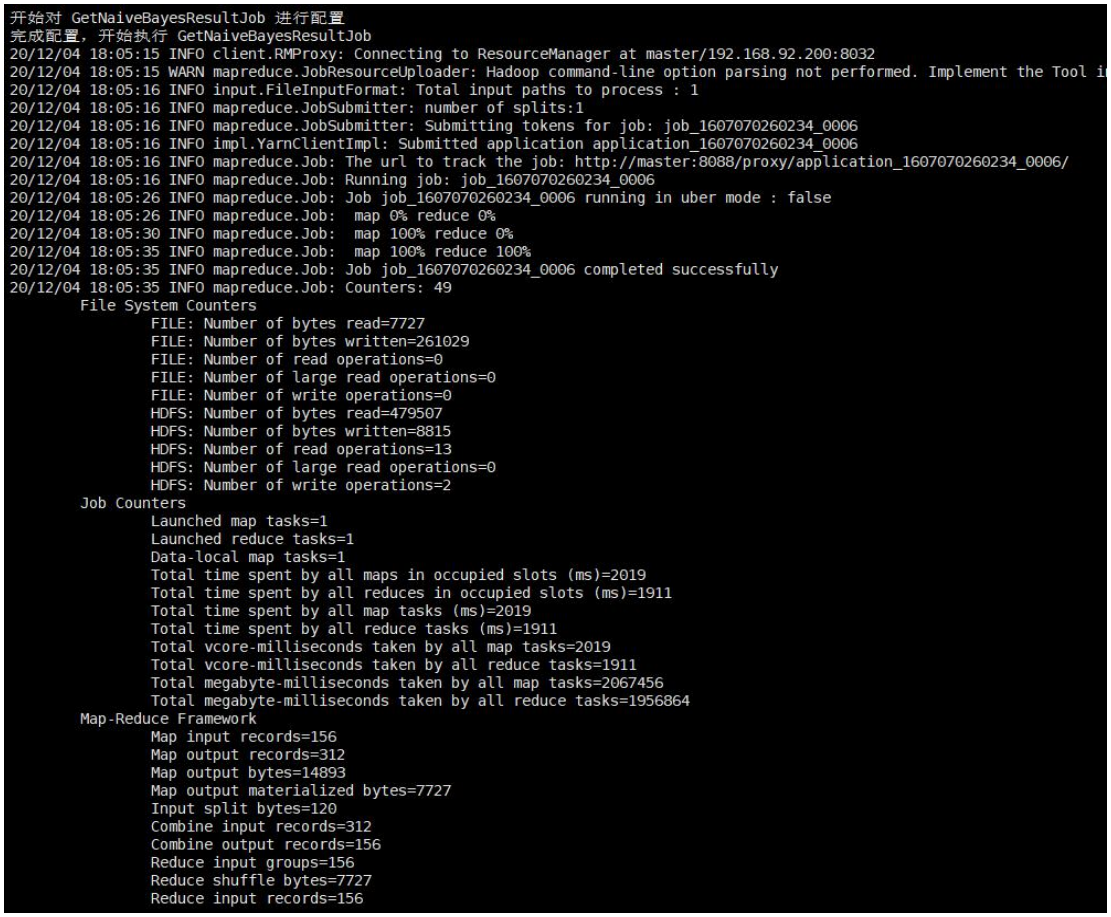


图 10 GetNaiveBayesResultJob 任务运行截图

## 6、Evaluation

Evaluation 程序为单机程序，因此没有 Map 和 Reduce 任务，该程序对各文档的贝叶斯分类结果进行评估，计算各文档 FP、TP、FN、TN、Precision、Recall、F1 以及整体的宏平均、微平均。

```
CANA TP= 79.0 FN= 0.0 FP= 40.0 TN= 37.0
CANA precision: 0.6638655462184874
CANA recall: 1.0
CANA f1: 0.797979797979798

CHINA TP= 37.0 FN= 40.0 FP= 0.0 TN= 79.0
CHINA precision: 1.0
CHINA recall: 0.4805194805194805
CHINA f1: 0.6491228070175439

Total TP= 116.0Total FN= 40.0Total FP= 40.0Total TN= 116.0

微平均
Precision: 0.7435897435897436
Recall: 0.7435897435897436
F1: 0.7435897435897437

宏平均
Precision: 0.8319327731092436
Recall: 0.7402597402597403
F1: 0.723551302498671
已计算测试集中各文档的贝叶斯分类结果
```

图 11 Evaluation 程序运行截图

## 7、Web 页面的作业监控截图

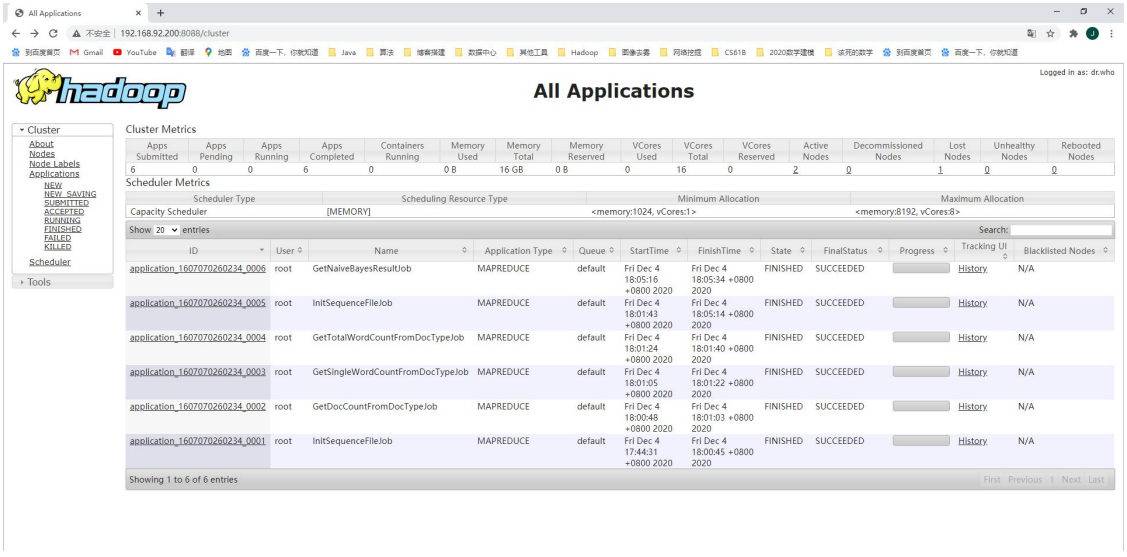


图 12 WEB 界面作业监控截图

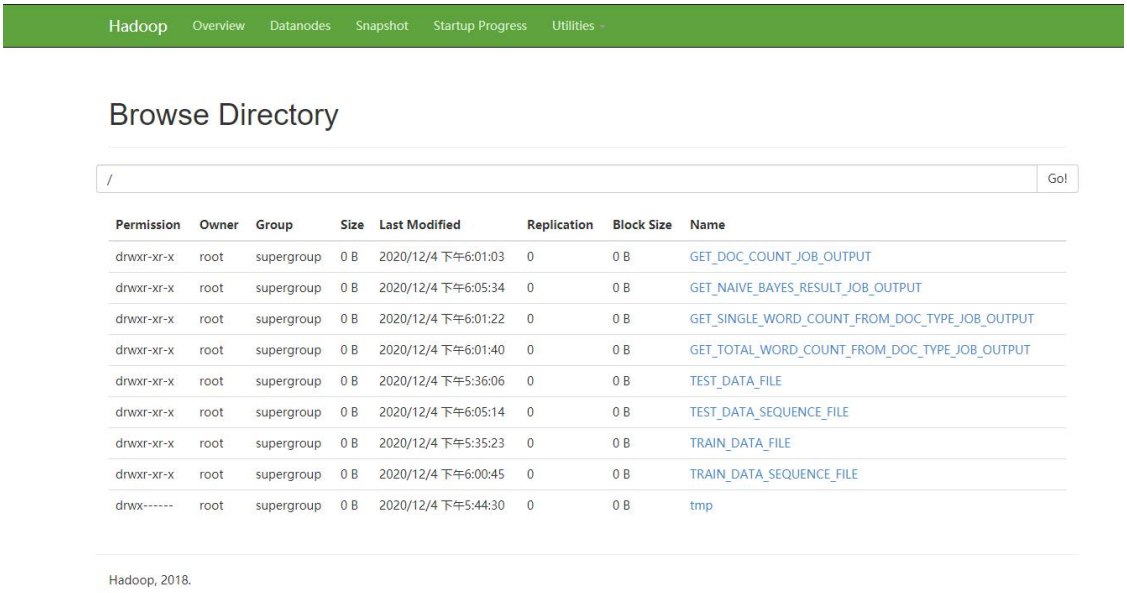


图 13 WEB 界面任务计算结果

## 七：实验结果分析

将上述的实验结果进行整理如下：

表 2 CANA 类的分类结果邻接表（Confusion Matrix）

| CANA | Yes(Ground Truth) | No(Ground Truth) |
|---|---|---|
| Yes(Classified) | 79 | 40 |
| No(Classified) | 0 | 37 |

表 3 CHINA 类的分类结果邻接表（Confusion Matrix）

| CHINA | Yes(Ground Truth) | No(Ground Truth) |
|---|---|---|
| Yes(Classified) | 37 | 0 |
| No(Classified) | 40 | 79 |

其中 CANA 的准确率为 0.6638655，召回率为 1，F1 值为 0.797979，CHINA 的准确率为 1，召回率为 0.48051948，F1 值为 0.649122807。

微平均的计算结果为：

Precision： 0.7435897435897436

Recall： 0.7435897435897436

F1： 0.7435897435897437

宏平均的计算结果为：

Precision： 0.8319327731092436

Recall： 0.7402597402597403

F1： 0.723551302498671