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

1. 朴素贝叶斯基本方法

假设X是定义在输入空间上的随机向量，Y是定义在输出空间上的随机变量。P(X, Y)是联合概率分布。训练数据集

$$\mathrm{T} = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

由P(X, Y)独立同分布产生。在本次工程中，对于每篇文档就是上述对应的特征向量x_i，并且文档中的每个单词对应为特征向量的特征。

朴素贝叶斯法通过训练数据集学习联合概率分布P(X, Y)。具体需要学习先验概率分布以及条件概率分布。先验概率分布

$$\mathrm{P}(\mathrm{Y} = c_k), \mathrm{k} = 1,2, \dots, \mathrm{K}$$

c_k对应了文档的类别。条件概率分布

$$\mathrm{P}(\mathrm{X} = \mathrm{x}|\mathrm{Y} = c_k), \mathrm{k} = 1,2, \dots, \mathrm{K}$$

即求在已知分类的条件下该文档出现的概率，最后得到联合概率分布P(X, Y)。

朴素贝叶斯法对条件作了条件独立性的假设，条件独立假设说明用于分类的特征在类别确定的条件下都是条件独立的。

$$\mathrm{P}(\mathrm{X} = \mathrm{x}|\mathrm{Y} = c_k) = \mathrm{P}\left(X^{(1)} = x^{(1)}, \dots, X^{(n)} = x^{(n)} \middle| \mathrm{Y} = c_k\right) = \prod_{j=1}^{n} P(X^{(j)} = x^{(j)}|\mathrm{Y} = c_k)$$

最后后验概率可以通过贝叶斯定理进行，并将后验概率最大的类别作为输出。

$$\mathrm{P}(\mathrm{Y} = c_k|\mathrm{X} = \mathrm{x}) = \frac{P(X = x|Y = c_k)P(Y = c_k)}{P(x)}$$

由于P(x)对所有类别都是一样的，所以朴素贝叶斯分类可表示为

$$\mathrm{y} = argmax_{c_k} P(Y = c_k) \prod_j P(X^{(j)} = x^{(j)}|Y = c_k)$$

2. 朴素贝叶斯算法

输入：训练数据$\mathrm{T} = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$，其中$x_i = \left(x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(n)}\right)^T$，$x_i^{(j)}$是第i个样本的第j个特征，$x_i^{(j)} \epsilon \left\{a_{j1}, a_{j2}, \dots, a_{jS_j}\right\}$，$a_{jl}$是第j个特征可能取的第l个值，$\mathrm{j} = 1,2, \dots \mathrm{n}$，$\mathrm{l} = 1,2, \dots, S_j$，$y_i \in \{c_1, c_2, \dots, c_k\}$；实例x；

输出：实例x的分类。

（1） 计算先验概率及条件概率

$$\mathrm{P}(\mathrm{Y} = c_k) = \frac{\sum_{i=1}^{N} I(y_i = c_k)}{N}$$

$$\mathrm{P}\left(X^{(j)}=a_{jl}\middle|\mathrm{Y}=c_k\right)=\frac{\sum_{i=1}^{N}I(x_i^{(j)}=a_{jl},y_i=c_k)}{\sum_{i=1}^{N}I(y_i=c_k)}$$

其中I是指示函数。

（2） 对于给定实例x，计算

$$P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)}|Y=c_k)$$

（3） 确定实例x的类别

$$\mathrm{y}=argmax_{c_k}P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)}|Y=c_k)$$

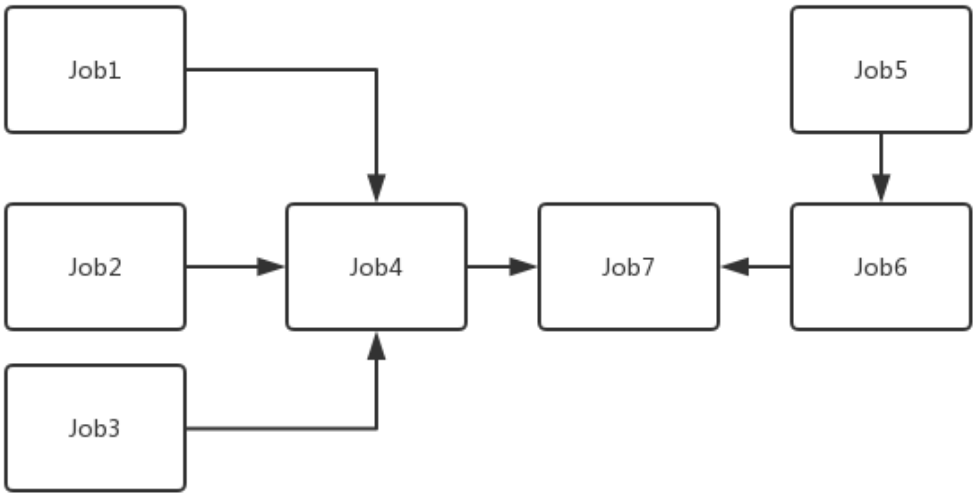
3. 拉普拉斯平滑

在上述的算法中，计算条件概率可能会出现概率值为 0 的情况，这时会影响到后验概率的计算结果，使得分类产生偏差，因此在本次工程中采用拉普拉斯平滑来处理。主要是对条件概率进行修改。

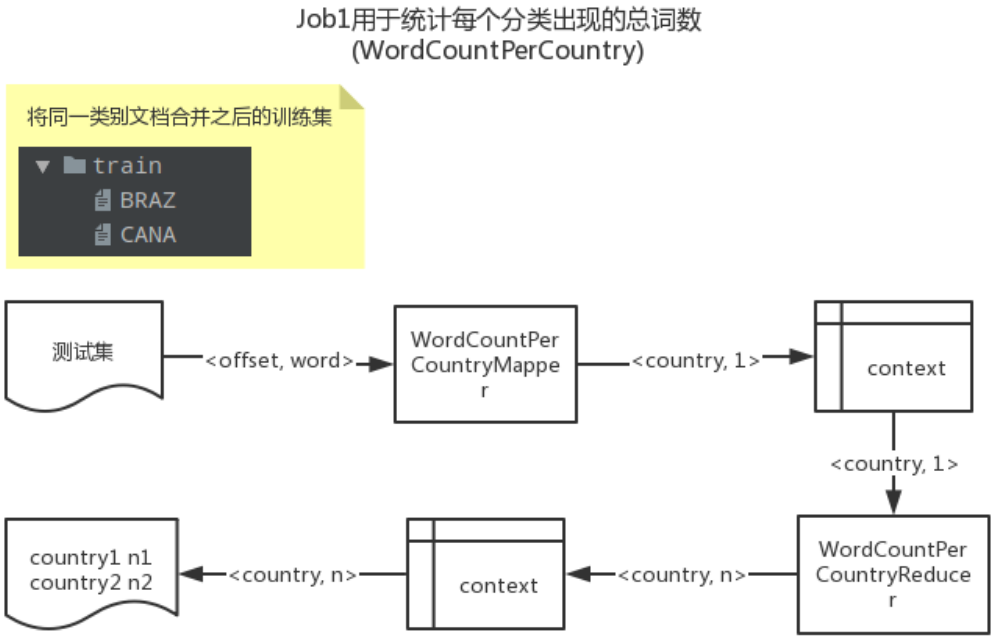
$$\mathrm{P}\left(X^{(j)}=a_{jl}\middle|\mathrm{Y}=c_k\right)=\frac{\sum_{i=1}^{N}I(x_i^{(j)}=a_{jl},y_i=c_k)+1}{\sum_{i=1}^{N}I(y_i=c_k)+S_j}$$

三 贝叶斯分类器训练的 MapReduce 算法设计

1. 在工程中一个设计了 7 个 Job 用于完成朴素贝叶斯的训练以及测试，其中 Job1-6 用于训练，Job7 用于测试。在训练部分分为两个阶段，第一阶段是由 Job1-4 完成，用于计算条件概率分布，第二阶段是由 Job5-6 完成，用于计算每个类别的先验概率。这 7 个 Job 有 JobControl 进行统一管理，其依赖关系如下图所示：



2. Job1 是用于统计每个分类出现的总词数，其 Data Flow 如下所示：

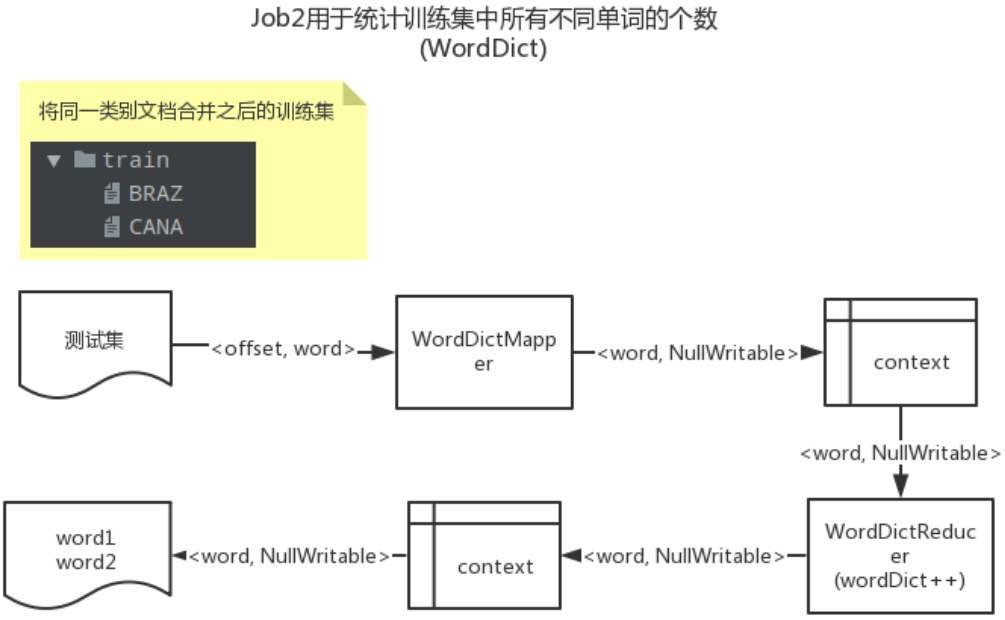


（1）Job1 的输入是根据类别合并之后的训练数据集，将其合并的主要原因是为了减少小文件的个数,减少 Mapper 的个数，从而提交 MapReduce 的运行速度，如下图所示：

```
new
york
new
jersey
metrostars
added
brazilian
midfielder
guido
```

（2）Job1 的输入数据格式是 TextInputFormat
输出的 key/value 的类型是<LongWritable, Text>，其中 key 表示每行起始字节偏移量，value 表示每行数据，即一个单词。
（3）Mapper 的格式为 Mapper<LongWritable, Text, Text, IntWritable>
输出的 key/value 类型是<Text, IntWritable>,其中 key 表示类别名,通过 split 获取当前路径来获取，value 始终为 1，表示该类别下的一个单词。
（4）Reducer 的格式为 Reducer<Text, IntWritable, Text, IntWritable>
输出的 key/value 类型是<Text, IntWritable>，其中 key 表示类别名，value 是对同一类别名下所有单词统计的结果。

3. Job2 用于统计训练集中所有不同单词的个数，用于拉普拉斯平滑，其 Data Flow 如下：



（1）Job2 的输入是根据类别合并之后的训练数据集，与 Job1 相同。

（2）Job2 的输入数据格式是 TextInputFormat

输出的 key/value 的类型是<LongWritable, Text>，其中 key 表示每行起始字节偏移量，value 表示每行数据，即一个单词。

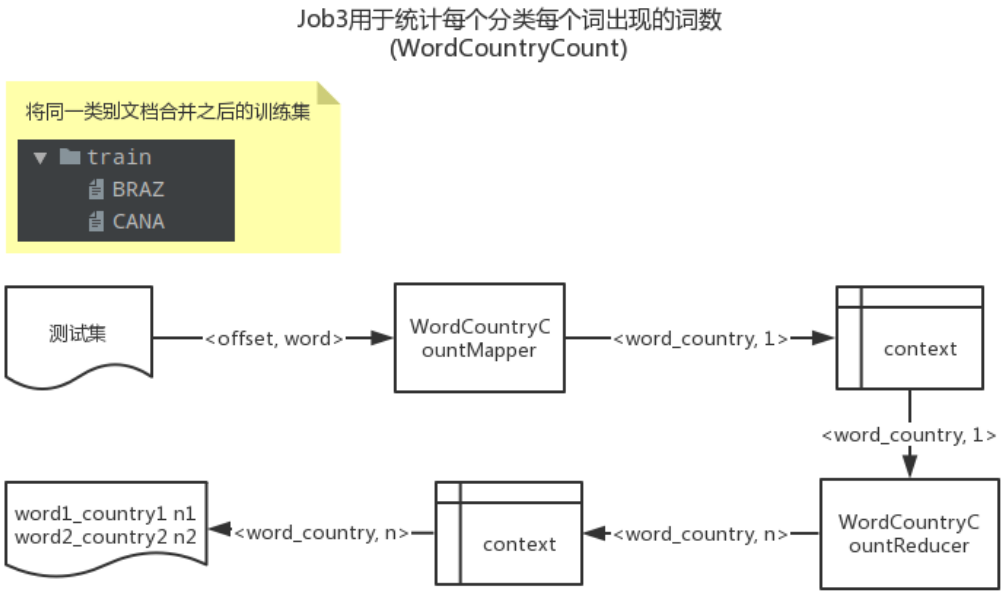
（3）Mapper 的格式为 Mapper<LongWritable, Text, Text, NullWritable>

输出的 key/value 的类型是<Text, NullWritable>,其中 key 是输入的 value 即 key 为单词，value 为 NullWritable。

（4）Reducer 的格式为 Reducer<Text, NullWritable, Text, NullWritable>

输出的 key/value 的类型是<Text, NullWritable>，其中 key 是输入的 value 即 key 为单词，value 为 NullWritable，只记录不同的单词。

4. Job3 用于统计每个分类每个词出现的次数，其 Data Flow 如下：



（1）Job3 的输入是根据类别合并之后的训练数据集，与 Job1 相同。

（2）Job3 的输入数据格式是 TextInputFormat

输出的 key/value 的类型是<LongWritable, Text>，其中 key 表示每行起始字节偏移量，value 表示每行数据，即一个单词。

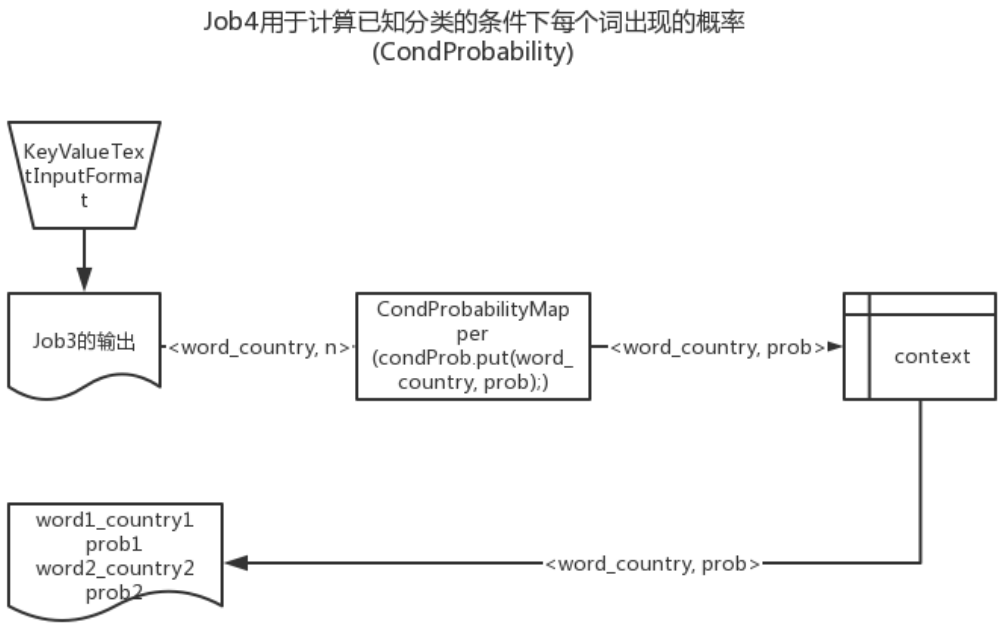
（3）Mapper 的格式为 Mapper<LongWritable, Text, Text, IntWritable>

输出的 key/value 的类型是<Text, IntWritable>，其中 key 是有单词+类别组成，中间由'_'分隔，value 是对 key 的计数。

（4）Reducer 的格式为 Reducer<Text, IntWritable, Text, IntWritable>

输出的 key/value 的类型是<Text, IntWritable>，其中 key 仍是单词+类别组成，中间由'_'分隔的，value 是对 key 的总数统计。

5. Job4 是用于计算已知分类的条件下每个词出现的概率，其 Data Flow 如下：

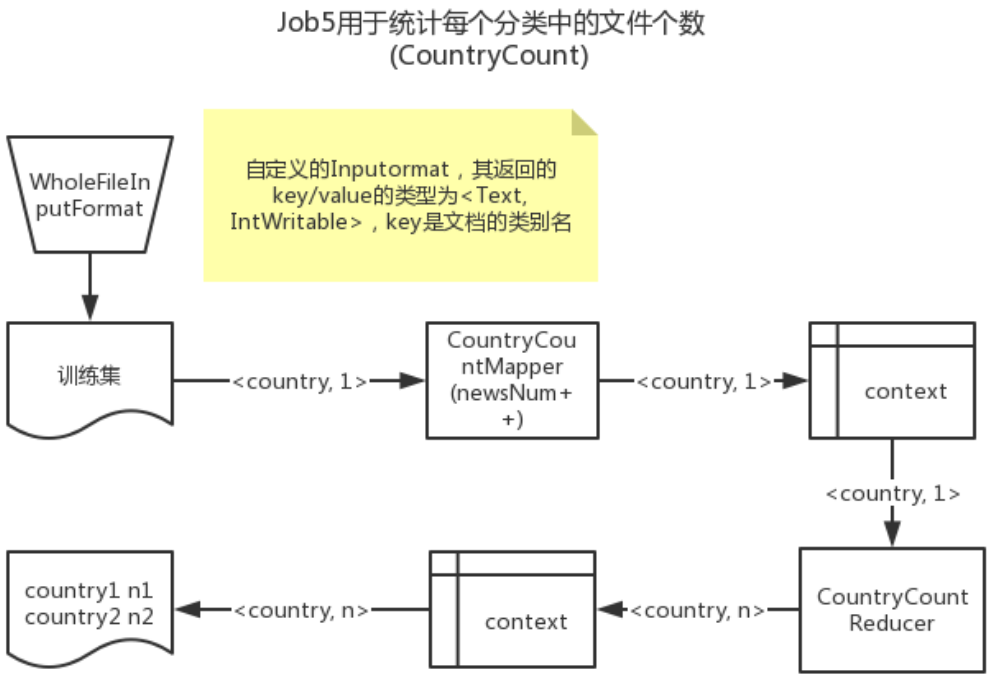


（1）Job4 的输入是 Job3 的输出结果，其具体格式如下图所示：

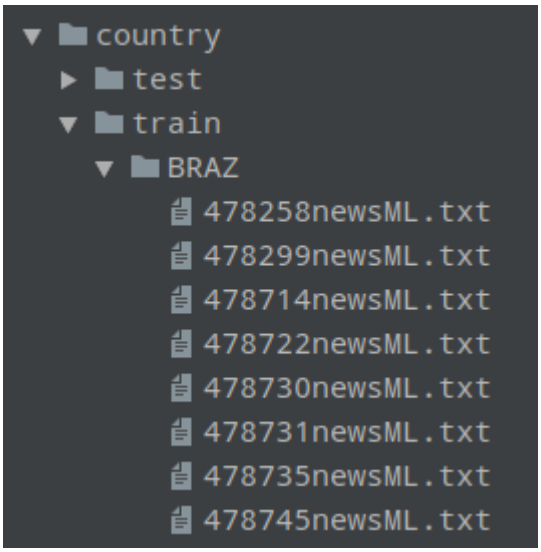
```
0-8_CANA          1
0.00group_BRAZ    2
0.01_BRAZ         1
0.02_CANA         1
0.03_CANA         3
0.05_BRAZ         2
0.05_CANA         3
0.06_BRAZ         1
```

（2）Job4 的输入数据格式是 KeyValueTextInputFormat
输出的 key/value 的类型是<Text, Text>，其将文档中的一行数据根据第一个空格进行分割，前面部分作为 key，后面作为 value，其中 key 是单词+类别，value 是其出现的次数。
（3）Mapper 的格式为 Mapper<Text, Text, Text, DoubleWritable>
输出的 key/value 的类型是<Text, DoubleWritable>，其中 key 仍是输入的 key，即是单词+类别，value 表示在该类别下该单词出现的条件概率。
（4）Job4 没有 Reducer，Mapper 的结果直接作为输出。

6. Job5 用于统计每个分类中的文档个数，其 Data Flow 如下：



（1）Job5 的输入是原始的训练数据集，其组织结构如下图所示：



（2）Job5 的输入数据格式是自定义的 WholeFileInputFormat，其将整个文件作为一个 split，所以可以针对每个文件获取其类别

输出的 key/value 的类型是<Text, IntWritable>，其中 key 表示的是文档所属的类别名称，value 是对文档的计数。

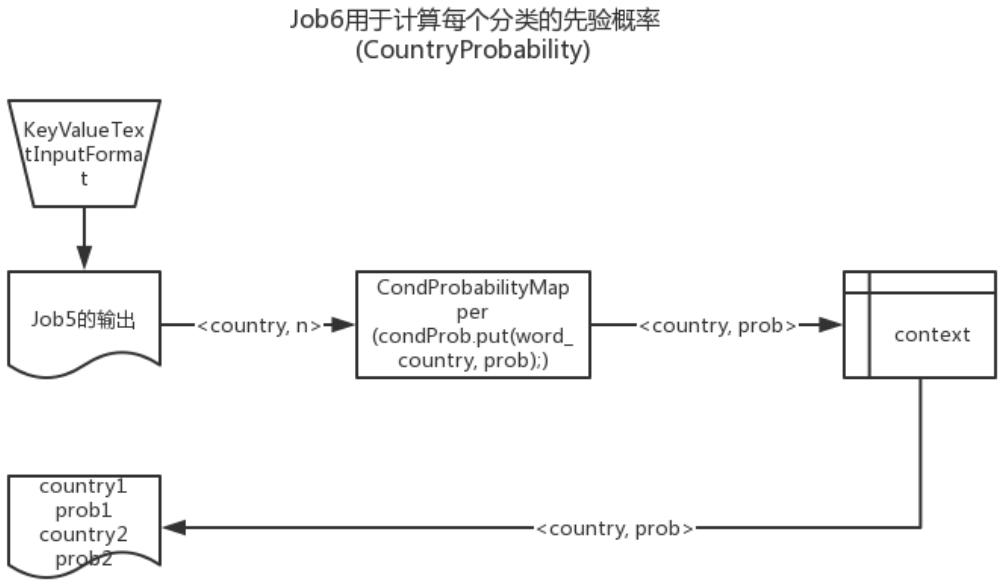
（3）Mapper 的格式为 Mapper<Text, IntWritable, Text, IntWritable>

输出的 key/value 的类型是<Text,IntWritable>，在 Mapper 中没有对输入数据进行处理而是直接将其输出，在 Mapper 对文档总数进行了计数，故 key 表示的是文档所属的类别名称，value 的值仍是 1。

（4）Reducer 的格式为 Reducer<Text, IntWritable, Text, IntWritable>

输出的 key/value 的类型是<Text, IntWritable>，其中 key 表示的是文档所属的类别名称，value 表示该类别下文档的总数。

7. Job6 用于计算每个分类的先验概率，其 Data Flow 如下：

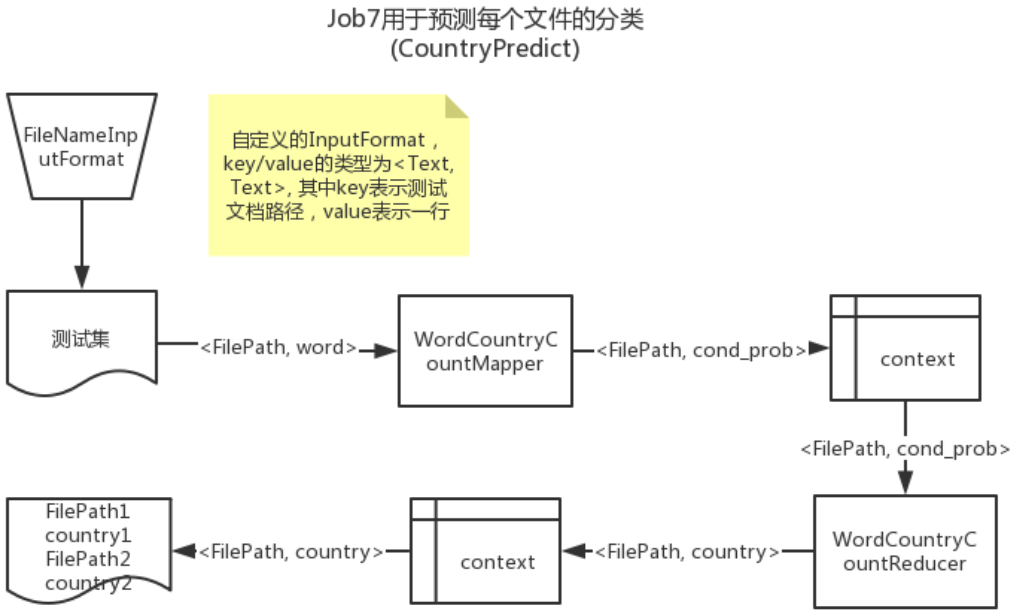


（1）Job6 的输入是 Job5 的输出，其具体格式如下图所示：

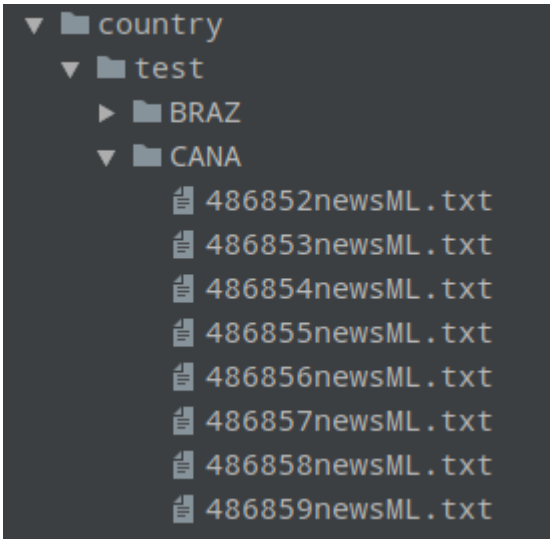
```
BRAZ    160
CANA    160
```

（2）Job6 的输入数据格式是 KeyValueTextInputFormat
输出的 key/value 的类型是<Text, Text>，其将文档中的一行数据根据第一个空格进行分割，前面部分作为 key，后面作为 value，其中 key 是类别，value 是该类别下的文档总数。
（3）Mapper 的格式为 Mapper<Text, Text, Text, DoubleWritable>
输出的 key/value 的类型是<Text, DoubleWritable>，其中 key 表示的类别的名称，value 是该类别的先验概率。
（4）Job6 没有 Reducer，Mapper 的结果直接作为输出。

8. Job7 用于预测每个文件的分类，其 Data Flow 如下：



（1）Job7 的输入是测试集，其具体组织结构如下图所示：



（2）Job7 的输入数据格式是自定义的 FileNameInputFormat.

输出的 key/value 的类型是<Text, Text>，其中 key 表示当前文档的路径名称，因为其路径名称包含了类型，方便后面进行评估，value 是测试文档中的一行数据，即一个单词。

（3）Mapper 的格式为 Mapper<Text, Text, Text, MapWritable>

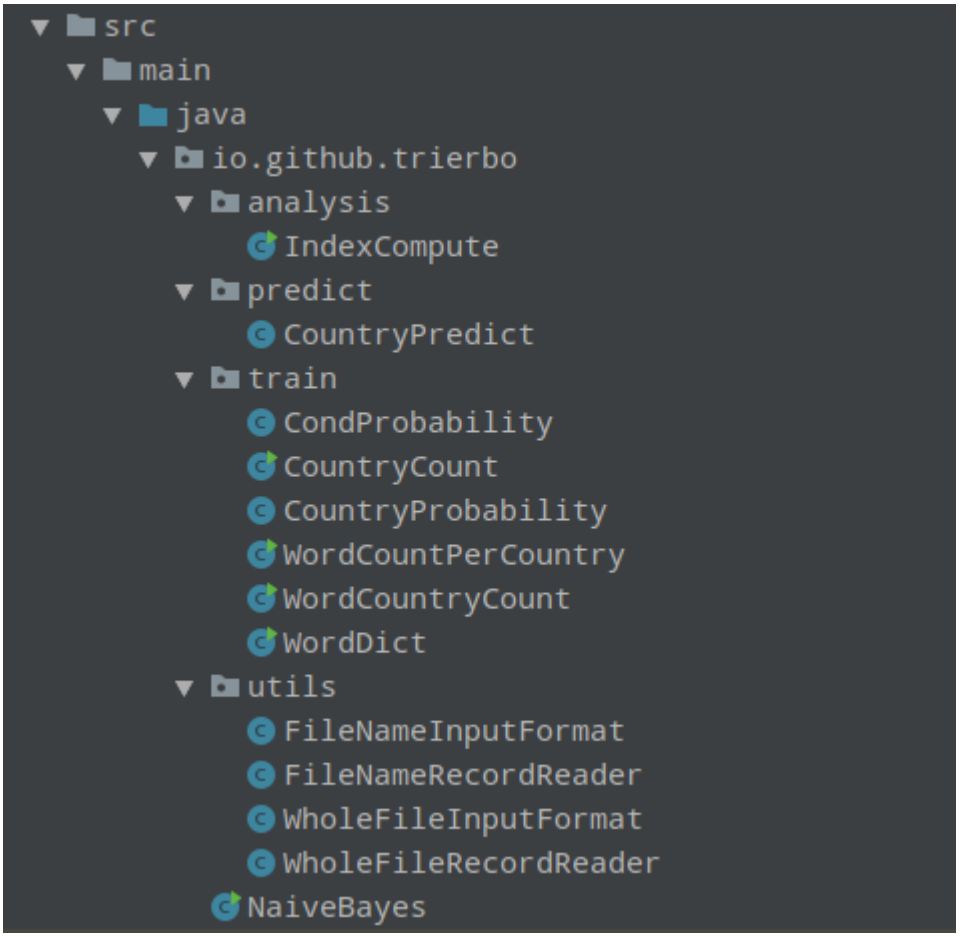
输出的 key/value 的类型是<Text, MapWritable>，其中 key 表示当前文档的路径名称，value 记录了当前单词在各个类别下的条件概率。

（4）Reducer 的格式为 Reducer<Text, MapWritable, Text, Text>

输出的 key/value 的类型是<Text, Text>，其中 key 表示当前文档的路径名称，value 表示预测的类别。

四　源代码清单

项目代码的目录结构如下图所示，其中 train 表示用于训练的代码，主要计算条件概率以及先验概率，utils 中是自定义的输入数据格式，predict 是用于对测试集进行预测，analysis 是用于准确率、召回率以及 F1 值的计算。



具体源代码如下:

NaiveBayes.java

```
package io.github.trierbo;

import io.github.trierbo.predict.CountryPredict;
import io.github.trierbo.train.*;
import io.github.trierbo.utils.FileNameInputFormat;
import io.github.trierbo.utils.WholeFileInputFormat;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.lib.jobcontrol.ControlledJob;
import org.apache.hadoop.mapreduce.lib.jobcontrol.JobControl;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

```
import java.util.HashMap;

/**
 *  贝叶斯公式为:
 * P(C|AB) ~ P(A|C)*P(B|C)*P(C)
 */
public class NaiveBayes {
    // 记录每个类别的总词数, key 表示国家名称, value 表示总词数
    public static HashMap<String, Integer> wordPerCountry = new HashMap<>();
    // 用于拉普拉斯平滑
    public static int wordDict = 0;
    // 训练集中文件总数, 用于计算每个类别的概率
    public static int newsNum = 0;
    // 每个类别下某个单词出现的概率
    public static HashMap<String, Double> condProb = new HashMap<>();
    // 每个类别出现的概率
    public static HashMap<String, Double> countryProb = new HashMap<>();

    /**
     * args[0] 将训练集相同类别下所有文档合并之后的路径
     *  args[1] 未作处理的训练集路径, 用于统计每个类别的文档个数, 方便计算类别的先验概率
     * args[2] Job1 的输出路径
     * args[3] Job2 的输出路径
     * args[4] Job3 的输出路径, 以及 Job4 的输入路径
     * args[5] Job4 的输出路径
     * args[6] Job5 的输出路径, 以及 Job6 的输入路径
     * args[7] Job6 的输出路径
     * args[8] 测试集的路径
     * args[9] 预测结果的输出路径
     */
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        conf.set("mapreduce.ifile.readahead", "false");

        // Job1 用于统计每个分类出现的总词数
        Job job1 = Job.getInstance(conf, NaiveBayes.class.getSimpleName() + '1');
        // Job2 用于统计训练集中所有不同单词的个数, 用于拉普拉斯平滑
        Job job2 = Job.getInstance(conf, NaiveBayes.class.getSimpleName() + '2');
        // Job3 用于统计每个分类每个词出现的次数
        Job job3 = Job.getInstance(conf, NaiveBayes.class.getSimpleName() + '3');
        // Job4 用于计算已知分类的条件下每个词出现的概率
        Job job4 = Job.getInstance(conf, NaiveBayes.class.getSimpleName() + '4');
        // Job5 用于统计每个分类中的文件个数
```

```
Job job5 = Job.getInstance(conf, NaiveBayes.class.getSimpleName() + '5');
// Job6 用于计算每个分类的先验概率
Job job6 = Job.getInstance(conf, NaiveBayes.class.getSimpleName() + '6');
// Job7 用于预测每个文件的分类
Job job7 = Job.getInstance(conf, NaiveBayes.class.getSimpleName() + '7');

job1.setJarByClass(NaiveBayes.class);
job1.setMapperClass(WordCountPerCountry.WordCountPerCountryMapper.class);
job1.setReducerClass(WordCountPerCountry.WordCountPerCountryReducer.class);
job1.setOutputKeyClass(Text.class);
job1.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPaths(job1, args[0]);
FileOutputFormat.setOutputPath(job1, new Path(args[2]));

job2.setJarByClass(NaiveBayes.class);
job2.setMapperClass(WordDict.WordDictMapper.class);
job2.setReducerClass(WordDict.WordDictReducer.class);
job2.setOutputKeyClass(Text.class);
job2.setOutputValueClass(NullWritable.class);
FileInputFormat.addInputPaths(job2, args[0]);
FileOutputFormat.setOutputPath(job2, new Path(args[3]));

job3.setJarByClass(NaiveBayes.class);
job3.setMapperClass(WordCountryCount.WordCountryCountMapper.class);
job3.setReducerClass(WordCountryCount.WordCountryCountReducer.class);
job3.setOutputKeyClass(Text.class);
job3.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPaths(job3, args[0]);
FileOutputFormat.setOutputPath(job3, new Path(args[4]));

job4.setJarByClass(NaiveBayes.class);
job4.setMapperClass(CondProbability.CondProbabilityMapper.class);
// 不进行 reduce 操作
job4.setNumReduceTasks(0);
job4.setOutputKeyClass(Text.class);
job4.setOutputValueClass(DoubleWritable.class);
job4.setInputFormatClass(KeyValueTextInputFormat.class);
FileInputFormat.addInputPath(job4, new Path(args[4]));
FileOutputFormat.setOutputPath(job4, new Path(args[5]));

job5.setJarByClass(CountryCount.class);
job5.setMapperClass(CountryCount.CountryCountMapper.class);
job5.setReducerClass(CountryCount.CountryCountReducer.class);
job5.setOutputKeyClass(Text.class);
```

```
job5.setOutputValueClass(IntWritable.class);
// 使用自定义的 InputFormat
job5.setInputFormatClass(WholeFileInputFormat.class);
FileInputFormat.addInputPaths(job5, args[1]);
FileOutputFormat.setOutputPath(job5, new Path(args[6]));

job6.setJarByClass(CountryProbability.class);
job6.setMapperClass(CountryProbability.CountryProbabilityMapper.class);
job6.setNumReduceTasks(0);
job6.setOutputKeyClass(Text.class);
job6.setOutputValueClass(IntWritable.class);
job6.setInputFormatClass(KeyValueTextInputFormat.class);
FileInputFormat.addInputPaths(job6, args[6]);
FileOutputFormat.setOutputPath(job6, new Path(args[7]));

job7.setJarByClass(CountryPredict.class);
job7.setMapperClass(CountryPredict.CountryPredictMapper.class);
job7.setReducerClass(CountryPredict.CountryPredictReducer.class);
job7.setMapOutputKeyClass(Text.class);
job7.setMapOutputValueClass(MapWritable.class);
job7.setOutputKeyClass(Text.class);
job7.setOutputValueClass(Text.class);
// 使用自定义的 InputFormat
job7.setInputFormatClass(FileNameInputFormat.class);
FileInputFormat.addInputPaths(job7, args[8]);
FileOutputFormat.setOutputPath(job7, new Path(args[9]));

ControlledJob controlledJob1 = new ControlledJob(conf);
ControlledJob controlledJob2 = new ControlledJob(conf);
ControlledJob controlledJob3 = new ControlledJob(conf);
ControlledJob controlledJob4 = new ControlledJob(conf);
ControlledJob controlledJob5 = new ControlledJob(conf);
ControlledJob controlledJob6 = new ControlledJob(conf);
ControlledJob controlledJob7 = new ControlledJob(conf);

controlledJob1.setJob(job1);
controlledJob2.setJob(job2);
controlledJob3.setJob(job3);
controlledJob4.setJob(job4);
controlledJob5.setJob(job5);
controlledJob6.setJob(job6);
controlledJob7.setJob(job7);

// Job4 依赖 Job1,Job2,Job3
```

```
            controlledJob4.addDependingJob(controlledJob1);
            controlledJob4.addDependingJob(controlledJob2);
            controlledJob4.addDependingJob(controlledJob3);
            // Job6 依赖 Job5
            controlledJob6.addDependingJob(controlledJob5);
            // Job7 依赖 Job4,Job6
            controlledJob7.addDependingJob(controlledJob4);
            controlledJob7.addDependingJob(controlledJob6);

            JobControl jobControl = new JobControl("NaiveBayes");
            jobControl.addJob(controlledJob1);
            jobControl.addJob(controlledJob2);
            jobControl.addJob(controlledJob3);
            jobControl.addJob(controlledJob4);
            jobControl.addJob(controlledJob5);
            jobControl.addJob(controlledJob6);
            jobControl.addJob(controlledJob7);

            Thread thread = new Thread(jobControl);
            thread.start();
            while(true){
                if(jobControl.allFinished()){
                    System.out.println(jobControl.getSuccessfulJobList());
                    jobControl.stop();
                    return;
                }
                if(jobControl.getFailedJobList().size() > 0){
                    System.out.println(jobControl.getFailedJobList());
                    jobControl.stop();
                    return;
                }
            }
        }
}
```

WordCountPerCountry.java

```
package io.github.trierbo.train;

import io.github.trierbo.NaiveBayes;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
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.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

/**
 * 统计每个类别下文档的单词总数，用于计算 P(A|C)
 * 在未进行拉普拉斯平滑时:
 * P(A|C) = 该类别下 A 出现的次数/该类别下文档的词数
 */
public class WordCountPerCountry {
    public static class WordCountPerCountryMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {
        protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
            InputSplit inputSplit = context.getInputSplit();
            // 通过文件路径获取类别
            String path = ((FileSplit) inputSplit).getPath().toString();
            String temp[] = path.split("/");
            String country = temp[temp.length - 1];
            IntWritable one = new IntWritable(1);
            context.write(new Text(country), one);
        }
    }

    public static class WordCountPerCountryReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {
        protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
            int sum = 0;
            for (IntWritable value: values) {
                sum += value.get();
            }
            // 将统计结果作为中间结果存放在内存中
            NaiveBayes.wordPerCountry.put(key.toString(), sum);
            context.write(key, new IntWritable(sum));
        }
    }
```

```
}
```

WordDict.java

```
package io.github.trierbo.train;

import io.github.trierbo.NaiveBayes;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
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.output.FileOutputFormat;

import java.io.IOException;

/**
 * 统计训练集中的所有不同的单词，作为词典，即文档中词所有可能的取值
 * 将其用于拉普拉斯平滑：
 * P(A|C) = (该类别下 A 出现的次数+1) / (该类别下文档的词数+词典大小)
 */
public class WordDict {

    public static class WordDictMapper extends Mapper<LongWritable, Text, Text, NullWritable>
{
        protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
            context.write(value, NullWritable.get());
        }
    }

    public static class WordDictReducer extends Reducer<Text, NullWritable, Text, NullWritable>
{
        protected void reduce(Text key, Iterable<NullWritable> values, Context context) throws
IOException, InterruptedException {
            // 每次调用 reduce 得到一个不同的词
            NaiveBayes.wordDict++;
            context.write(key, NullWritable.get());
        }
    }
```

```
}
```

WordCountryCount.java

```
package io.github.trierbo.train;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
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.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

/**
 * 统计每个类别每个单词出现的次数
 * 即用于计算 P(A|C)的分子
 */
public class WordCountryCount {

    public static class WordCountryCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
        protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
            InputSplit inputSplit = context.getInputSplit();
            String path = ((FileSplit) inputSplit).getPath().toString();
            String temp[] = path.split("/");
            String country = temp[temp.length - 1];
            IntWritable one = new IntWritable(1);
            // key 的格式为(单词_类别)
            context.write(new Text(value.toString() + '_' + country), one);
        }
    }

    public static class WordCountryCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
```

```
                int sum = 0;
                for (IntWritable value: values) {
                    sum += value.get();
                }
                context.write(key, new IntWritable(sum));
            }
        }

}
```

CondProbability.java

```
package io.github.trierbo.train;

import io.github.trierbo.NaiveBayes;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

/**
 * 用于计算条件概率，即 P(A|C)
 */
public class CondProbability {

    public static class CondProbabilityMapper extends Mapper<Text, Text, Text, DoubleWritable> {
        // 利用 KeyValueTextInputFormat 读取数据，其中 key 是 word_country, value 是出现的次数
        protected void map(Text key, Text value, Context context) throws IOException, InterruptedException {
            // 获取 country
            String[] splits = key.toString().split("_");
            String country = splits[splits.length - 1];

            // 获取出现次数
            int num = Integer.parseInt(value.toString());
            double prob = Math.log((double) (num + 1) / (NaiveBayes.wordPerCountry.get(country) + NaiveBayes.wordDict));
            NaiveBayes.condProb.put(key.toString(), prob);
            context.write(key, new DoubleWritable(prob));
        }
    }
}
```

CountryCount.java

```
package io.github.trierbo.train;

import io.github.trierbo.NaiveBayes;
import io.github.trierbo.utils.WholeFileInputFormat;
import org.apache.hadoop.conf.Configuration;
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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

/**
 * 统计每个类别下的文档总数
 * 用于计算类别的先验概率 P(C)
 */
public class CountryCount {

    public static class CountryCountMapper extends Mapper<Text, IntWritable, Text, IntWritable> {
        @Override
        protected void map(Text key, IntWritable value, Context context) throws IOException, InterruptedException {
            // 记录所有的文档总数
            NaiveBayes.newsNum++;
            context.write(key, value);
        }
    }

    public static class CountryCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        @Override
        protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable value: values) {
                sum += value.get();
            }
```

```
                context.write(key, new IntWritable(sum));
            }
        }

        public static void main(String[] args) throws Exception {
            if (args.length != 2) {
                System.err.println("Usage: WordCountPerCountry <input path> <output path>");
                System.exit(-1);
            }

            Configuration conf = new Configuration();
            conf.set("mapreduce.ifile.readahead", "false");
            Job job = Job.getInstance(conf);

            job.setJarByClass(CountryCount.class);
            job.setMapperClass(CountryCount.CountryCountMapper.class);
            job.setReducerClass(CountryCount.CountryCountReducer.class);
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(IntWritable.class);

            // 使用自定义的 InputFormat
            job.setInputFormatClass(WholeFileInputFormat.class);

            FileInputFormat.addInputPaths(job, args[0]);
            FileOutputFormat.setOutputPath(job, new Path(args[1]));

            System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
}
```

CountryProbability.java

```
package io.github.trierbo.train;

import io.github.trierbo.NaiveBayes;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

/**
 * 计算每个类别的概率
 */
public class CountryProbability {
```

```
    public static class CountryProbabilityMapper extends Mapper<Text, Text, Text, DoubleWritable> {
        // 利用 KeyValueTextInputFormat 读取数据，其中 key 是 country, value 是出现的次数
        protected void map(Text key, Text value, Context context) throws IOException, InterruptedException {
            int num = Integer.parseInt(value.toString());
            double prob = Math.log((double) num / NaiveBayes.newsNum);
            NaiveBayes.countryProb.put(key.toString(), prob);
            context.write(key, new DoubleWritable(prob));
        }
    }
}
```

CountryPredict.java

```
package io.github.trierbo.predict;

import io.github.trierbo.NaiveBayes;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.MapWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;
import java.util.HashMap;

/**
 * 预测测试集中每个文档的类别
 */
public class CountryPredict {
    public static class CountryPredictMapper extends Mapper<Text, Text, Text, MapWritable> {
        // 计算每个单词的条件概率, key 为文件路径，包含了文档的类别
        @Override
        protected void map(Text key, Text value, Context context) throws IOException, InterruptedException {
            // 记录当前 value 每个类别下的条件概率
            MapWritable wordCondPro = new MapWritable();
            for (HashMap.Entry<String, Double> entry : NaiveBayes.countryProb.entrySet()) {
                // word + country
                String wordCountry = value.toString() + '_' + entry.getKey();
                if (NaiveBayes.condProb.containsKey(wordCountry)) {
                    wordCondPro.put(new Text(entry.getKey()), new DoubleWritable(NaiveBayes.condProb.get(wordCountry)));
                } else {
```

```
                        // 说明训练集中该类别下的文档不包含该单词，防止整体预测概率为 0，使用拉普拉斯平滑
                        wordCondPro.put(new Text(entry.getKey()),
                                new DoubleWritable(Math.log(
                                        (double) 1 / ((NaiveBayes.wordPerCountry.get(entry.getKey())) + NaiveBayes.wordDict))));
                    }
                }
                context.write(key, wordCondPro);
            }
        }

        public static class CountryPredictReducer extends Reducer<Text, MapWritable, Text, Text> {
            @Override
            protected void reduce(Text key, Iterable<MapWritable> values, Context context) throws IOException, InterruptedException {
                // 记录该文档属于每个类别的概率
                HashMap<String, Double> countryProb = new HashMap<>();
                for (MapWritable value: values) {
                    for (String country: NaiveBayes.countryProb.keySet()) {
                        if (!countryProb.containsKey(country)) {
                            countryProb.put(country, ((DoubleWritable)value.get(new Text(country))).get());
                        } else {
                            countryProb.put(country, countryProb.get(country) + ((DoubleWritable)value.get(new Text(country))).get());
                        }
                    }
                }
                double max = Double.NEGATIVE_INFINITY;
                String country = null;
                // 查找最大概率所属的类别
                for (HashMap.Entry<String, Double> entry: NaiveBayes.countryProb.entrySet()) {
                    // 添加类别的先验概率
                    double prob = countryProb.get(entry.getKey()) + entry.getValue();
                    if (prob > max) {
                        max = prob;
                        country = entry.getKey();
                    }
                }
                context.write(key, new Text(country));
            }
        }
```

```
}
```

WholeFileInputFormat.java

```
package io.github.trierbo.utils;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.JobContext;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import java.io.IOException;

/**
 * key: Text 内容为类别名
 * value: IntWritable 用于计数
 */
public class WholeFileInputFormat extends FileInputFormat<Text, IntWritable> {

    // 文档不可分，使得针对每个文档产生一个类别
    @Override
    protected boolean isSplitable(JobContext context, Path filename) {
        return false;
    }

    @Override
    public RecordReader<Text, IntWritable> createRecordReader(InputSplit inputSplit,
TaskAttemptContext taskAttemptContext) throws IOException, InterruptedException {
        WholeFileRecordReader recordReader = new WholeFileRecordReader();
        recordReader.initialize(inputSplit, taskAttemptContext);
        return recordReader;
    }
}
```

WholeFileRecordReader.java

```
package io.github.trierbo.utils;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
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.FileSplit;

public class WholeFileRecordReader extends RecordReader<Text, IntWritable> {
    private FileSplit fileSplit;
    private Text key;
    private boolean processed = false;

    @Override
    public void initialize(InputSplit inputSplit, TaskAttemptContext taskAttemptContext) {
        this.fileSplit = (FileSplit) inputSplit;
    }

    // 每个文档是一个 split，只会产生一对 key/value
    @Override
    public boolean nextKeyValue() {
        if (!processed) {
            String[] splits = fileSplit.getPath().toString().split("/");
            key = new Text(splits[splits.length - 2]);
            processed = true;
            return true;
        }
        return false;
    }

    @Override
    public Text getCurrentKey() {
        return key;
    }

    @Override
    public IntWritable getCurrentValue() {
        return new IntWritable(1);
    }

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

    @Override
    public void close() {}
}
```

FileNameInputFormat.java

```
package io.github.trierbo.utils;

import org.apache.hadoop.io.Text;
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 java.io.IOException;

/**
 * key: Text 内容为文件路径
 * value: Text 内容为一行
 */
public class FileNameInputFormat extends FileInputFormat<Text, Text> {
    @Override
    public RecordReader<Text, Text> createRecordReader(InputSplit inputSplit,
TaskAttemptContext taskAttemptContext) throws IOException, InterruptedException {
        return new FileNameRecordReader();
    }
}
```

FileNameRecordReader.java

```
package io.github.trierbo.utils;

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.Text;
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.*;
import org.apache.hadoop.util.LineReader;

import java.io.IOException;

public class FileNameRecordReader extends RecordReader<Text, Text> {

    private LineReader lineReader;
    private Text key;
    private Text value = new Text();
```

```
    private long start;
    private long end;
    private long pos;

    @Override
    public void initialize(InputSplit inputSplit, TaskAttemptContext taskAttemptContext) throws
IOException {
        FileSplit split =(FileSplit) inputSplit;
        Configuration conf = taskAttemptContext.getConfiguration();
        Path path = split.getPath();
        this.key = new Text(path.toString());
        FileSystem fs = path.getFileSystem(conf);
        FSDataInputStream is = fs.open(path);
        lineReader = new LineReader(is,conf);

        // 处理起始点和终止点
        start =split.getStart();
        end = start + split.getLength();
        is.seek(start);
        if(start!=0){
            start += lineReader.readLine(new Text(),0,
                    (int)Math.min(Integer.MAX_VALUE, end-start));
        }
        pos = start;
    }

    @Override
    public boolean nextKeyValue() throws IOException {
        if (pos > end)
            return false;
        pos += lineReader.readLine(value);
        return value.getLength() != 0;
    }

    @Override
    public Text getCurrentKey() {
        return key;
    }

    @Override
    public Text getCurrentValue() {
        return value;
    }
```

```
    @Override
    public float getProgress() {
        if (start == end) {
            return 0.0f;
        } else {
            return Math.min(1.0f, (pos - start) / (float) (end - start));
        }
    }

    @Override
    public void close() throws IOException {
        lineReader.close();
    }

}
```

IndexCompute.java

```
package io.github.trierbo.analysis;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;

public class IndexCompute {
    // 选取 BRAZ 作为正例, CANA 作为反例
    public static void main(String[] args) throws Exception {
        /*
         * matrix[0] --- TP
         * matrix[1] --- FN
         * matrix[2] --- TN
         * matrix[3] --- FP
         */
        int[] matrix = new int[4];
        File predict = new File("dataset/predict.txt");
        try (BufferedReader reader = new BufferedReader(new FileReader(predict))) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] classification = line.split("\t");
                String[] filename = classification[0].split("/");
                String country = filename[filename.length - 2];
                String prediction = classification[1];
                if (country.equals("BRAZ")) {
                    if (country.equals(prediction))
                        matrix[0]++;
```

```
                        else
                            matrix[1]++;
                    } else {
                        if (country.equals(prediction))
                            matrix[2]++;
                        else
                            matrix[3]++;
                    }
                }
            }

            double precision = (double) matrix[0] / (matrix[0] + matrix[3]);
            double recall = (double) matrix[0] / (matrix[0] + matrix[1]);
            double F1 = (double) (2 * matrix[0]) / (2 * matrix[0] + matrix[3] + matrix[1]);

            System.out.println("----------------------------------------------------");
            System.out.println("Precision: " + precision);
            System.out.println("Recall: " + recall);
            System.out.println("F1: " + F1);
            System.out.println("----------------------------------------------------");
        }
}
```

五　数据集说明

在本次工程中选用了 400 个文档，包括 BRAZ 和 CANA 两个类别，其中每个类别分别有 200 个文档。将 400 个文档分为训练集和测试集，在工程中是按照 4:1 划分数据集的，故最后有 320 个文档作为训练集，80 个文档作为测试集。每个分类都是 160 个训练文档以及 40 个测试文档。不过由于每个类别选用的文档数完全相同，在计算先验概率时，会产生相同的概率，故会导致先验概率不起作用。

	BRAZ	CANA	总数
训练集	160	160	320
测试集	40	40	80
总数	200	200	400

六　程序运行说明

Job 名称	Map 数	Reduce 数
Job1	Map 数为 2 由于使用合并之后的训练数据，只有两个文档，并且文档大小小于 HDFS 的 block 的大小，故每篇文档对应一个 Map	Reduce 数为 1 使用的是默认 Reduce 任务数
Job2	Map 数为 2 理由与 Job1 相同	Reduce 数为 1 使用的是默认 Reduce 任务数

Job3	Map 数为 2 理由与 Job1 相同	Reduce 数为 1 使用的是默认 Reduce 任务数
Job4	Map 数为 1 由于 Job4 的输入是 Job3 的输出，并且 Job3 的 Reduce 数目是 1，所以只会产生一个 Partition 文件，并且在工程中该文件小于 block 的大小，所以只会有一个 Map	Reduce 数为 0 由于设置了 Reduce 为 0
Job5	Map 数为 320 由于使用未合并的小文档作为输入，文档个数为 320 个，故产生 320 个 Map 任务	Reduce 数为 1 使用的是默认 Reduce 任务数
Job6	Map 数为 1 由于 Job6 的输入是 Job5 的输出，并且 Job5 的 Reduce 数目是 1，所以只会产生一个 Partition 文件，并且在工程中该文件小于 block 的大小，所以只会有一个 Map	Reduce 数为 0 由于设置了 Reduce 为 0
Job7	Map 数为 80 由于输入是训练集，包含了 80 篇小文档，所以会产生 80 个 Map 任务	Reduce 数为 1 使用的是默认 Reduce 任务数

运行截图如下所示：

```
job id: NaiveBayes0
job state:      SUCCESS
job mapred id:  job_local258113530_0001
job message:    just initialized
job has no depending job:
, job name:     NaiveBayes2
job id: NaiveBayes1
job state:      SUCCESS
job mapred id:  job_local614479970_0002
job message:    just initialized
job has no depending job:
, job name:     NaiveBayes3
job id: NaiveBayes2
job state:      SUCCESS
job mapred id:  job_local186020840_0003
job message:    just initialized
job has no depending job:
, job name:     NaiveBayes4
job id: NaiveBayes3
job state:      SUCCESS
job mapred id:  job_local1827075322_0005
job message:    just initialized
job has 3 dependeng jobs:
        depending job 0:       NaiveBayes1
        depending job 1:       NaiveBayes2
        depending job 2:       NaiveBayes3
, job name:     NaiveBayes5
job id: NaiveBayes4
job state:      SUCCESS
job mapred id:  job_local1471425570_0004
job message:    just initialized
job has no depending job:
, job name:     NaiveBayes6
job id: NaiveBayes5
job state:      SUCCESS
job mapred id:  job_local245273504_0006
job message:    just initialized
job has 1 dependeng jobs:
        depending job 0:       NaiveBayes5
, job name:     NaiveBayes7
job id: NaiveBayes6
job state:      SUCCESS
job mapred id:  job_local832120660_0007
job message:    just initialized
job has 2 dependeng jobs:
        depending job 0:       NaiveBayes4
        depending job 1:       NaiveBayes6
]
falcon@falcon-vm:~/code/NaiveBayesClassifier/shell$
```

七：实验结果分析

选取 BRAZ 为正例，CANA 为反例，预测结果如下图所示：

```
hdfs://localhost:9000/user/falcon/country/test/BRAZ/487364newsML.txt    BRAZ
hdfs://localhost:9000/user/falcon/country/test/BRAZ/487371newsML.txt    BRAZ
hdfs://localhost:9000/user/falcon/country/test/BRAZ/487373newsML.txt    BRAZ
hdfs://localhost:9000/user/falcon/country/test/BRAZ/487376newsML.txt    BRAZ
hdfs://localhost:9000/user/falcon/country/test/BRAZ/487380newsML.txt    BRAZ
hdfs://localhost:9000/user/falcon/country/test/BRAZ/487382newsML.txt    BRAZ
hdfs://localhost:9000/user/falcon/country/test/BRAZ/487384newsML.txt    BRAZ
hdfs://localhost:9000/user/falcon/country/test/BRAZ/487387newsML.txt    BRAZ
hdfs://localhost:9000/user/falcon/country/test/BRAZ/487388newsML.txt    BRAZ
hdfs://localhost:9000/user/falcon/country/test/BRAZ/487392newsML.txt    BRAZ
hdfs://localhost:9000/user/falcon/country/test/BRAZ/487394newsML.txt    BRAZ
hdfs://localhost:9000/user/falcon/country/test/BRAZ/487400newsML.txt    BRAZ
hdfs://localhost:9000/user/falcon/country/test/BRAZ/487411newsML.txt    BRAZ
hdfs://localhost:9000/user/falcon/country/test/BRAZ/487415newsML.txt    BRAZ
```

```
/usr/lib/jvm/java-8-openjdk-amd64/bin/java ...
--------------------------------------------------------
TP:40
FN:0
TN:39
FP:1
Precision: 0.975609756097561
Recall: 1.0
F1: 0.9876543209876543
--------------------------------------------------------

Process finished with exit code 0
```

其混淆矩阵如下所示：

实际值\预测值	BRAZ	CANA
BRAZ	40	0
CANA	1	39

准确率，召回率，F1 分别如下：

Precision	Recall	F1
0.976	1.000	0.988