

---

## 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

对于问题：给定一个类标签集合  $C = \{c_1, c_2, \dots, c_n\}$  以及一个文档  $d$ ，给文档  $d$  分配一个最合适的类别标签。

解决方法就是计算条件概率： $P(c_i | d)$ ，( $i = 1, 2, \dots, n$ )，选出其中最大的一个条件概率  $P(c_j | d) = \max(P(c_i | d) (i = 1, 2, \dots, n))$ ，那么就认为文档  $d$  属于  $c_j$  类。

对于该概率的计算，有 Bayes 公式：

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

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

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

对于先验概率有：

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

根据词项独立性假设，对于似然概率有：

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

因此，估计  $p(d | c_i)$  就需要估计  $p(t_k | c_i)$ ：

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

为了避免出现 0 次的词项对分类决策的影响，采用 +1 平滑：

$$p(t_k | c_i) = \frac{t_k \text{ 在类型为 } c_i \text{ 的文档中出现的次数} + 1}{\text{在类型为 } c_i \text{ 的文档中出现的 term 的总数} + B}$$

其中  $B$  为词典的大小。

### 三、贝叶斯分类器训练的 MapReduce 算法设计

#### 1. Job1:

第一个 job 是统计每个类中文件的数量总数，于是不希望文件被切分，而是用一个 mapper 完整地处理每一个输入文件，我采用的是使用 `FileInputFormat` 的具体子类（如 `TextInputFormat`），并且重新实现 `isSplittable()` 方法把返回值设为 `false`。在输入路径 `hdfs://master/input` 下有 4 个以类名命名的文件夹，每个类的文件夹下又有一定数量的文档。Job1 执行时会找到 `/input` 路径下的每个文档，并将每一个文档作为一个整体的输入交给 `map` 进行任务处理，`map` 获取每个文件的类名，并将 `map` 的输出设置为 `<类名, 1>`。`Map` 的输出丢到 `map` 的上下文中，自动地将 `key` 值相同的 `value` 合并在一个集合中，作为 `reduce` 的输入。`Reduce` 的输出写到 `hdfs://master/out1` 下的 `part-r-000000` 文档中，格式为每行的内容：“类名 文档名 1”。Job1 的 dataflow 示意图如下所示：

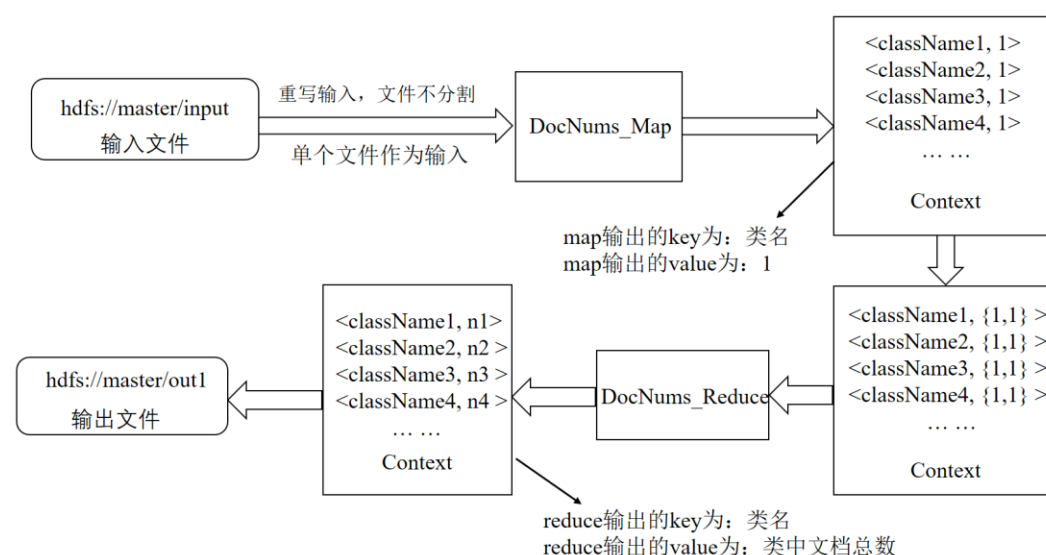


图 3.1: Job1 Data Flow 示意图

#### 2. Job2:

第二个 job 是统计每个类中各个单词的数量总数，输入是训练集中所有的数据，在输入路径 `hdfs://master/input` 下有 4 个以类名命名的文件夹，每个类的文件夹下又有一定数量的文档。Job1 执行时会找到 `/input` 路径下的每个文档，并读取每个文件中的内容，将每个文件中行偏移量为 `key`，行的数据为 `value` 作为输入交给 `map` 进行任务处理，`map` 获取每个文件的类名，并将 `value` 进行分词，`map` 的输出设置为 `<<类名,单词名>, 1>`。`Map` 的输出丢到 `map` 的上下文中，自动地

将 key 值相同的 value 合并在一个集合中，作为 reduce 的输入。Reduce 则将相同 key 值的 value 进行累加，Reduce 的输出写到 `hdfs://master/out2` 下的 `part-r-00000` 文档中，格式为每行的内容：<<类名,单词名>，该单词总数>。Job2 的 dataflow 示意图如下所示：

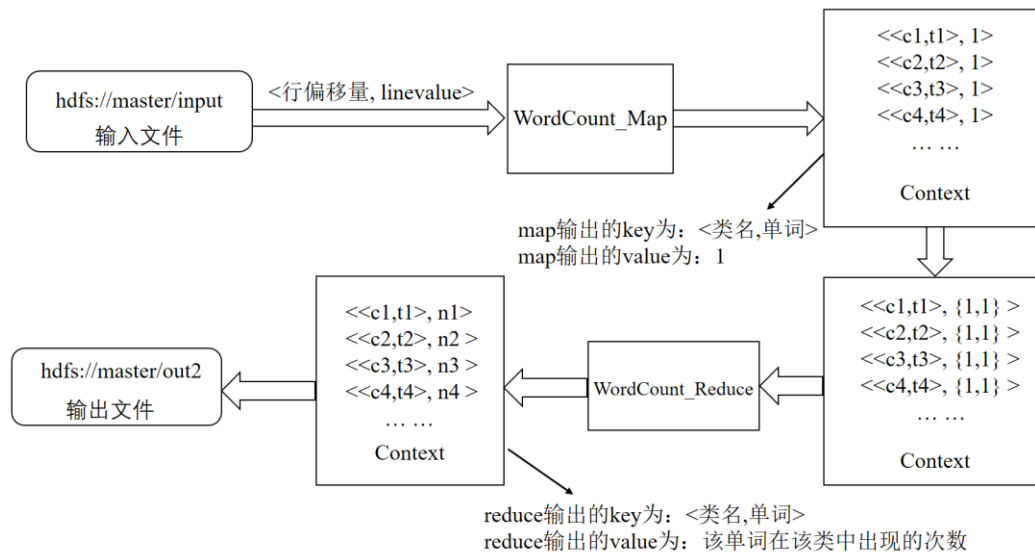


图 3.2: Job2 Data Flow 示意图

### 3. Job3:

第三个 job 是为第四个 job 进行前处理，对测试集进行处理，将测试集中的文档输出到一个文档进行处理，输出为<<类名,文档名>,单词 1,单词 2,单词 3.....>。输入是测试集中所有的数据，在输入路径 `hdfs://master/test` 下有 4 个以类名命名的文件夹，每个类的文件夹下又有一定数量的文档。Job3 执行时会找到 `/test` 路径下的每个文档，并读取每个文件中的内容，将每个文件中行偏移量为 key，行的数据为 value 作为输入交给 map 进行任务处理，map 获取每个文件的类名，并将 value 进行分词，map 的输出设置为<<类名,文档名>, 单词>。Map 的输出丢到 map 的上下文中，自动地将 key 值相同的 value 合并在一个集合中，作为 reduce 的输入。Reduce 则将相同 key 值的 value 进行连接，Reduce 的输出写到 `hdfs://master/out3` 下的 `part-r-00000` 文档中，格式为每行的内容：<<类名,文档名>, 单词 1,单词 2,单词 3.....>。Job3 的 dataflow 示意图如下所示：

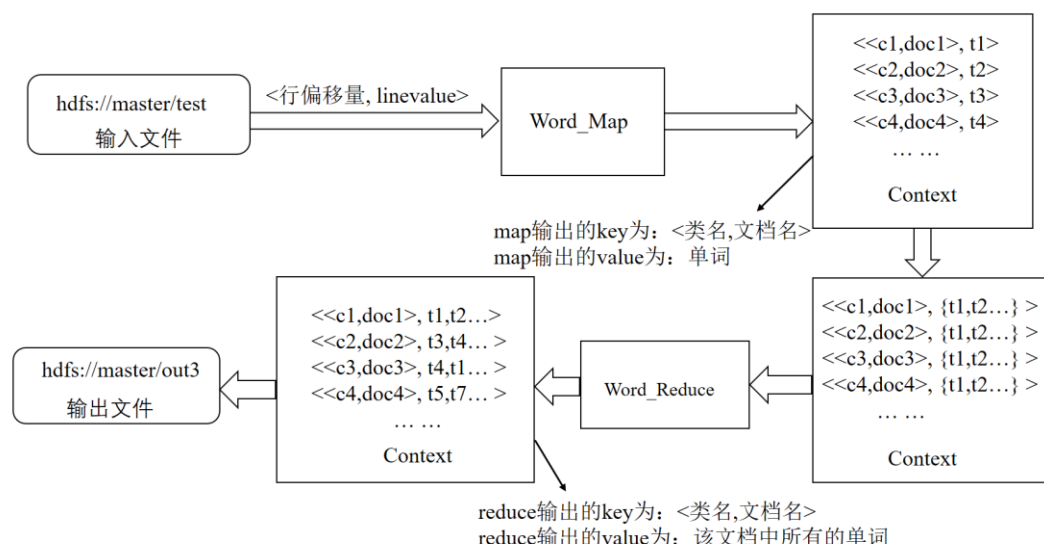


图 3.3: Job3 Data Flow 示意图

#### 4. Job4:

第四个 job 是对测试集中的文档进行预测，输入为 job3 的输出文件 `hdfs://master/out3` 下有 1 个文档 `part-r-00000`，输出为<文档名，类名>，将对文档的类别进行预测。在执行 Job4 之前会先将通过读取 job2 的输出 `out2` 将训练集中的先验概率和条件概率计算出来并保存到 `HashMap` 中，Job4 执行时会找到 `out3` 下的文件，并读取每个文件中的内容，将每个文件中行偏移量为 key，行的数据为 value 作为输入交给 map 进行任务处理。Map 将 value 中的单词进行分词，并分别获取每个单词的在每个类下的条件概率，计算出其属于每个类的概率，所以 map 的输出为<文档名,<类名，概率>>。Map 的输出丢到 map 的上下文中，自动地将 key 值相同的 value 合并在一个集合中，作为 reduce 的输入。Reduce 则将相同 key 值的中的概率求最大值，取最大的概率为真正的类别，Reduce 的输出写到 `hdfs://master/out4` 下的 `part-r-00000` 文档中，格式为每行的内容：<文档名，类名>。Job4 的 dataflow 示意图如下所示：

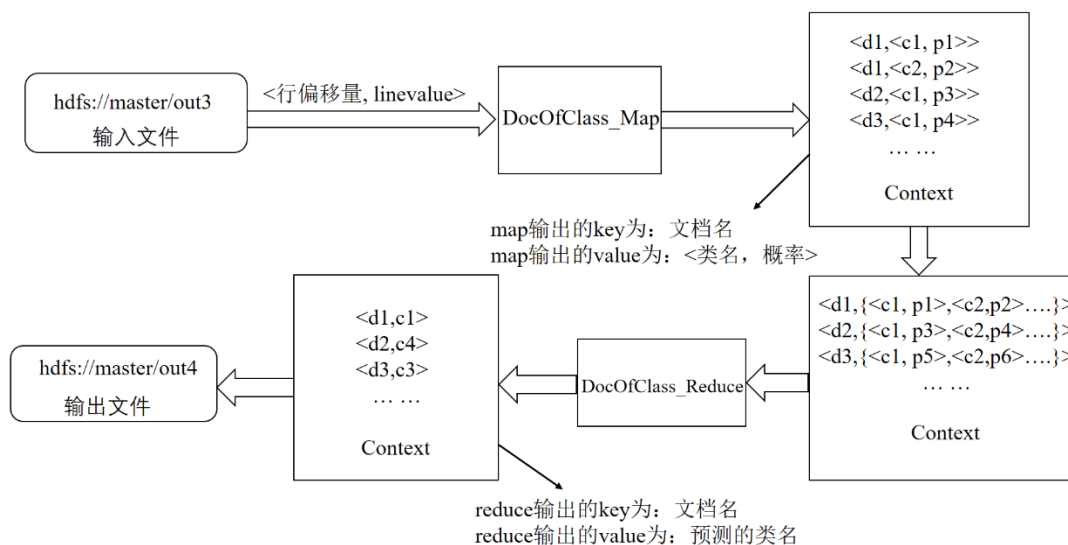


图 3.4: Job4 Data Flow 示意图

## 5. Job5:

第五个和第六个 job 是对预测进行评估，输入为 job3 的输出文件 `hdfs://master/out3` 下有 1 个文档 `part-r-00000`，输出为 `<类名, 类中所有的文档名>`。Job5 执行时会找到 `out3` 下的文件，并读取文件中的内容，将每个文件中行偏移量为 key，行的数据为 value 作为输入交给 map 进行任务处理。每行的第一个单词为类名，第二个单词为文档名，分别提取第一个和第二个单词，将其作为 map 的输出，所以 map 的输出为 `<类名, 文档名>`，Map 的输出丢到 map 的上下文中，自动地将 key 值相同的 value 合并在一个集合中，作为 reduce 的输入。Reduce 则将相同 key 值的 value 进行连接，Reduce 的输出写到 `hdfs://master/out5` 下的 `part-r-00000` 文档中，格式为每行的内容：`<类名, 文档名 1, 文档名 2.....>`。Job5 的 dataflow 示意图如下所示：

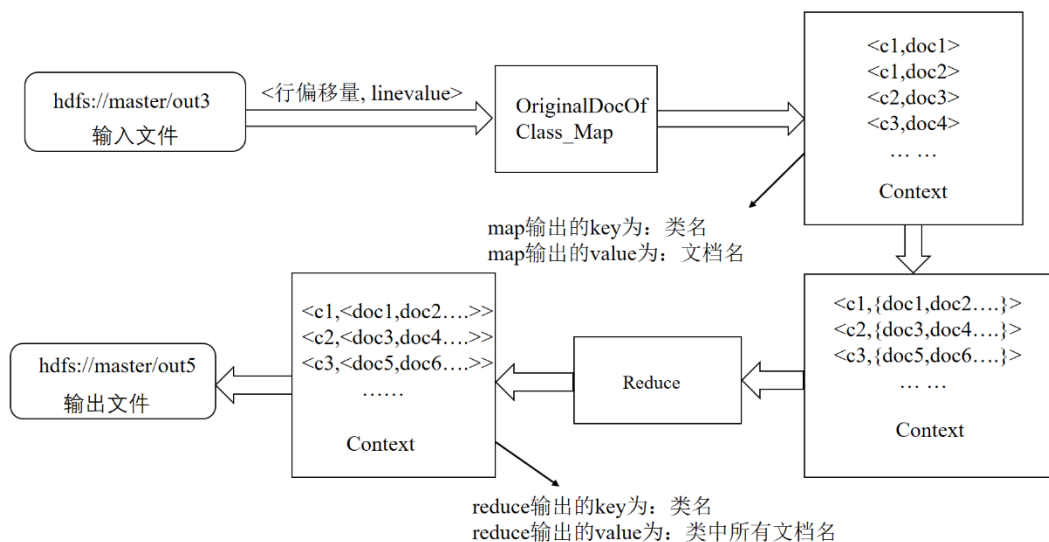


图 3.5: Job5 Data Flow 示意图

## 6. Job6:

第六个 job 是对预测进行评估，输入为 job4 的输出文件 hdfs://master/out4 下有 1 个文档 part-r-00000，输出为<类名，类中所有的文档名>。Job6 执行时会找到 out4 下的文件，并读取文件中的内容，将每个文件中行偏移量为 key，行的数据为 value 作为输入交给 map 进行任务处理。每行的第一个单词为类名，第二个单词为文档名，分别提取第一个和第二个单词，将其位置交换后作为 map 的输出，所以 map 的输出为<类名，文档名>，Map 的输出丢到 map 的上下文中，自动地将 key 值相同的 value 合并在一个集合中，作为 reduce 的输入。Reduce 则将相同 key 值的 value 进行连接，Reduce 的输出写到 hdfs://master/out6 下的 part-r-00000 文档中，格式为每行的内容：<类名,文档名 1，文档名 2.....>。Job6 的 dataflow 示意图如下所示：

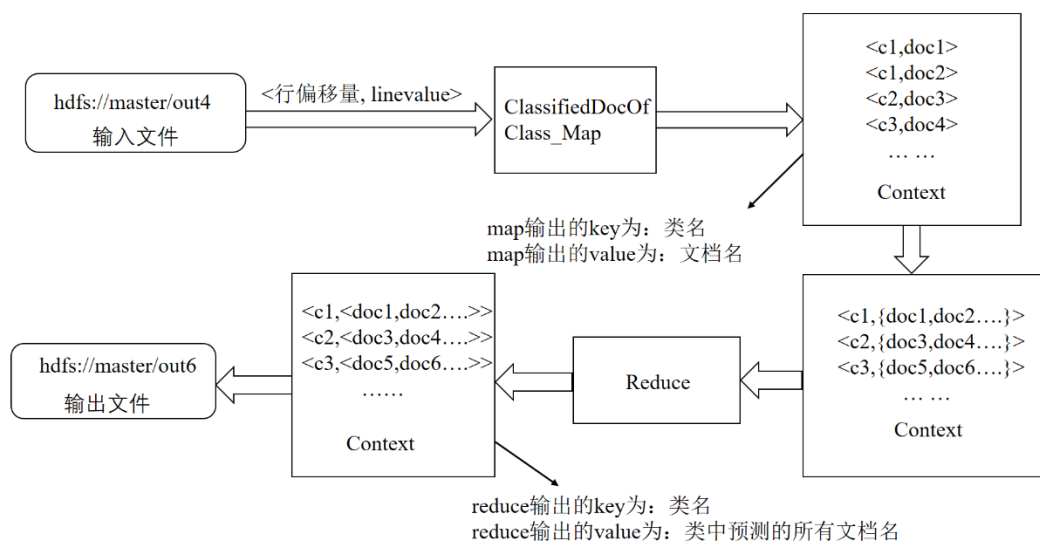


图 3.6: Job6 Data Flow 示意图

## 四、源代码清单

总共有三个 java 文件，分别为 BayesClassify.java、Prediction.java、Evaluation.java，其中 BayesClassify.java 中完成了 job1 和 job2，对训练集中的数据进行统计；Prediction.java 中完成了 job3 和 job4，完成了条件概率和先验概率的计算并对测试集中文档的类别进行预测；Evaluation.java 中完成了 job5 和 job6，对预测的结果进行一个综合评估，分别计算出了 Precision，Recall 和 F1 的值。

### 1. BayesClassify.java 文件

```
package com.loring.bayes;
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.*;
import org.apache.hadoop.mapreduce.*;
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;
import java.util.StringTokenizer;

/**
 * @author XYL
 * @date 2018.12.28
 * 贝叶斯分类器
 */
```

---

```
public class BayesClassify {
    public static void main(String[] args) throws IOException,
        ClassNotFoundException, InterruptedException {
        Configuration conf = new Configuration();
        FileSystem hdfs = FileSystem.get(conf);

        Path outputPath1 = new Path(args[1]);
        if(hdfs.exists(outputPath1))
            hdfs.delete(outputPath1, true);
        Job job1 = Job.getInstance(conf, "DocNumbers");
        job1.setJarByClass(BayesClassify.class);
        //设置输入输出格式
        job1.setInputFormatClass(WholeFileInputFormat.class);
        job1.setMapperClass(DocNums_Map.class);
        job1.setCombinerClass(DocNums_Reduce.class);
        job1.setReducerClass(DocNums_Reduce.class);
        FileInputFormat.setInputDirRecursive(job1,true);
        job1.setOutputKeyClass(Text.class); //reduce 阶段的输出的 key
        job1.setOutputValueClass(IntWritable.class); //reduce 阶段的输出
        出的 value
        FileInputFormat.addInputPath(job1, new Path(args[0]));
        FileOutputFormat.setOutputPath(job1, new Path(args[1]));
        boolean isSuccess = job1.waitForCompletion(true);
        if(!isSuccess) {
            System.exit(1);
        }

        Path outputPath2 = new Path(args[2]);
        if(hdfs.exists(outputPath2))
            hdfs.delete(outputPath2, true);
        Job job2 = Job.getInstance(conf, "WordCount");
        job2.setJarByClass(BayesClassify.class);
        job2.setMapperClass(WordCount_Map.class);
        job2.setCombinerClass(WordCount_Reduce.class);
        job2.setReducerClass(WordCount_Reduce.class);
        FileInputFormat.setInputDirRecursive(job2,true);
        job2.setOutputKeyClass(Text.class); //reduce 阶段的输出的 key
        job2.setOutputValueClass(IntWritable.class); //reduce 阶段的输出
        出的 value
        FileInputFormat.addInputPath(job2, new Path(args[0]));
        FileOutputFormat.setOutputPath(job2, new Path(args[2]));
        System.exit(job2.waitForCompletion(true) ? 0 : 1);
    }

    /*
```



---

```

    * 第一个 MapReduce 用于统计每个类对应的文件数量
    * 为计算先验概率准备:
    * 输入:args[0],训练集
    * 输出:args[1],key 为类名,value 为类对应的文档数目,即
    <ClassName,DocNums>
    */
    public static class DocNums_Map extends Mapper<NullWritable,
BytesWritable, Text, IntWritable> {
        private Text newKey = new Text();
        private final static IntWritable one = new IntWritable(1);
        public void map(NullWritable key, BytesWritable value, Context
context) throws IOException, InterruptedException{
            //得到当前所处理分片
            InputSplit inputsplit = context.getInputSplit();
            //将当前所处理分片的路径名按照目录结构解析为: 类名、文档名
            String                                className                                =
((FileSplit)inputsplit).getPath().getParent().getName();
            //将当前所处理分片所属的类名和文档名中间加上制表符组合成一个字符串
            //String classAndDoc = className;
            newKey.set(className);
            context.write(newKey, one);
        }
    }

    public static class DocNums_Reduce extends Reducer<Text,
IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();
        public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException{
            int sum = 0;
            for(IntWritable value:values){
                sum += value.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    public static class WholeFileInputFormat extends
FileInputFormat<NullWritable, BytesWritable>{

        @Override
        protected boolean isSplittable(JobContext context, Path
filename) {
            return false;
        }
    }

```

---

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

public static class WholeFileRecordReader extends
RecordReader<NullWritable, BytesWritable> {
    private FileSplit fileSplit;           //保存输入的分片，它将被转
换成一条 (key, value) 记录
    private Configuration conf;           //配置对象
    private BytesWritable value = new BytesWritable(); //value
对象，内容为空
    private boolean processed = false;    //布尔变量记录记录是否被处
理过

    @Override
    public void initialize(InputSplit split, TaskAttemptContext
context)
        throws IOException, InterruptedException {
        this.fileSplit = (FileSplit) split;           //将输入分片强制
转换成 FileSplit
        this.conf = context.getConfiguration(); //从 context 获取
配置信息
    }

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

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

    @Override
    public boolean nextKeyValue() throws IOException,
InterruptedException {
        if (!processed) { //如果记录没有被处理过
            //从 fileSplit 对象获取 split 的字节数，创建 byte 数组 contents
```

---

```

        byte[] contents = new byte[(int) fileSplit.getLength()];
        Path file = fileSplit.getPath(); //从 fileSplit 对象获取
输入文件路径
        FileSystem fs = file.getFileSystem(conf); //获取文件系
统对象
        FSDataInputStream in = null; //定义文件输入流对象
        try {
            in = fs.open(file); //打开文件, 返回文件输入流对象
            IOUtils.readFully(in, contents, 0,
contents.length); //从输入流读取所有字节到 contents
            value.set(contents, 0, contents.length); // 将
contents 内容设置到 value 对象中
        } finally {
            IOUtils.closeStream(in); //关闭输入流
        }
        processed = true; //将是否处理标志设为 true, 下次调用该方
法会返回 false
        return true;
    }
    return false; //如果记录处理过, 返回 false, 表示 split 处理完
毕
}

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

@Override
public void close() throws IOException {
    // do nothing
}

}

/*
 * 第二个 MapReduce 用于统计每个类下单词的数量
 * 输入:args[0],训练集,输入为<行偏移量, 单词>
 * 输出:args[2],输出为<类名_单词名, 数量>
 */
public static class WordCount_Map extends Mapper<LongWritable,
Text, Text, IntWritable>{
    private Text nameAndWord = new Text();
// KEYOUT

```

---

```

        private final static IntWritable one = new IntWritable(1); //
VALUEOUT
        @Override
        public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
            InputSplit inputsplit = context.getInputSplit();
            String                className                =
((FileSplit)inputsplit).getPath().getParent().getName();
            String cAndTValue;
            String lineValue = value.toString();
            // 分词：将每行的单词进行分割,按照" \t\n\r\f"(空格、制表符、换行
符、回车符、换页)进行分割
            StringTokenizer tokenizer = new StringTokenizer(lineValue);
            // 遍历
            while (tokenizer.hasMoreTokens()) {
                //获取每个单词
                String wordValue = tokenizer.nextToken();
                // 设置 map 输出的 key 值为类名和单词中间加上制表符组合成的字符
串

                cAndTValue = className + '\t' + wordValue;
                //将类名单词字符串的值赋给 hadoop 的 Text 对象
                nameAndWord.set(cAndTValue);
                //将<类名单词, 1>键值对写入上下文
                context.write(nameAndWord, one);
            }
        }
    }

    //将相同的类名_单词累加
    public static class WordCount_Reduce extends Reducer<Text,
IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();
        @Override
        public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }
}

```

---

## 2. Prediction.java 文件

```
package com.loring.bayes;

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.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.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URI;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.StringTokenizer;

public class Prediction {
    public static void main(String[] args) throws IOException,
        ClassNotFoundException, InterruptedException {
        Configuration conf = new Configuration();
        FileSystem hdfs = FileSystem.get(conf);

        Path outputPath1 = new Path(args[1]);
        if(hdfs.exists(outputPath1))
            hdfs.delete(outputPath1, true);
        Job job1 = Job.getInstance(conf, "Word");
        job1.setJarByClass(Prediction.class);
        job1.setMapperClass(Prediction.WordMapper.class);
        job1.setCombinerClass(Prediction.WordReducer.class);
        job1.setReducerClass(Prediction.WordReducer.class);
        FileInputFormat.setInputDirRecursive(job1,true);
        job1.setOutputKeyClass(Text.class);//reduce 阶段的输出的 key
        job1.setOutputValueClass(Text.class);//reduce 阶段的输出的
value
        FileInputFormat.addInputPath(job1, new Path(args[0]));
        FileOutputFormat.setOutputPath(job1, new Path(args[1]));
        boolean isSuccess = job1.waitForCompletion(true);
        if(!isSuccess) {
```

---

```

        System.exit(1);
    }

    Path outputPath2 = new Path(args[2]);
    if(hdfs.exists(outputPath2))
        hdfs.delete(outputPath2, true);
    Job job2 =Job.getInstance(conf, "Prediction");
    job2.setJarByClass(Prediction.class);
    job2.setMapperClass(Prediction.DocOfClassMap.class);
    job2.setCombinerClass(Prediction.DocOfClassReduce.class);
    job2.setReducerClass(Prediction.DocOfClassReduce.class);
    FileInputFormat.setInputDirRecursive(job2,true);
    job2.setOutputKeyClass(Text.class);//reduce 阶段的输出的 key
    job2.setOutputValueClass(Text.class);//reduce 阶段的输出的
value
    FileInputFormat.addInputPath(job2, new Path(args[1]));
    FileOutputFormat.setOutputPath(job2, new Path(args[2]));
    System.exit(job2.waitForCompletion(true) ? 0 : 1);
}

```

```

    public static class WordMapper extends Mapper<LongWritable, Text,
Text, Text>{
        private Text newKey = new Text();
        private Text newValue = new Text();
        public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
            InputSplit inputsplit = context.getInputSplit();
            // 类名
            String                className                =
((FileSplit)inputsplit).getPath().getParent().getName();
            // 文档名
            String                docName                  =
((FileSplit)inputsplit).getPath().getName();
            StringTokenizer        itr                      =
new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                newKey.set(className+"\t"+docName);
                newValue.set(itr.nextToken());
                context.write(newKey,newValue);
            }
        }
    }
}

```

```

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

```

---

```

        private Text result = new Text();
        private StringBuffer stringBuffer;
        public void reduce(Text key, Iterable<Text> values, Context
context) throws IOException, InterruptedException {
            stringBuffer=new StringBuffer();
            for(Text word:values){
                stringBuffer = stringBuffer.append(word.toString()+"
");
            }
            result.set(stringBuffer.toString());
            System.out.println("key==>" +key);
            System.out.println("value==>" +result.toString());
            context.write(key, result);
        }
    }

    /*
    * 第三个 MapReduce 进行贝叶斯测试
    * 输入:args[3], 处理后的测试数据, 测试数据格式<<class doc>,word1
word2 ...>
    *      HashMap<String,Double> classProbably 先验概率
    *      HashMap<String,Double> wordsProbably 条件概率
    * 输出:args[4], 输出每一份文档经贝叶斯分类后所对应的类, 格式为
<doc,class>
    */
    public static class DocOfClassMap extends Mapper<LongWritable,
Text, Text, Text> {
        public void setup(Context context) throws IOException{
            GetPriorProbably(); //先验概率
            GetConditionProbably(); //条件概率
        }
        private Text newKey = new Text();
        private Text newValue = new Text();
        public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException{
            // 分词:将每行的单词进行分割,按照"\t\n\r\f" (空格、制表符、换行符、
回车符、换页)进行分割
            String[] result = value.toString().split("\\s");
            String docName = result[1]; // 第二个, 第一个是类名
            for(Map.Entry<String, Double>
entry:classProbably.entrySet()) { //外层循环遍历所有类别
                String myKey = entry.getKey();//类名
                newKey.set(docName);//新的键值的 key 为<文档名>
                double tempValue = Math.log(entry.getValue());//构建临

```

---

时键值对的 value 为各概率相乘, 转化为各概率取对数再相加

```
        for(int i=2; i<result.length; i++){
            String tempKey = myKey + "\t" + result[i]; //构建临时键值对<class_word>, 在 wordsProbably 表中查找对应的概率
            if(wordsProbably.containsKey(tempKey)){
                //如果测试文档的单词在训练集中出现过, 则直接加上之前计算的概率
                tempValue += Math.log(wordsProbably.get(tempKey));
            }
            else{//如果测试文档中出现了新单词则加上之前计算新单词概率
                tempValue += Math.log(wordsProbably.get(myKey));
            }
            newValue.set(myKey + "\t" + tempValue); //新的键值的 value 为<类名 概率>
            context.write(newKey, newValue); //一份文档遍历在一个类中遍历完毕, 则将结果写入文件, 即<docName, <class probably>>
            System.out.println(newKey + "\t" + newValue);
        }
    }

    public static class DocOfClassReduce extends Reducer<Text, Text, Text, Text> {
        Text newValue = new Text();
        public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException{
            boolean flag = false; //标记, 若第一次循环则先赋值, 否则比较若概率更大则更新
            String tempClass = null;
            double tempProbably = 0.0;
            for(Text value: values){
                System.out.println("value....."+value.toString());
                String[] result = value.toString().split("\\s");
                String className=result[0];
                String probably=result[1];

                if(flag != true){ //循环第一次
                    tempClass = className;
                    tempProbably = Double.parseDouble(probably);
                    flag = true;
                }
            }
        }
    }
}
```



---

```

        }else{//否则当概率更大时就更新 tempClass 和 tempProbably
            if(Double.parseDouble(probably) > tempProbably){
                tempClass = className;
                tempProbably = Double.parseDouble(probably);
            }
        }
    }

    newValue.set(tempClass + "\t" +tempProbably);
    //newValue.set(tempClass+"."+values.iterator().next());
    context.write(key, newValue);
    System.out.println(key + "\t" + newValue);
}
}

/*计算先验概率:
 * 该静态函数计算每个类的文档在总类中占的比例,即先验概率  $P(c)$  =类 c 下文件总数/整个训练样本的文件总数
 * 输入:对应第一个 MapReduce 的输出 args[1]
 * 输出:得到 HashMap<String,Double>存放的是<类名,概率>
 */
private static HashMap<String, Double> classProbably = new
HashMap<String, Double>();//<类别, 概率>, 即<class,priorProbably>

public static HashMap<String, Double> GetPriorProbably() throws
IOException {
    Configuration conf = new Configuration();
    String filePath = "/output1/part-r-00000";
    FSDataInputStream fsr = null;
    BufferedReader bufferedReader = null;
    String lineValue = null;
    double sum = 0; //文档总数量

    try {
        FileSystem fs = FileSystem.get(URI.create(filePath),
conf);
        fsr = fs.open(new Path(filePath));
        bufferedReader = new BufferedReader(new
InputStreamReader(fsr));
        while ((lineValue = bufferedReader.readLine()) != null)
        { //按行读取
            // 分词: 将每行的单词进行分割,按照" \t\n\r\f"(空格、制表符、
            换行符、回车符、换页)进行分割
            StringTokenizer tokenizer = new
StringTokenizer(lineValue);

```

---

```

        String className = tokenizer.nextToken(); //类名
        String num_C_Tmp = tokenizer.nextToken(); //文档数量
        double numC = Double.parseDouble(num_C_Tmp);
        classProbably.put(className, numC);
        sum = sum + numC; //文档总数量
    }
} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (bufferedReader != null) {
        try {
            bufferedReader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

Iterator iterator = classProbably.entrySet().iterator();
while (iterator.hasNext()) {
    Map.Entry entry = (Map.Entry) iterator.next();
    Object key = entry.getKey();
    double val =
Double.parseDouble(entry.getValue().toString()) / sum;
    classProbably.put(key.toString(), val);
    System.out.println(classProbably.get(key));
}
return classProbably;
}

/* 计算条件概率
 * 条件概率  $P(t_k|c)$ =(类 c 下单词  $t_k$  在各个文档中出现过的次数之和+1)/(类 c
下单词总数+不重复的单词总数)
 * 输入:对应第二个 MapReduce 的输出<<class,word>,counts>
 * 输出:得到 HashMap<String,Double>,即<<类名:单词>,概率>
 */
private static HashMap<String, Double> wordsProbably = new
HashMap<String, Double>();
public static HashMap<String, Double> GetConditionProbably()
throws IOException {
    String filePath = "/output2/part-r-00000";
    Configuration conf = new Configuration();
    FSDataInputStream fsr = null;
    BufferedReader bufferedReader = null;

```

---

```

        String lineValue = null;
        HashMap<String,Double> wordSum=new HashMap<String, Double>();
//存放的为<类名, 单词总数>

        try {
            FileSystem fs = FileSystem.get(URI.create(filePath),
            conf);

            fsr = fs.open(new Path(filePath));
            bufferedReader = new BufferedReader(new
            InputStreamReader(fsr));
            while ((lineValue = bufferedReader.readLine()) != null)
            { //按行读取

                // 分词：将每行的单词进行分割,按照" \t\n\r\f"(空格、制表符、
                换行符、回车符、换页)进行分割
                StringTokenizer tokenizer = new
                StringTokenizer(lineValue);
                String className = tokenizer.nextToken();
                String word =tokenizer.nextToken();
                String numWordTmp = tokenizer.nextToken();
                double numWord = Double.parseDouble(numWordTmp);
                if(wordSum.containsKey(className))

wordSum.put(className,wordSum.get(className)+numWord+1.0); //加 1.0
是因为每一次都是一个不重复的单词
                else
                    wordSum.put(className,numWord+1.0);
            }
            fsr.close();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (bufferedReader != null) {
                try {
                    bufferedReader.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

// 现在来计算条件概率
        try {
            FileSystem fs = FileSystem.get(URI.create(filePath),
            conf);

            fsr = fs.open(new Path(filePath));
            bufferedReader = new BufferedReader(new

```

---

```

InputStreamReader(fsr));
        while ((lineValue = bufferedReader.readLine()) != null)
    { //按行读取
        // 分词：将每行的单词进行分割,按照" \t\n\r\f" (空格、制表符、
        换行符、回车符、换页)进行分割
        StringTokenizer tokenizer = new
StringTokenizer(lineValue);
        String className = tokenizer.nextToken();
        String word =tokenizer.nextToken();
        String numWordTmp = tokenizer.nextToken();
        double numWord = Double.parseDouble(numWordTmp);
        String key=className+"\t"+word;

wordsProbably.put(key, (numWord+1.0)/wordSum.get(className));

//System.out.println(className+"\t"+word+"\t"+wordsProbably.get(k
ey));
    }
    fsr.close();
} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (bufferedReader != null) {
        try {
            bufferedReader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

// 对测试集中出现的新单词定义概率
Iterator iterator = wordSum.entrySet().iterator(); // 获取
key 和 value 的 set
    while (iterator.hasNext()) {
        Map.Entry entry = (Map.Entry) iterator.next(); // 把
hashmap 转成 Iterator 再迭代到 entry
        Object key = entry.getKey(); //从 entry 获取 key

wordsProbably.put(key.toString(),1.0/Double.parseDouble(entry.get
Value().toString()));
    }

//      Iterator iter = wordsProbably.entrySet().iterator(); //
获取 key 和 value 的 set

```

---

```
//      while (iter.hasNext()) {
//          Map.Entry entry = (Map.Entry) iter.next();    //      把
//          hashmap 转成 Iterator 再迭代到 entry
//          Object key = entry.getKey();                //从 entry 获取 key
//          Object val =entry.getValue();
//          System.out.println(key.toString()+"\t"+
//          val.toString());
//      }

      return wordsProbably;
  }
}
```

### 3. Evaluation.java 文件

```
package com.loring.bayes;
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.LongWritable;
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.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URI;
import java.util.ArrayList;

public class Evaluation {
    public static void main(String[] args) throws IOException,
    ClassNotFoundException, InterruptedException {
        Configuration conf = new Configuration();
        FileSystem hdfs = FileSystem.get(conf);

        Path outputPath1 = new Path(args[1]);
        if(hdfs.exists(outputPath1))
            hdfs.delete(outputPath1, true);
        Job job1 =Job.getInstance(conf, "OriginalDocOfClass");
        job1.setJarByClass(Evaluation.class);
        job1.setMapperClass(Evaluation.OriginalDocOfClassMap.class);
        job1.setCombinerClass(Evaluation.Reduce.class);
        job1.setReducerClass(Evaluation.Reduce.class);
```

---

```

        FileInputFormat.setInputDirRecursive(job1,true);
        job1.setOutputKeyClass(Text.class);//reduce 阶段的输出的 key
        job1.setOutputValueClass(Text.class);//reduce 阶段的输出的
value
        FileInputFormat.addInputPath(job1, new Path(args[0]));
        FileOutputFormat.setOutputPath(job1, new Path(args[1]));
        boolean isSuccess = job1.waitForCompletion(true);
        if(!isSuccess) {
            System.exit(1);
        }

        Path outputPath2 = new Path(args[3]);
        if(hdfs.exists(outputPath2))
            hdfs.delete(outputPath2, true);
        Job job2 =Job.getInstance(conf, "ClassifiedDocOfClass");
        job2.setJarByClass(Evaluation.class);
        job2.setMapperClass(Evaluation.ClassifiedDocOfClassMap.class);
        job2.setCombinerClass(Evaluation.Reduce.class);
        job2.setReducerClass(Evaluation.Reduce.class);
        FileInputFormat.setInputDirRecursive(job2,true);
        job2.setOutputKeyClass(Text.class);//reduce 阶段的输出的 key
        job2.setOutputValueClass(Text.class);//reduce 阶段的输出的
value
        FileInputFormat.addInputPath(job2, new Path(args[2]));
        FileOutputFormat.setOutputPath(job2, new Path(args[3]));
        //System.exit(job2.waitForCompletion(true) ? 0 : 1);
        isSuccess = job2.waitForCompletion(true);
        if(!isSuccess) {
            System.exit(1);
        }

        GetEvaluation(conf,                args[1]+"/part-r-00000",
        args[3]+"/part-r-00000");

/**
 * 得到原本的文档分类
 * 输入:初始数据集合,格式为<<ClassName Doc>,word1 word2...>
 * 输出:原本的文档分类, 即<ClassName,Doc>
 */
public static class OriginalDocOfClassMap extends
Mapper<LongWritable, Text, Text, Text> {
    private Text newKey = new Text();
    private Text newValue = new Text();

```

---

```

        public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException{
            // 分词:将每行的单词进行分割,按照"\t\n\r\f" (空格、制表符、换行符、
回车符、换页)进行分割
            String[] result = value.toString().split("\\s");
            String className = result[0]; // 类名
            String docName = result[1]; // 文档名
            newKey.set(className);
            newValue.set(docName);
            context.write(newKey, newValue);
            System.out.println(newKey + "\t" + newValue);
        }
    }

    /**
     * 得到经贝叶斯分分类器分类后的文档分类
     * 读取经贝叶斯分类器分类后的结果文档<Doc,ClassName 概率>,并将其转化为
<ClassName,Doc>形式
     */
    public static class ClassifiedDocOfClassMap extends
Mapper<LongWritable, Text, Text, Text>{
        private Text newKey = new Text();
        private Text newValue = new Text();
        public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException{
            // 分词:将每行的单词进行分割,按照"\t\n\r\f" (空格、制表符、换行符、
回车符、换页)进行分割
            String[] result = value.toString().split("\\s");
            String docName = result[0]; // 文档名
            String className = result[1]; // 类名
            newKey.set(className);
            newValue.set(docName);
            context.write(newKey, newValue);
        }
    }

    public static class Reduce extends Reducer<Text, Text, Text, Text>
    {
        private Text result = new Text();
        public void reduce(Text key, Iterable<Text>values, Context
context) throws IOException, InterruptedException{
            //生成文档列表
            StringBuffer fileList = new StringBuffer();
            for(Text value:values){
                fileList.append(value + "\t");
            }
        }
    }

```

---

```

        }
        result.set(fileList.toString());
        context.write(key, result);
    }
}

/**
 * 第一个 MapReduce 计算得出初始情况下各个类有哪些文档,第二个 MapReduce
计算得出经贝叶斯分类后各个类有哪些文档
 * 此函数作用就是统计初始情况下的分类和贝叶斯分类两种情况下各个类公有的文
档数目(即针对各个类分类正确的文档数目 TP)
 * 初始情况下的各个类总数目减去分类正确的数目即为原本正确但分类错误的数目
(FN = OriginalCounts-TP)
 * 贝叶斯分类得到的各个类的总数目减去分类正确的数目即为原本不属于该类但分
到该类的数目(FP = ClassifiedCounts - TP)
 */
//Precision 精度:  $P = TP / (TP + FP)$ 
//Recall 精度:  $R = TP / (TP + FN)$ 
//P 和 R 的调和平均:  $F1 = 2PR / (P + R)$ 
//针对所有类别:
//Macroaveraged(宏平均) precision:  $(p1 + p2 + \dots + pN) / N$ 
//Microaveraged(微平均) precision: 对应各项相加再计算总的 P、R 值
public static void GetEvaluation(Configuration conf, String
ClassifiedDocOfClassFilePath, String OriginalDocOfClassFilePath)
throws IOException{

    //原始文档
    FileSystem fs1 =
    FileSystem.get(URI.create(OriginalDocOfClassFilePath), conf);
    FSDataInputStream fsr1 = fs1.open(new
    Path(OriginalDocOfClassFilePath));
    BufferedReader reader1 = new BufferedReader(new
    InputStreamReader(fsr1));

    //分类后的文档
    FileSystem fs2 =
    FileSystem.get(URI.create(ClassifiedDocOfClassFilePath), conf);
    FSDataInputStream fsr2 = fs2.open(new
    Path(ClassifiedDocOfClassFilePath));
    BufferedReader reader2 = new BufferedReader(new
    InputStreamReader(fsr2));

    ArrayList<String> ClassNames = new ArrayList<String>();    //
依次得到分类的类名

```



---

```

        ArrayList<Integer> TruePositive = new ArrayList<Integer>(); //
TP,记录真实情况和经分类后,正确分类的文档数目
        ArrayList<Integer> FalseNegative = new
ArrayList<Integer>(); // FN,记录属于该类但是没有分到该类的数目
        ArrayList<Integer> FalsePositive = new
ArrayList<Integer>(); // FP,记录不属于该类但是被分到该类的数目
        ArrayList<Double> precision = new ArrayList<Double>();
        ArrayList<Double> recall = new ArrayList<Double>();
        ArrayList<Double> F1 = new ArrayList<Double>();

        try{
            reader1 = new BufferedReader(new
InputStreamReader(fsrl)); //创建 Reader 对象
            String lineValue1 = null;
            String lineValue2 = null;
            while ((lineValue1 = reader1.readLine()) != null &&
(lineValue2 = reader2.readLine()) != null) { //按行读取
                // 分词: 将每行的单词进行分割,按照" \t\n\r\f" (空格、制表符、
                换行符、回车符、换页)进行分割
                String[] result1 = lineValue1.split("\\s");
                String[] result2 = lineValue2.split("\\s");
                //后面可以逐条记录处理 (因为每次读入的分类前后的类是相同的)
                //System.out.println(result1[0]+"\\t"+result2[0]);
                String className = result1[0];
                ClassNames.add(className);

                int TP = 0;
                for(int i=1; i<result2.length; i++){ // result2 是分类
文档
                    for(int j=1; j<result1.length; j++){
                        if(result2[i].equals(result1[j])){
                            TP++;
                        }
                    }
                }

                TruePositive.add(TP);
                int FP = result2.length - TP - 1; // FP = ClassifiedCounts
- TP, 减 1 是因为最开始的是类名
                int FN = result1.length - TP - 1; // FN = OriginalCounts
- TP, 减 1 是因为最开始的是类名
                FalsePositive.add(FP); // FP
                FalseNegative.add(FN); // TP
                double p = TP * 1.0 / ( TP + FP );
                double r = TP * 1.0 / ( TP + FN );

```

---

```

        double F = 2 * p * r / ( p + r );
        precision.add(p);
        recall.add(r);
        F1.add(F);
        System.out.println(className + "\t precision: " + p);
        System.out.println(className + "\t recall: " + r);
        System.out.println(className + "\t F1: " + F);
        System.out.println();
    }

    //Calculate MacroAverage
    double precisionSum = 0.0;
    double recallSum = 0.0;
    double F1Sum = 0.0;

    //Macroaveraged(宏平均) precision: (p1+p2+...+pN)/N
    for(int i=0; i<ClassNames.size(); i++){
        precisionSum += precision.get(i);
        recallSum += recall.get(i);
        F1Sum += F1.get(i);
    }

    int n = ClassNames.size();
    System.out.println("average precision: " + precisionSum
/n );

    System.out.println("average recall: " + recallSum / n );
    System.out.println("average F1: " + F1Sum / n );

    }finally{
        reader1.close();
        reader2.close();
    }
}
}
}

```

## 五、数据集说明

本工程中用到 4 个类别：AUSTR、EEC、JAP、SINGP。将每个类中的文档 70% 作为训练集，30% 作为测试集。具体的文档个数如下表所示：

表 5-1: 数据集说明

类别 \ 用途 文档数	训练集	测试集	总数
AUSTR	214	91	305
EEC	127	55	182
JAP	329	141	470
SINGP	96	41	137
总数	766	328	1094

## 六、程序运行说明

### 1. Job1

A. 运行时 Web 页面监控截图：

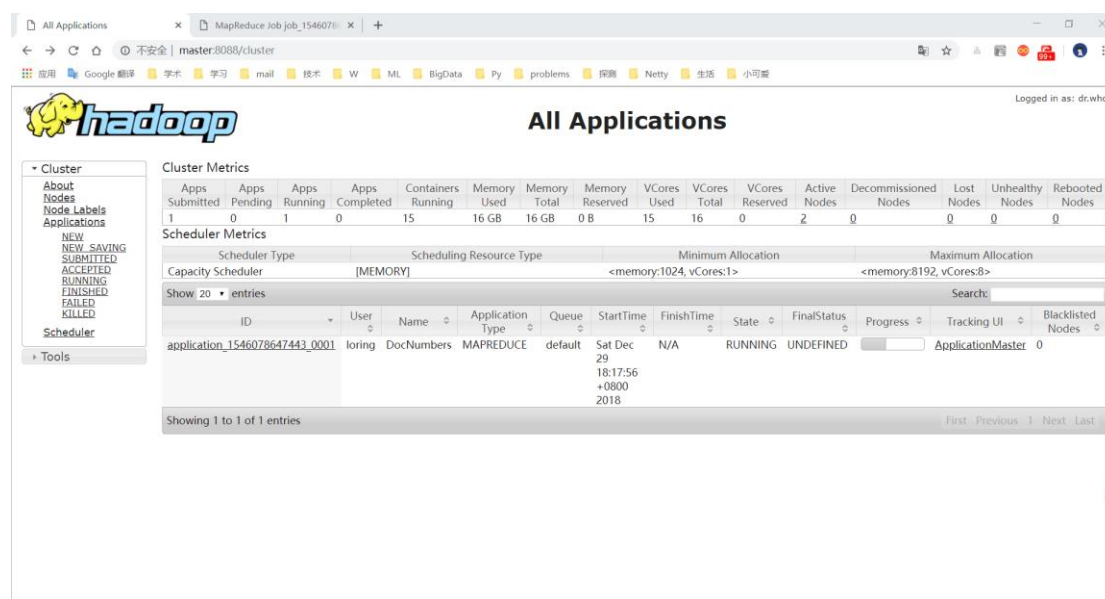
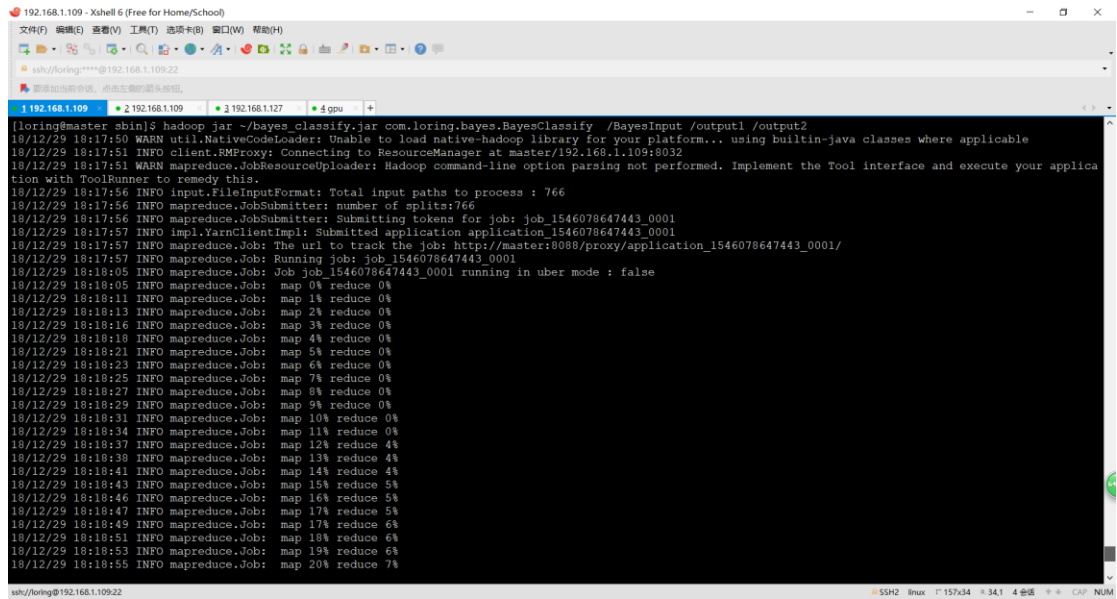


图 6.1.1: Job1 运行时 Web 页面监控截图

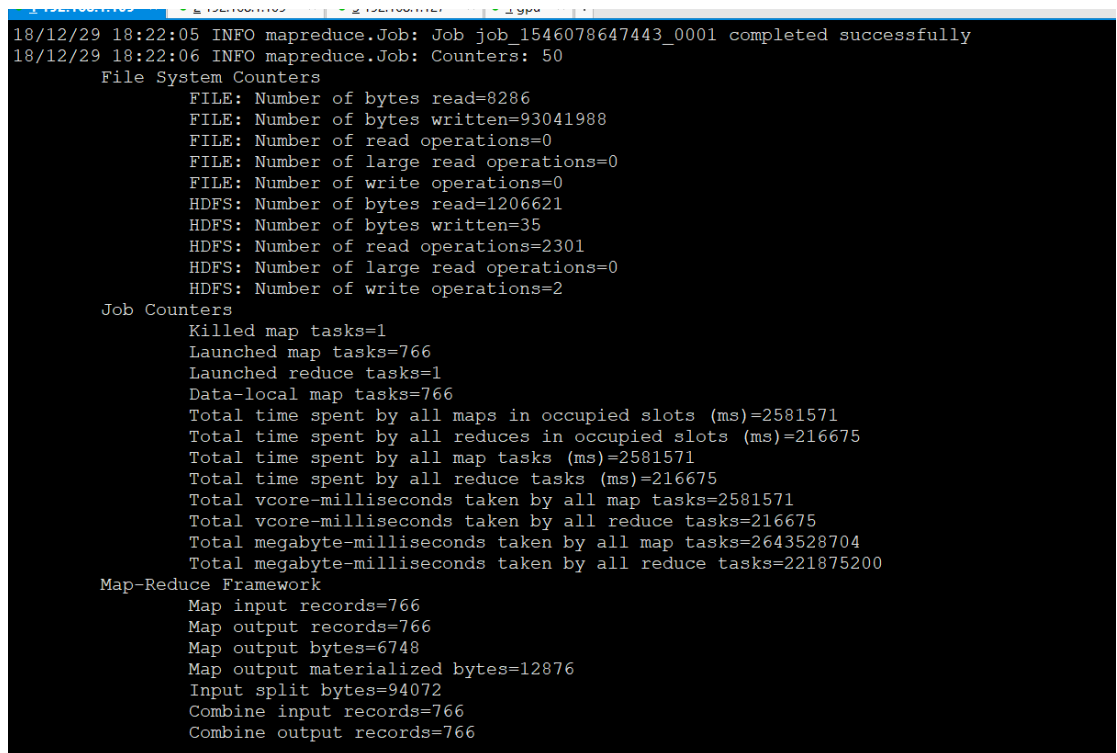
B. job1 运行截图：



```
[loring@master sbin]$ hadoop jar ~/bayes_classify.jar com.loring.bayes.BayesClassify /BayesInput /output1 /output2
18/12/29 18:17:50 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
18/12/29 18:17:51 INFO client.RMProxy: Connecting to ResourceManager at master/192.168.1.109:8032
18/12/29 18:17:51 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
18/12/29 18:17:56 INFO input.FileInputFormat: Total input paths to process : 766
18/12/29 18:17:56 INFO mapreduce.JobSubmitter: number of splits:766
18/12/29 18:17:56 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1546078647443_0001
18/12/29 18:17:57 INFO impl.YarnClientImpl: Submitted application application_1546078647443_0001
18/12/29 18:17:57 INFO mapreduce.Job: the url to track the job: http://master:8080/proxy/application_1546078647443_0001/
18/12/29 18:17:57 INFO mapreduce.Job: Running job: job_1546078647443_0001
18/12/29 18:18:05 INFO mapreduce.Job: Job job_1546078647443_0001 running in uber mode : false
18/12/29 18:18:05 INFO mapreduce.Job: map 0% reduce 0%
18/12/29 18:18:11 INFO mapreduce.Job: map 1% reduce 0%
18/12/29 18:18:13 INFO mapreduce.Job: map 2% reduce 0%
18/12/29 18:18:16 INFO mapreduce.Job: map 3% reduce 0%
18/12/29 18:18:18 INFO mapreduce.Job: map 4% reduce 0%
18/12/29 18:18:21 INFO mapreduce.Job: map 5% reduce 0%
18/12/29 18:18:23 INFO mapreduce.Job: map 6% reduce 0%
18/12/29 18:18:25 INFO mapreduce.Job: map 7% reduce 0%
18/12/29 18:18:27 INFO mapreduce.Job: map 8% reduce 0%
18/12/29 18:18:29 INFO mapreduce.Job: map 9% reduce 0%
18/12/29 18:18:31 INFO mapreduce.Job: map 10% reduce 0%
18/12/29 18:18:34 INFO mapreduce.Job: map 11% reduce 0%
18/12/29 18:18:37 INFO mapreduce.Job: map 12% reduce 0%
18/12/29 18:18:38 INFO mapreduce.Job: map 13% reduce 0%
18/12/29 18:18:41 INFO mapreduce.Job: map 14% reduce 0%
18/12/29 18:18:43 INFO mapreduce.Job: map 15% reduce 0%
18/12/29 18:18:46 INFO mapreduce.Job: map 16% reduce 0%
18/12/29 18:18:47 INFO mapreduce.Job: map 17% reduce 0%
18/12/29 18:18:49 INFO mapreduce.Job: map 17% reduce 0%
18/12/29 18:18:51 INFO mapreduce.Job: map 18% reduce 0%
18/12/29 18:18:53 INFO mapreduce.Job: map 19% reduce 0%
18/12/29 18:18:55 INFO mapreduce.Job: map 20% reduce 0%
```

图 6.1.2: Job1 运行截图

### C. map 和 reduce 数量截图:



```
18/12/29 18:22:05 INFO mapreduce.Job: Job job_1546078647443_0001 completed successfully
18/12/29 18:22:06 INFO mapreduce.Job: Counters: 50
File System Counters
  FILE: Number of bytes read=8286
  FILE: Number of bytes written=93041988
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=1206621
  HDFS: Number of bytes written=35
  HDFS: Number of read operations=2301
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=2
Job Counters
  Killed map tasks=1
  Launched map tasks=766
  Launched reduce tasks=1
  Data-local map tasks=766
  Total time spent by all maps in occupied slots (ms)=2581571
  Total time spent by all reduces in occupied slots (ms)=216675
  Total time spent by all map tasks (ms)=2581571
  Total time spent by all reduce tasks (ms)=216675
  Total vcore-milliseconds taken by all map tasks=2581571
  Total vcore-milliseconds taken by all reduce tasks=216675
  Total megabyte-milliseconds taken by all map tasks=2643528704
  Total megabyte-milliseconds taken by all reduce tasks=221875200
Map-Reduce Framework
  Map input records=766
  Map output records=766
  Map output bytes=6748
  Map output materialized bytes=12876
  Input split bytes=94072
  Combine input records=766
  Combine output records=766
```

图 6.1.3: job1 中 map 和 reduce 截图

Job1 共有 766 个 map 任务，1 个 reduce 任务。有 766 个 map 任务是因为训练集中共有 766 个文档，每个文档的大小都不超过 hadoop 的默认分片大小（128MB），所以每个文档不会被拆成多个分片。因此对于这个拥有 766 个输入文档的训练集来说，它作为 job1 的输入，就有 766 个分片（每个文档为一个分片），每个分片对应一个 map 任务，所以 job1 有 766 个 map 任务。Reduce 任务

的个数是自己设定的，这里设定为 1 个 reduce 任务。

## 2. Job2

### A. 运行时 Web 页面监控截图：

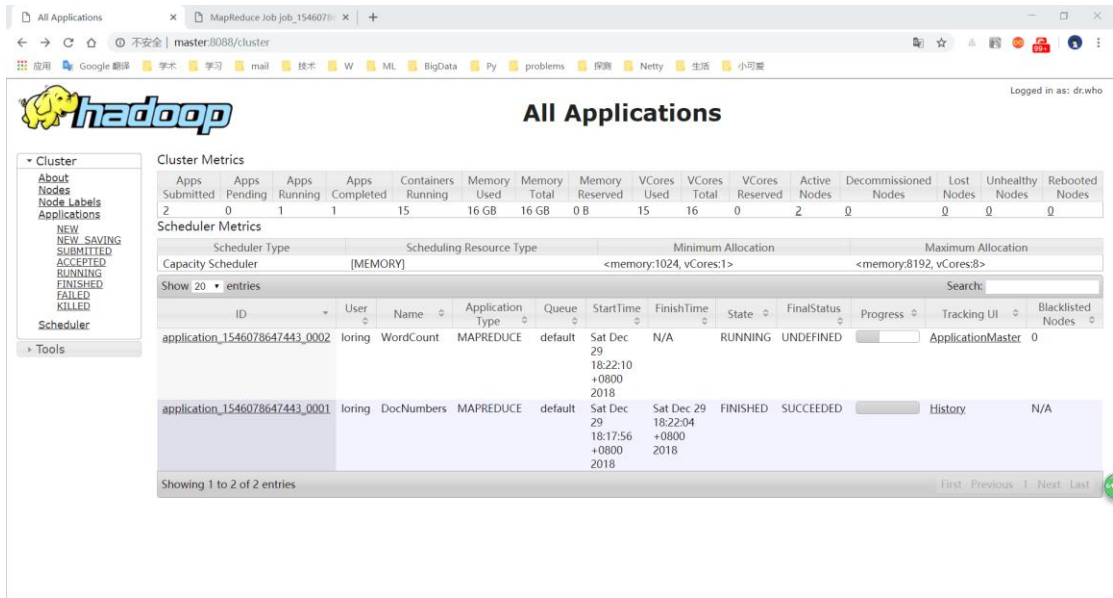


图 6.2.1: Job2 运行时 Web 页面监控截图

### B. Job2 运行截图：

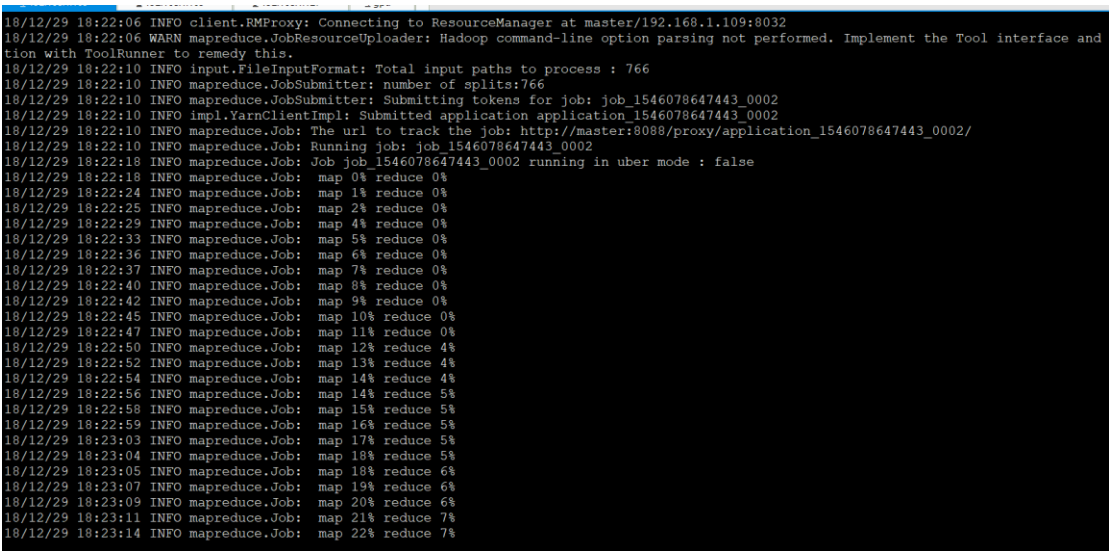


图 6.2.2: Job2 运行截图

### C. map 和 reduce 数量截图：

```
FILE: Number of bytes written=96181163
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=1206621
HDFS: Number of bytes written=422948
HDFS: Number of read operations=2301
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
Job Counters
  Launched map tasks=766
  Launched reduce tasks=1
  Data-local map tasks=766
  Total time spent by all maps in occupied slots (ms)=2493630
  Total time spent by all reduces in occupied slots (ms)=222070
  Total time spent by all map tasks (ms)=2493630
  Total time spent by all reduce tasks (ms)=222070
  Total vcore-milliseconds taken by all map tasks=2493630
  Total vcore-milliseconds taken by all reduce tasks=222070
  Total megabyte-milliseconds taken by all map tasks=2553477120
  Total megabyte-milliseconds taken by all reduce tasks=227399680
Map-Reduce Framework
  Map input records=131694
  Map output records=131694
  Map output bytes=2151733
  Map output materialized bytes=1652644
  Input split bytes=94072
  Combine input records=131694
  Combine output records=88708
  Reduce input groups=27305
  Reduce shuffle bytes=1652644
  Reduce input records=88708
  Reduce output records=27305
  Spilled Records=177416
```

图 6.2.3: job2 中 map 和 reduce 截图

Job2 共有 766 个 map 任务，1 个 reduce 任务。有 766 个 map 任务是因为训练集中共有 766 个文档，每个文档的大小都不超过 hadoop 的默认分片大小（128MB），所以每个文档不会被拆成多个分片。因此对于这个拥有 766 个输入文档的训练集来说，它作为 job1 的输入，就有 766 个分片（每个文档为一个分片），每个分片对应一个 map 任务，所以 job1 有 766 个 map 任务。Reduce 任务的个数是自己设定的，这里设定为 1 个 reduce 任务。

### 3. Job3

A. 运行时 Web 页面监控截图：

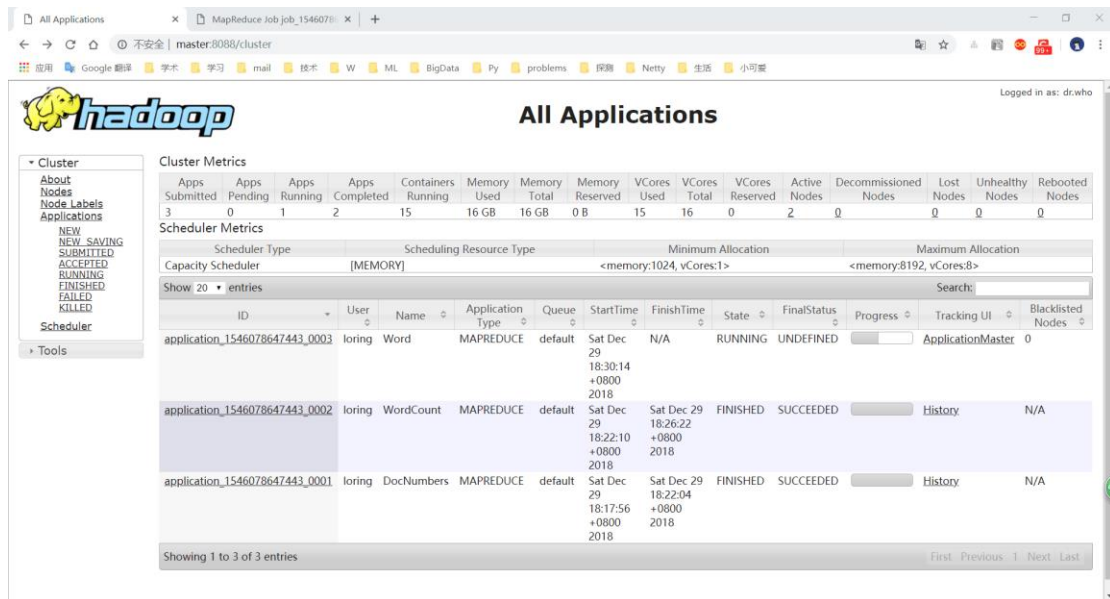


图 6.3.1: Job3 运行时 Web 页面监控截图

## B. job1 运行截图:

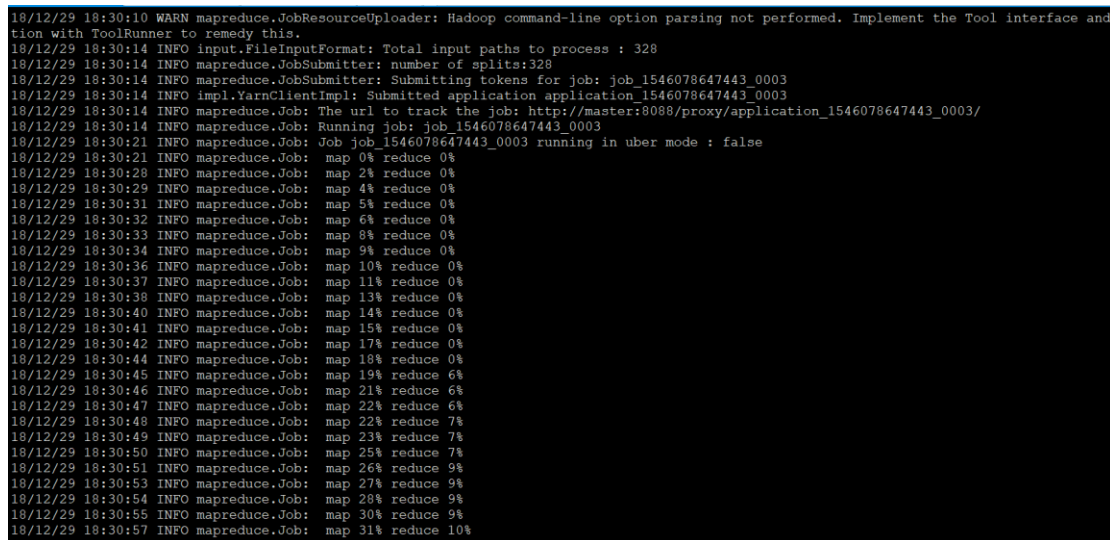


图 6.3.2: Job1 运行截图

### C. map 和 reduce 数量截图:

```
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=445979
HDFS: Number of bytes written=365206
HDFS: Number of read operations=987
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
Job Counters
  Killed map tasks=1
  Launched map tasks=328
  Launched reduce tasks=1
  Data-local map tasks=328
  Total time spent by all maps in occupied slots (ms)=1108120
  Total time spent by all reduces in occupied slots (ms)=89306
  Total time spent by all map tasks (ms)=1108120
  Total time spent by all reduce tasks (ms)=89306
  Total vcore-milliseconds taken by all map tasks=1108120
  Total vcore-milliseconds taken by all reduce tasks=89306
  Total megabyte-milliseconds taken by all map tasks=1134714880
  Total megabyte-milliseconds taken by all reduce tasks=91449344
Map-Reduce Framework
  Map input records=48629
  Map output records=48629
  Map output bytes=1427353
  Map output materialized bytes=368738
  Input split bytes=39952
  Combine input records=48629
  Combine output records=328
  Reduce input groups=328
  Reduce shuffle bytes=368738
  Reduce input records=328
  Reduce output records=328
  Spilled Records=656
```

图 6.3.3: job1 中 map 和 reduce 截图

Job3 共有 328 个 map 任务，1 个 reduce 任务。有 328 个 map 任务是因为训练集中共有 328 个文档，每个文档的大小都不超过 hadoop 的默认分片大小（128MB），所以每个文档不会被拆成多个分片。因此对于这个拥有 328 个输入文档的训练集来说，它作为 job3 的输入，就有 328 个分片（每个文档为一个分片），每个分片对应一个 map 任务，所以 job3 有 328 个 map 任务。Reduce 任务的个数是自己设定的，这里设定为 1 个 reduce 任务。

## 4. Job4

### A. 运行时 Web 页面监控截图:



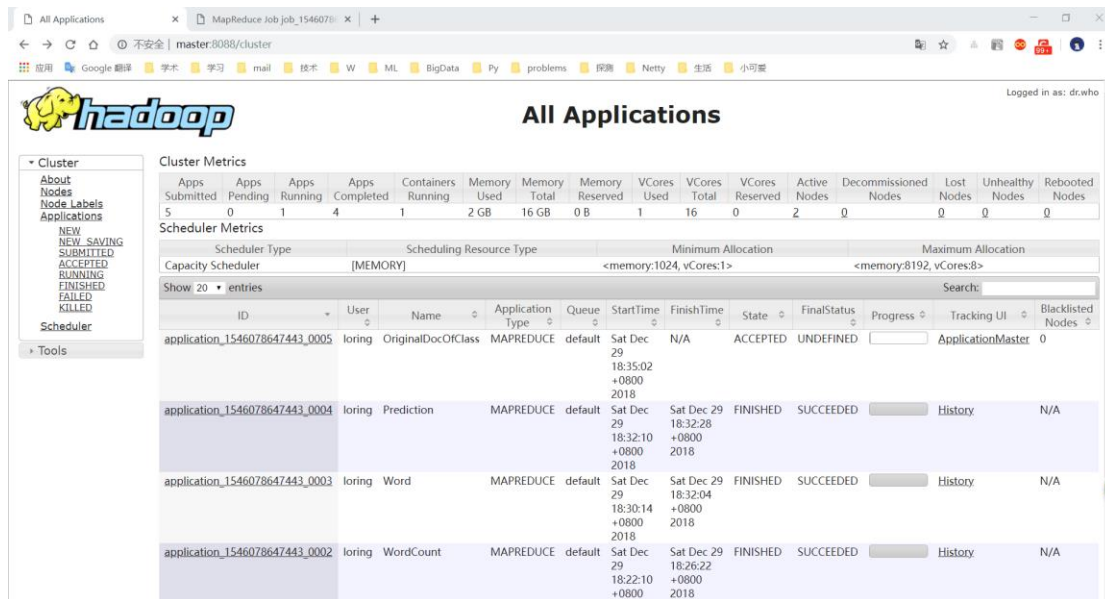


图 6.4.1: Job4 运行时 Web 页面监控截图

## B. Job4 运行截图:

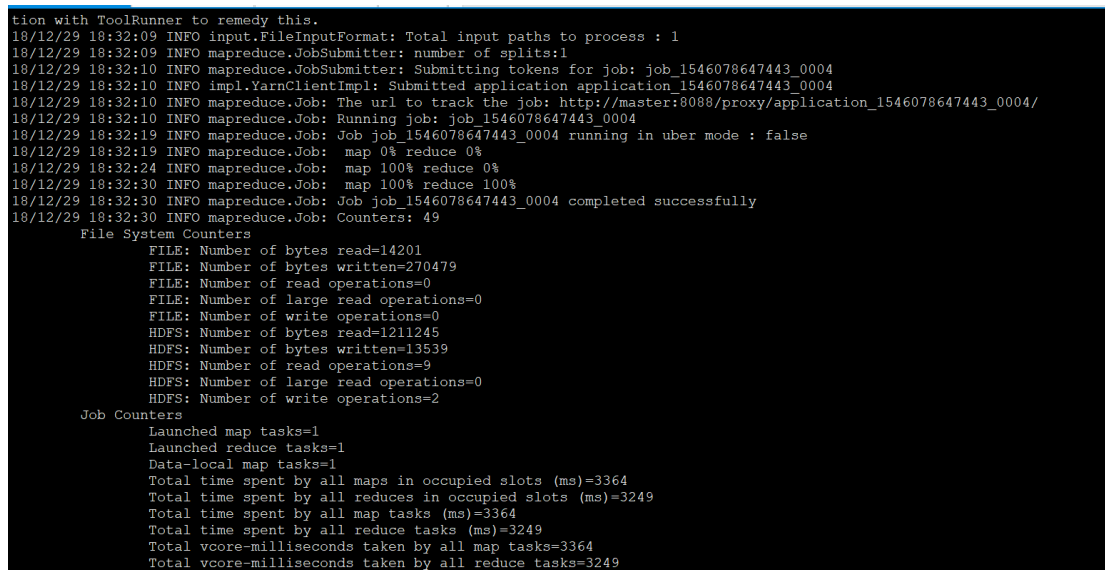


图 6.4.2: Job4 运行截图

## C. map 和 reduce 数量截图:

```

tion with ToolRunner to remedy this.
18/12/29 18:32:09 INFO input.FileInputFormat: Total input paths to process : 1
18/12/29 18:32:09 INFO mapreduce.JobSubmitter: number of splits:1
18/12/29 18:32:10 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1546078647443_0004
18/12/29 18:32:10 INFO impl.YarnClientImpl: Submitted application application_1546078647443_0004
18/12/29 18:32:10 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/application_1546078647443_0004/
18/12/29 18:32:10 INFO mapreduce.Job: Running job: job_1546078647443_0004
18/12/29 18:32:19 INFO mapreduce.Job: Job job_1546078647443_0004 running in uber mode : false
18/12/29 18:32:19 INFO mapreduce.Job: map 0% reduce 0%
18/12/29 18:32:24 INFO mapreduce.Job: map 100% reduce 0%
18/12/29 18:32:30 INFO mapreduce.Job: map 100% reduce 100%
18/12/29 18:32:30 INFO mapreduce.Job: Job job_1546078647443_0004 completed successfully
18/12/29 18:32:30 INFO mapreduce.Job: Counters: 49
File System Counters
FILE: Number of bytes read=14201
FILE: Number of bytes written=270479
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=1211245
HDFS: Number of bytes written=13539
HDFS: Number of read operations=9
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
Job Counters
Launched map tasks=1
Launched reduce tasks=1
Data-local map tasks=1
Total time spent by all maps in occupied slots (ms)=3364
Total time spent by all reduces in occupied slots (ms)=3249
Total time spent by all map tasks (ms)=3364
Total time spent by all reduce tasks (ms)=3249
Total vcore-milliseconds taken by all map tasks=3364
Total vcore-milliseconds taken by all reduce tasks=3249

```

图 6.4.3: job4 中 map 和 reduce 截图

Job4 共有 1 个 map 任务，1 个 reduce 任务。有 1 个 map 任务是因为 job4 的输入为 out3，out3 只有一个文件，且该文档的大小都不超过 hadoop 的默认分片大小（128MB），所以每个文档不会被拆成多个分片。因此对于这个拥有 1 个输入文档的训练集来说，它作为 job4 的输入，就有 1 个分片（每个文档为一个分片），也就对应一个 map 任务，所以 job1 有 1 个 map 任务。Reduce 任务的个数是自己设定的，这里设定为 1 个 reduce 任务。

## 5. Job5

### A. 运行时 Web 页面监控截图：

The screenshot shows the Hadoop All Applications web interface. The top navigation bar includes links for Cluster, About, Nodes, Node Labels, Applications, NEW, NEW SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED, Scheduler, and Tools. The main content area displays Cluster Metrics and a table of Applications.

Cluster Metrics															
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
5	0	1	4	1	2 GB	16 GB	0 B	1	16	0	2	0	0	0	0

Scheduler Metrics													
Scheduler Type		Scheduling Resource Type		Minimum Allocation		Maximum Allocation							
Capacity Scheduler		[MEMORY]		<memory:1024, vCores:1>		<memory:8192, vCores:8>							
Show 20 entries													
ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes		
application_1546078647443_0005	loring	OriginalDocOfClass	MAPREDUCE	default	Sat Dec 29 18:35:02 +0800 2018	N/A	ACCEPTED	UNDEFINED		ApplicationMaster	0		
application_1546078647443_0004	loring	Prediction	MAPREDUCE	default	Sat Dec 29 18:32:28 +0800 2018	Sat Dec 29 18:32:28 +0800 2018	FINISHED	SUCCEEDED		History	N/A		
application_1546078647443_0003	loring	Word	MAPREDUCE	default	Sat Dec 29 18:30:14 +0800 2018	Sat Dec 29 18:32:04 +0800 2018	FINISHED	SUCCEEDED		History	N/A		
application_1546078647443_0002	loring	WordCount	MAPREDUCE	default	Sat Dec 29 18:22:10 +0800 2018	Sat Dec 29 18:26:22 +0800 2018	FINISHED	SUCCEEDED		History	N/A		

图 6.5.1: Job5 运行时 Web 页面监控截图

## B. Job5 运行截图:

```
tion with ToolRunner to remedy this.
18/12/29 18:35:01 INFO input.FileInputFormat: Total input paths to process : 1
18/12/29 18:35:01 INFO mapreduce.JobSubmitter: number of splits:1
18/12/29 18:35:02 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1546078647443_0005
18/12/29 18:35:02 INFO impl.YarnClientImpl: Submitted application application_1546078647443_0005
18/12/29 18:35:02 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/application_1546078647443_0005/
18/12/29 18:35:02 INFO mapreduce.Job: Running job: job_1546078647443_0005
18/12/29 18:35:09 INFO mapreduce.Job: Job job_1546078647443_0005 running in uber mode : false
18/12/29 18:35:09 INFO mapreduce.Job: map 0% reduce 0%
18/12/29 18:35:14 INFO mapreduce.Job: map 100% reduce 0%
18/12/29 18:35:20 INFO mapreduce.Job: map 100% reduce 100%
18/12/29 18:35:20 INFO mapreduce.Job: Job job_1546078647443_0005 completed successfully
18/12/29 18:35:20 INFO mapreduce.Job: Counters: 49
    File System Counters
        FILE: Number of bytes read=5630
        FILE: Number of bytes written=253335
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=365314
        HDFS: Number of bytes written=5604
        HDFS: Number of read operations=6
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
    Job Counters
        Launched map tasks=1
        Launched reduce tasks=1
        Data-local map tasks=1
        Total time spent by all maps in occupied slots (ms)=3603
        Total time spent by all reduces in occupied slots (ms)=3260
        Total time spent by all map tasks (ms)=3603
        Total time spent by all reduce tasks (ms)=3260
        Total vcore-milliseconds taken by all map tasks=3603
        Total vcore-milliseconds taken by all reduce tasks=3260
```

图 6.5.2: Job5 运行截图

## C. map 和 reduce 数量截图:

```
tion with ToolRunner to remedy this.
18/12/29 18:35:01 INFO input.FileInputFormat: Total input paths to process : 1
18/12/29 18:35:01 INFO mapreduce.JobSubmitter: number of splits:1
18/12/29 18:35:02 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1546078647443_0005
18/12/29 18:35:02 INFO impl.YarnClientImpl: Submitted application application_1546078647443_0005
18/12/29 18:35:02 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/application_1546078647443_0005/
18/12/29 18:35:02 INFO mapreduce.Job: Running job: job_1546078647443_0005
18/12/29 18:35:09 INFO mapreduce.Job: Job job_1546078647443_0005 running in uber mode : false
18/12/29 18:35:09 INFO mapreduce.Job: map 0% reduce 0%
18/12/29 18:35:14 INFO mapreduce.Job: map 100% reduce 0%
18/12/29 18:35:20 INFO mapreduce.Job: map 100% reduce 100%
18/12/29 18:35:20 INFO mapreduce.Job: Job job_1546078647443_0005 completed successfully
18/12/29 18:35:20 INFO mapreduce.Job: Counters: 49
    File System Counters
        FILE: Number of bytes read=5630
        FILE: Number of bytes written=253335
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=365314
        HDFS: Number of bytes written=5604
        HDFS: Number of read operations=6
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
    Job Counters
        Launched map tasks=1
        Launched reduce tasks=1
        Data-local map tasks=1
        Total time spent by all maps in occupied slots (ms)=3603
        Total time spent by all reduces in occupied slots (ms)=3260
        Total time spent by all map tasks (ms)=3603
        Total time spent by all reduce tasks (ms)=3260
        Total vcore-milliseconds taken by all map tasks=3603
        Total vcore-milliseconds taken by all reduce tasks=3260
```

图 6.5.3: job5 中 map 和 reduce 截图

Job5 共有 1 个 map 任务, 1 个 reduce 任务。有 1 个 map 任务是因为 job5 的输入为 out3, out3 只有一个文件, 且该文档的大小都不超过 hadoop 的默认分片大小 (128MB), 所以每个文档不会被拆成多个分片。因此对于这个拥有 1 个输入文档的训练集来说, 它作为 job5 的输入, 就有 1 个分片 (每个文档为一个分片), 也就对应一个 map 任务, 所以 job5 有 1 个 map 任务。Reduce 任务的个数是自己设定的, 这里设定为 1 个 reduce 任务。

## 6. Job6

### A. 运行时 Web 页面监控截图：

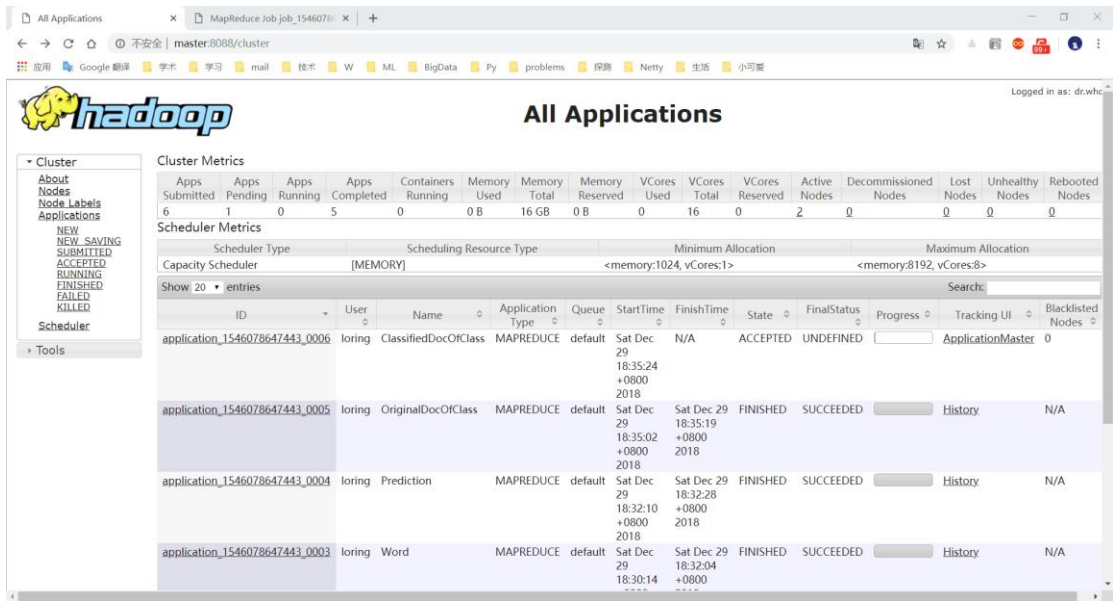


图 6.6.1: Job6 运行时 Web 页面监控截图

### B. Job6 运行截图：

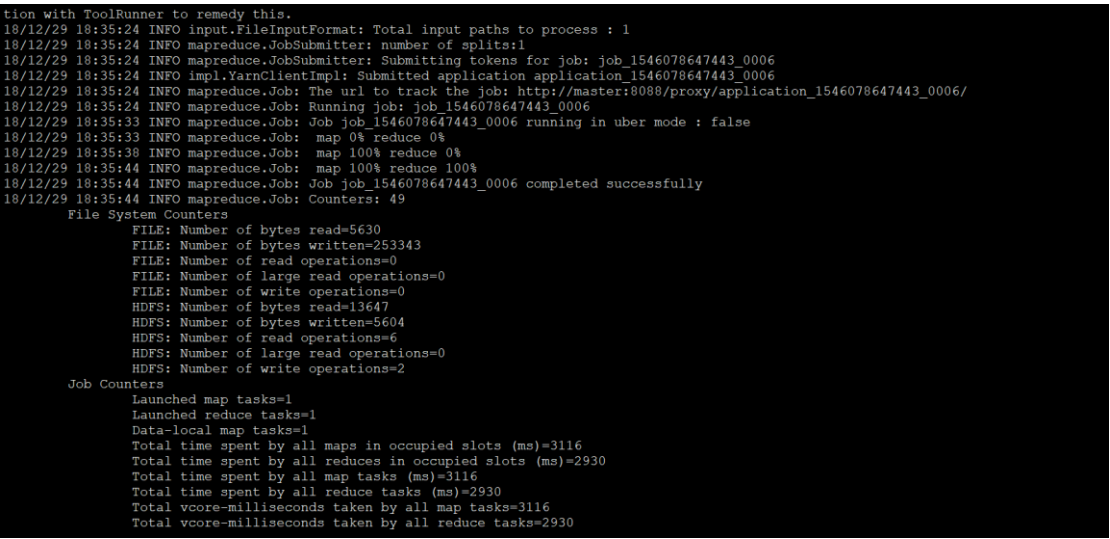


图 6.6.2: Job6 运行截图

### C. map 和 reduce 数量截图：

```

tion with ToolRunner to remedy this.
18/12/29 18:35:24 INFO input.FileInputFormat: Total input paths to process : 1
18/12/29 18:35:24 INFO mapreduce.JobSubmitter: number of splits:1
18/12/29 18:35:24 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1546078647443_0006
18/12/29 18:35:24 INFO impl.YarnClientImpl: Submitted application application_1546078647443_0006
18/12/29 18:35:24 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/application_1546078647443_0006/
18/12/29 18:35:24 INFO mapreduce.Job: Running job: job_1546078647443_0006
18/12/29 18:35:33 INFO mapreduce.Job: Job job_1546078647443_0006 running in uber mode : false
18/12/29 18:35:33 INFO mapreduce.Job:  map 0% reduce 0%
18/12/29 18:35:38 INFO mapreduce.Job:  map 100% reduce 0%
18/12/29 18:35:44 INFO mapreduce.Job:  map 100% reduce 100%
18/12/29 18:35:44 INFO mapreduce.Job: Job job_1546078647443_0006 completed successfully
18/12/29 18:35:44 INFO mapreduce.Job: Counters: 49
  File System Counters
    FILE: Number of bytes read=5630
    FILE: Number of bytes written=253343
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=13647
    HDFS: Number of bytes written=5604
    HDFS: Number of read operations=6
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=2
  Job Counters
    Launched map tasks=1
    Launched reduce tasks=1
    Data-local map tasks=1
    Total time spent by all maps in occupied slots (ms)=3116
    Total time spent by all reduces in occupied slots (ms)=2930
    Total time spent by all map tasks (ms)=3116
    Total time spent by all reduce tasks (ms)=2930
    Total vcore-milliseconds taken by all map tasks=3116
    Total vcore-milliseconds taken by all reduce tasks=2930

```

图 6.1.2: job6 中 map 和 reduce 截图

Job6 共有 1 个 map 任务，1 个 reduce 任务。有 1 个 map 任务是因为 job4 的输入为 out4，out4 只有一个文件，且该文档的大小都不超过 hadoop 的默认分片大小（128MB），所以每个文档不会被拆成多个分片。因此对于这个拥有 1 个输入文档的训练集来说，它作为 job6 的输入，就有 1 个分片（每个文档为一个分片），也就对应一个 map 任务，所以 job1 有 1 个 map 任务。Reduce 任务的个数是自己设定的，这里设定为 1 个 reduce 任务。

## 七：实验结果分析

计算分类结果的 Precision，Recall 和 F1 值。

如下图所示，程序运行到最后通过命令行输出了 AUSTR、EEC、JAP、SINGPs 这四个类的 precision、recall、F1 值，和通过宏平均算法算出的这个分类器的 precision、recall、F1 值。

```

AUSTR precision: 0.6373626373626373
AUSTR recall: 1.0
AUSTR F1: 0.7785234899328859

EEC precision: 0.9818181818181818
EEC recall: 1.0
EEC F1: 0.9908256880733944

JAP precision: 0.8581560283687943
JAP recall: 0.983739837398374
JAP F1: 0.9166666666666667

SINGP precision: 1.0
SINGP recall: 0.44086021505376344
SINGP F1: 0.6119402985074627

average precision: 0.8693342118874035
average recall: 0.8561500131130344
average F1: 0.8244890357951025

```

图 7.1: 工程输出评价标准数值截图

下表列出了工程中用到的 4 个类的 precision, recall, F1 值，以及通过宏平均算出的整个分类器的 precision, recall, F1 值。

表 7-1: 工程评价标准数值

类别 \ 标准	Precision	Recall	F1
AUSTR	63.74%	100.00%	77.85%
EEC	98.18%	100.00%	99.08%
JAP	85.82%	98.37%	91.67%
SINGP	100.00%	44.09%	61.19%
宏平均	86.94%	85.62%	82.45%

由表中数据可以看出，本工程中实现的 bayes 分类器可以准确地将大多数测试文档正确地分类，效果好、可用性高。同时也有可以提升的空间，未来可以通过词项的归一化、去掉停用词等操作来进一步提高 bayes 分类器的效果。