基于 MapReduce 的朴素贝叶斯分类器

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

本次实验用朴素贝叶斯方法给文本文件分类，即给定一个类标签集合 $C=\{c_1,c_2,\ldots,c_j\}$ 以及一个文档 d，给文档 d 分配一个最合适的类别标签 $c_i(i=1,\ldots,j)$。

解决方法的基本思想就是对于类标签集合 C 中的每个类标签 c_i $(i=1,\ldots,j)$, 计算条件概率 p (ci |d)，使条件概率 p (ci |d)最大的类别作为文档 d 最终的类别。

现在将文本分类问题变成计算条件概率 p (ci |d)，要计算条件概率，就要用到概率论中的贝叶斯(Bayes)公式，这也是该方法名称的由来。贝叶斯公式如下所示：

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

公式中的：p (ci|d)为后验概率或条件概率(posterior)、p (ci): 先验概率(prior)、p (d|ci): 似然概率(likelihood)、p (d): 证据(evidence)。从公式中可以观察到，当 p(d)是一定值时，后验概率 p (ci|d)取决于似然概率 p (d|ci)和先验概率 p (ci)。故贝叶斯可变为：

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

由公式可知，后验概率 p (ci|d)与似然概率 p (d|ci)和先验概率 p (ci)的乘积成正比，要计算后验概率 p (ci|d)的最大值，只需计算 $p(d|c_i)p(c_i)$最大值即

可。

首先先验概率 $p(c_i)$有：

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

为了计算似然概率 $p(d|c_i)$，需要 Term 独立性假设，即文档中每个 term 的出现都是彼此独立的，基与这个假设，基与这个假设，似然概率 $p(d|c_i)$：

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

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

因此，贝叶斯分类器是通过用训练集数据来计算先验概率 $p(c_i)$和似然概率 $p(d|c_i)$，然后利用贝叶斯公式来预测文档 d 应该所属的类别标签 c_i。

三、贝叶斯分类器训练的 MapReduce 算法设计

该算法实现过程流程和框架在 Main 这个类中定义，主要分三部分：第一部分是训练过程，对训练集中数据进行训练，即求出每个类中文件数和每个类中的单词种类和数目，并将结果保存在相应文件夹下；第二部分是预测过程，先对测试测试集数据进行处理，然后通过第一部分的训练过程结果求出先验概率和条件概率，然后通过贝叶斯公式找后验概率的最大值来预测测试集文件的类别；第三部分是评估过程，通过第二部分预测的结果和每个预测集文件真实类别的结果来求出精确率 precision、召回率 recall 和 F1，以此判断该系统对文件分类的可靠性。

该系统共使用了 4 个 mapreduce 的过程，记为 job1、job2、job3、job4。在第一部分训练过程用了 job1 和 job2，job1 是用来统计每个类的文件数目，job2 的作用是统计每个类中出现单词数量。在第二部分预测过程中有 job3 和 job4，job3 是 job4 的一个准备过程，是对测试集数据进行欲处理，

job4 过程用贝叶斯公式原理来预测文件类别。第三部分评估过程没有用mapreduce 程序，而是直接用单机 java 程序读取 job4 中的预测结果进行评估该系统的预测可靠性。

1.Job1

第一个 job 是用来统计训练集中每个类中的文件数目，要统计文件数目，在输入过程中文件就不应该被切分，因此要写一个 FileInputFormat 的子类（命名为 WholeFileInputFormat），该子类首先实现 isSplitable 为 false，然后重写其 RecordReader 方法，使其输入到 mapper 的 key 为文件类名，value 为文件内容。第一个 job 的输入文件是所有训练集，训练集下有四个文件夹，每个文件夹名都代表其类名，每个文件夹下有若干文件，这些文件都输入该类的。在 FileInputFormat 中文件不被切片，直接整个文件传入 map 中，map 输入的 key 值就是文件类名，因此 map 的处理过程比较简单，只需要将输出设为<文件类名，1>。Map 的输出丢到上下文中，会经过 shuffle 过程将 key 值相同的 value 合并为一个集合作为 reduce 的输入，在 reduce 中将合并的 value 集合进行数量统计得到的总和即是该类中文件的总数目，并将其作为新的 value 从 reduce 输出。最后 reduce 的输出保存到 hdfs：//master/output/output1 下的 part-r-00000 文档中，格式为：类名 文件数目。Job1 的 dataflow 示意图如下所示：

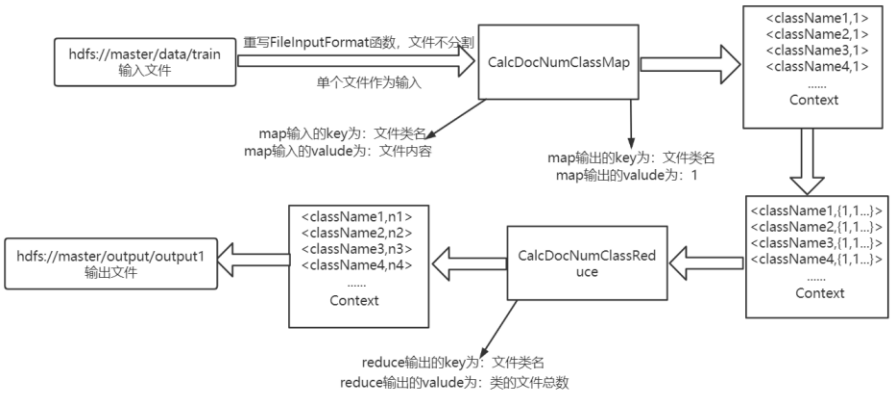

图 3.1：Job1 DataFlow 示意图

2.Job2

第二个 job 是用来统计每个类中出现的各个单词的总数，其输入和 Job1 一样都是训练集的所有文件。Job2 执行时会找到训练集中的每个文件，读取每个文件的内容并切片，将每行的偏移量和行内容作为该 job 中的 map 输入，map 获取每个文件的类名，并将每个 value 转行成相应的单词，将类名和单词拼接作为新的 key 值，最后 map 的输出为：<<类名，单词>，1>。Map 的输出丢到上下文中，经过 shuffle 处理，将相同 key 值的 value 合并为一个集合作为 reduce 的输入，在 reduce 中将合并的 value 集合进行数量统计得到的总和即是类中该单词出现的总次数，然后 reduce 的输出为<<类名，单词>，该单词出现次数>，将其写到输出文件 hdfs：//master/output/output3 的 part-r-00000 文档中。Job2 的 dataflow 示意图如下所示：

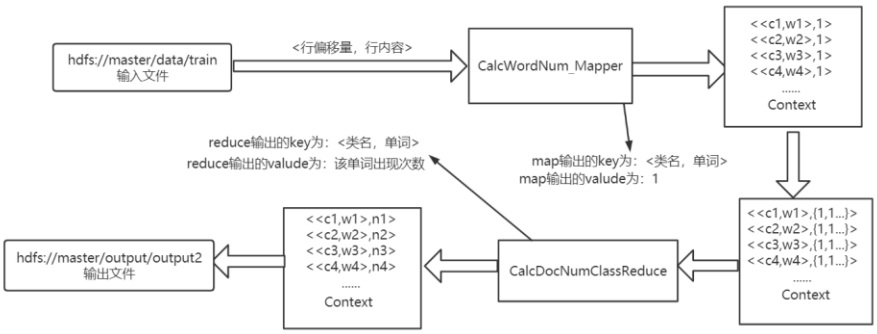

图 3.2：Job2 DataFlow 示意图

3.Job3

第三个 Job 是对测试集数据进行预处理，为接下来文件分类做准备，将测试集的文档内容汇总为<<类名，文档名>, 单词 1，单词 2, ……> 进行输出。该 Job 的输入是测试集的所有文件，读取每个文件的内容并切片，将每行的偏移量和行内容作为该 job 中的 map 输入，map 获取每个文件的类名和文件名，并将其拼接为新的 key 值输出，每个 value 转行成相应的单词，最后 map 的输出为：<<类名，文件名>，单词>。Map 的输出丢到上下文中，经过 shuffle 处理，将相同 key 值的 value 合并为一个集合作为 reduce 的输入，在 reduce 中将合并的 value 集合中的每一个单词拼接成一个字符串作为新的 value，则 reduce 的输出格式为：<<类名,文档名>，单词 1,单词 2,……>，并将其写到 hdfs：//master/output/output3。Job2 的 dataflow 示意图如下所示：

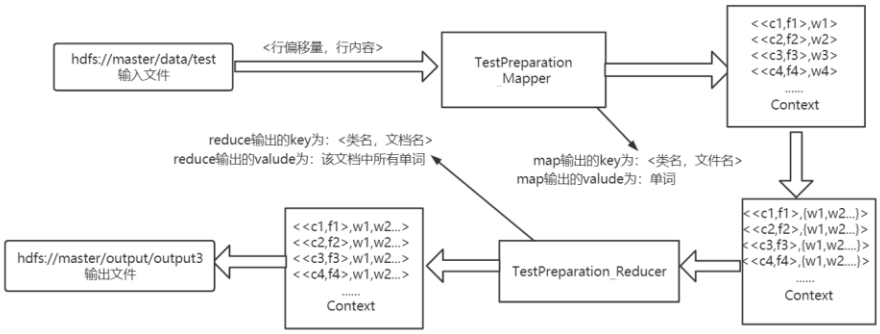

图 3.3：Job3 DataFlow 示意图

4.Job4

第四个 job 是对测试集中的文件类别进行预测，输入文件为 Job3 中的输出文件，输出格式为:<<真实类名，文件名>，预测类名>。在执行该 Job 前，首先要读入 Job1 和 Job2 的输出文件来计算先验概率和条件概率，并将其结果保存在 HashMap 中。Job4 的输入是 job3 的输出文件，首先读取该文档的内容并进行切片，将每行的偏移量和行内容作为该 job 中的 map 输入，map 获取每个文件的真实类名和文件名，并将其拼接为新的 key 值输出，并且通过 value 分词来获取每个单词在每个类下的条件概率，然后计算出该文件输入每个类的概率，所以 map 的输出为：<<真实类名，文件名>，<预测类名，概率>>。Map 的输出丢到上下文中，经过 shuffle 处理，将相同 key 值的 value 合并为一个集合作为 reduce 的输入，在 reduce 中选取概率最大的预测类名作为该文件的真正的预测类名并作为新的 value 输出，所以 reduce 的输出为：<<真实类别，文件名>，预测类别>，输出到 hdfs：//master/output/output4 中。Job4 的 dataflow 示意图如下所示：

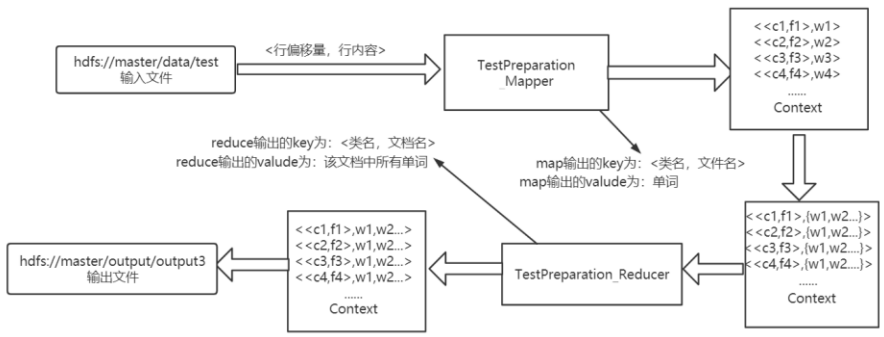

图 3.4：Job4 DataFlow 示意图

四：源代码清单

源代码共有 7 个 java 文件，分别是 Main、Utils、CalcDocNumClass、CalcWordNumClass、TestPreparation、TestPrediction、Evaluation，Main 文

件时入口文件，有程序流程任务调度作用，Utils 文件下存放路径参数(下面源码中未列出该文件)， CalcDocNumClass 、 CalcWordNumClass 、TestPreparation、TestPrediction 文件分别实现了 job1、job2、job3、job4，Evaluation 文件用于最后计算 precision、recall、F1。

1 Main.java 文件

```
package com.Bayes;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.util.ToolRunner;
public class Main {
    public static void main(String[] args) throws Exceptio
n {
        Configuration configuration = new Configuration();
        //统计每个类中文件数目
        CalcDocNumClass computeDocNumInClass = new CalcDoc
NumClass();
        ToolRunner.run(configuration, computeDocNumInClass
, args);
        //统计每个类中出现单词总数
        CalcWordNumClass calcWordNumClass = new CalcWordNu
mClass();
        ToolRunner.run(configuration, calcWordNumClass, ar
gs);
        //测试集数据预处理
        TestPreparation testPreparation = new TestPreparat
ion();
        ToolRunner.run(configuration, testPreparation, arg
s);
        //预测测试集文件类别
        TestPrediction testPrediction = new TestPrediction
();
        ToolRunner.run(configuration, testPrediction, args
);
        //评估测试效果，计算 precision，recall，F1
```

```
        Evaluation evaluation = new Evaluation();
        ToolRunner.run(configuration, evaluation, args);
    }
}
```

2 CalcDocNumClass.java 文件

```
package com.Bayes;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
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.FileInputForm
at;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFo
rmat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import java.io.IOException;

public class CalcDocNumClass  extends Configured implement
s Tool {
    /*
     * 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:
     */
    public static class CalcDocNumClassMap extends Mapper<
Text, BytesWritable, Text, IntWritable> {
//        private Text newKey = new Text();
        private final static IntWritable one = new IntWrit
able(1);
        public void map(Text key, BytesWritable value, Con
text context) throws IOException, InterruptedException{
```

```
            context.write(key, one);
        }
    }
    public static class CalcDocNumClassReduce extends Redu
cer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();
        public void reduce(Text key, Iterable<IntWritable>
 values, Context context) throws IOException, InterruptedE
xception{
            int sum = 0;
            for(IntWritable value:values){
                sum += value.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }
    @Override
    public int run(String[] strings) throws Exception {
        Configuration conf = getConf();
        FileSystem hdfs = FileSystem.get(conf);

        Path outputPath1 = new Path(Utils.DocNumClass);
        if(hdfs.exists(outputPath1))
            hdfs.delete(outputPath1, true);

        Job job1 =Job.getInstance(conf, "CalcDocNum");
        job1.setJarByClass(CalcDocNumClass.class);
        //设置输入输出格式
        job1.setInputFormatClass(WholeFileInputFormat.clas
s);
        job1.setMapperClass(CalcDocNumClassMap.class);
        job1.setCombinerClass(CalcDocNumClassReduce.class)
;
        job1.setReducerClass(CalcDocNumClassReduce.class);
```

```
        FileInputFormat.setInputDirRecursive(job1,true);
        job1.setOutputKeyClass(Text.class);
  //reduce 阶段的输出的 key
        job1.setOutputValueClass(IntWritable.class);
  //reduce 阶段的输出的 value
        FileInputFormat.addInputPath(job1, new Path(Utils.
TRAIN_DATA_PATH));
        FileOutputFormat.setOutputPath(job1, new Path(Util
s.DocNumClass));
        return job1.waitForCompletion(true) ? 0 : 1;
    }

    public static class WholeFileInputFormat extends FileI
nputFormat<Text, BytesWritable>{

        @Override
        protected boolean isSplitable(JobContext context,
Path filename) {
            return false;    //文件输入的时候不再切片
        }

        @Override
        public RecordReader<Text, BytesWritable> createRec
ordReader(InputSplit inputSplit, TaskAttemptContext taskAt
temptContext) throws IOException, InterruptedException {
            WholeFileRecordReader reader = new WholeFileRe
cordReader();
            reader.initialize(inputSplit, taskAttemptConte
xt);
            return reader;
        }

    }
```

```
    public static class WholeFileRecordReader extends Reco
rdReader<Text, 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 Text key = new Text();          //key 对象，
初始值为空
        private BytesWritable value = new BytesWritable();
 //value 对象，内容为空
        private boolean isRead = false;   //布尔变量记录记
录是否被处理过

        @Override
        public void initialize(InputSplit split, TaskAttem
ptContext 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 boolean nextKeyValue() throws IOException,
InterruptedException {
            if (!isRead) {  //如果记录有没有被处理过
                //定义缓存区
                byte[] contents = new byte[(int) fileSplit
.getLength()];
                FileSystem fs = null;
                FSDataInputStream fis = null;
                try {
                    //获取文件系统
                    Path path = fileSplit.getPath();
                    fs = path.getFileSystem(conf);
                    //读取数据
```

```
                    fis = fs.open(path);
                    //读取文件内容
                    IOUtils.readFully(fis, contents, 0, co
ntents.length);
                    //输出内容文件
                    value.set(contents, 0, contents.length
);
                    //获取文件所属类名称
                    String classname = fileSplit.getPath()
.getParent().getName();
                    key.set(classname);
                }catch (Exception e){
                    System.out.println(e);
                }finally {
                    IOUtils.closeStream(fis);
                }
                isRead = true;   //将是否处理标志设为 true，
下次调用该方法会返回 false
                return true;
            }
            else {
                return false;   //如果记录处理过，返回
false，表示 split 处理完毕
            }
        }
        @Override
        public Text getCurrentKey() throws IOException, In
terruptedException {
            return key;
        }
        @Override
        public BytesWritable getCurrentValue() throws IOEx
ception, InterruptedException {
            return value;
        }
```

```
        @Override
        public float getProgress() throws IOException {
            return isRead ? 1.0f : 0.0f;
        }
        @Override
        public void close() throws IOException {
        }
    }
    public static void main(String[] args) throws Exceptio
n {
        int res = ToolRunner.run(new Configuration(),
                new CalcDocNumClass(), args);
        System.exit(res);
    }
}
```

3 CalcWordNumClass.java 文件

```
package com.Bayes;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.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.FileInputForm
at;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFo
rmat;
```

```
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import java.io.IOException;
public class CalcWordNumClass extends Configured implement
s Tool {
    /*
     * 第二个 MapReduce 用于统计每个类下单词的数量
     */
    public static class CalcWordNum_Mapper extends Mapper<
LongWritable, Text, Text, IntWritable> {
        private Text key_out = new Text();
        private IntWritable one = new IntWritable(1);
        @Override
        protected void map(LongWritable key, Text value, C
ontext context) throws IOException, InterruptedException {
            InputSplit inputSplit = context.getInputSplit(
);
            String className = ((FileSplit)inputSplit).get
Path().getParent().getName();
            String line_context = value.toString();
            key_out.set(className + '\t' + line_context);
            context.write(key_out, one);
        }
    }
    public static class CalcWordNum_Reducer extends Reduce
r<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();
        @Override
        protected void reduce(Text key, Iterable<IntWritab
le> values, Context context) throws IOException, Interrupt
edException {
            int num = 0;
            for (IntWritable value: values){
                num += value.get();
            }
```

```
            result.set(num);
            context.write(key, result);
        }
    }
    @Override
    public int run(String[] strings) throws Exception {
        Configuration conf = getConf();
        Job job2 = Job.getInstance(conf, "CalcWordNum");
        FileSystem fileSystem = FileSystem.get(conf);
        Path outputPath2 = new Path(Utils.WordNumClass);
        if(fileSystem.exists(outputPath2))
            fileSystem.delete(outputPath2, true);
        //设置 jar 加载路径
        job2.setJarByClass(CalcWordNumClass.class);
        //设置 map 和 reduce 类
        job2.setMapperClass(CalcWordNum_Mapper.class);
        job2.setCombinerClass(CalcWordNum_Reducer.class);
        job2.setReducerClass(CalcWordNum_Reducer.class);
        //设置 map reduce 输出格式
        job2.setMapOutputKeyClass(Text.class);
        job2.setMapOutputValueClass(IntWritable.class);
        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(IntWritable.class);
        //设置输入和输出路径
        FileInputFormat.setInputDirRecursive(job2,true);
        FileInputFormat.addInputPath(job2, new Path(Utils.
TRAIN_DATA_PATH));
//        FileInputFormat.setInputPaths(job2, new Path(Uti
ls.TRAIN_DATA_PATH));
        FileOutputFormat.setOutputPath(job2, new Path(Util
s.WordNumClass));
        boolean result = job2.waitForCompletion(true);
        return (result ? 0 : 1);
    }
```

```
    public static void main(String[] args) throws Exceptio
n {
        int res = ToolRunner.run(new Configuration(),
                new CalcWordNumClass(), args);
        System.exit(res);
    }
}
```

4 TestPreparation.java 文件

```
package com.Bayes;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.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.FileInputForm
at;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFo
rmat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import java.io.IOException;
public class TestPreparation extends Configured implements
 Tool {
    public static class TestPreparation_Mapper extends Map
per<LongWritable, Text, Text, Text> {
        private Text key_out = new Text();
        private Text value_out = new Text();
        @Override
```

```
        protected void map(LongWritable key, Text value, C
ontext context) throws IOException, InterruptedException {
            InputSplit inputsplit = context.getInputSplit(
);
            //获取类名
            String className = ((FileSplit)inputsplit).get
Path().getParent().getName();
            //获取文档名
            String fileName = ((FileSplit)inputsplit).getP
ath().getName();

            key_out.set(className + "\t" +fileName);
            value_out.set(value.toString());
            context.write(key_out, value_out);
        }
    }
    public static class TestPreparation_Reducer extends Re
ducer<Text, Text, Text, Text> {
        private Text result = new Text();
        private StringBuffer stringBuffer;
        @Override
        protected void reduce(Text key, Iterable<Text> val
ues, Context context) throws IOException, InterruptedExcep
tion {
            stringBuffer = new StringBuffer();
            for (Text value : values){
                stringBuffer = stringBuffer.append(value.t
oString() + " ");
            }
            result.set(stringBuffer.toString());
            context.write(key, result);
        }
    }
    @Override
    public int run(String[] strings) throws Exception {
```

```
        Configuration conf = getConf();
        Job job3 = Job.getInstance(conf, "TestPreparation"
);

        FileSystem fileSystem = FileSystem.get(conf);
        Path outputPath3 = new Path(Utils.Test_Preparation
);
        if(fileSystem.exists(outputPath3))
            fileSystem.delete(outputPath3, true);
        //设置 jar 加载路径
        job3.setJarByClass(TestPreparation.class);
        //设置 map 和 reduce 类
        job3.setMapperClass(TestPreparation_Mapper.class);
        job3.setCombinerClass(TestPreparation_Reducer.clas
s);
        job3.setReducerClass(TestPreparation_Reducer.class
);
        //设置 map reduce 输出格式
        job3.setMapOutputKeyClass(Text.class);
        job3.setMapOutputValueClass(Text.class);
        job3.setOutputKeyClass(Text.class);
        job3.setOutputValueClass(Text.class);
        //设置输入和输出路径
        FileInputFormat.setInputDirRecursive(job3,true);
        FileInputFormat.addInputPath(job3, new Path(Utils.
TEST_DATA_PATH));
//        FileInputFormat.setInputPaths(job2, new Path(Uti
ls.TRAIN_DATA_PATH));
        FileOutputFormat.setOutputPath(job3, new Path(Util
s.Test_Preparation));
        boolean result = job3.waitForCompletion(true);
        return (result ? 0 : 1);
    }
    public static void main(String[] args) throws Exceptio
n {
```

```
        int res = ToolRunner.run(new Configuration(),
                new TestPreparation(), args);
        System.exit(res);
    }
}
```

4.5TestPredication.java 文件

```
package com.Bayes;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
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.FileInputForm
at;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFo
rmat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
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 TestPrediction extends Configured implements
Tool {
    private static HashMap<String, Double> priorProbabilit
y = new HashMap<String, Double>(); // 类的先验概率
    private static HashMap<String, Double> conditionalProb
ability = new HashMap<>(); // 每个单词在类中的条件概率
    //计算类的先验概率
    public static void Get_PriorProbability() throws IOExc
eption {
        Configuration conf = new Configuration();
        FSDataInputStream fsr = null;
        BufferedReader bufferedReader = null;
        String lineValue = null;
        HashMap<String, Double> temp = new HashMap<>(); //
暂存类名和文档数
        double sum = 0;     //文档总数量
        try {
            FileSystem fs = FileSystem.get(URI.create(Util
s.DocNumClass + "part-r-00000"), conf);
            fsr = fs.open(new Path( Utils.DocNumClass + "/
part-r-00000"));
            bufferedReader = new BufferedReader(new InputS
treamReader(fsr)); //文档读入流
            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 StringToke
nizer(lineValue);
                String className = tokenizer.nextToken();
//类名
                String num_C_Tmp = tokenizer.nextToken();
//文档数量
                double numC = Double.parseDouble(num_C_Tmp
);
```

```
                temp.put(className, numC);
                sum = sum + numC; //文档总数量
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            bufferedReader.close();     //关闭资源
        }

        Iterator<Map.Entry<String, Double>> it = temp.entr
ySet().iterator();
        while (it.hasNext()){           //遍历计算先验概率
            Map.Entry val = (Map.Entry)it.next();
            String key = val.getKey().toString();
            double value = Double.parseDouble(val.getValue
().toString());
            value /= sum;
            priorProbability.put(key, value);
        }
    }
    public static void Get_ConditionProbability() throws I
OException {
        String filePath =Utils.WordNumClass + "/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(file
Path), conf);
            fsr = fs.open(new Path(filePath));
```

```
            bufferedReader = new BufferedReader(new InputS
treamReader(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 StringToke
nizer(lineValue);
                String className = tokenizer.nextToken();
                String word =tokenizer.nextToken();
                String numWordTmp = tokenizer.nextToken();
                double numWord = Double.parseDouble(numWor
dTmp);
                if(wordSum.containsKey(className))
                    wordSum.put(className,wordSum.get(clas
sName)+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(file
Path), conf);
```

```
            fsr = fs.open(new Path(filePath));
            bufferedReader = new BufferedReader(new InputS
treamReader(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 StringToke
nizer(lineValue);
                String className = tokenizer.nextToken();
                String word =tokenizer.nextToken();
                String numWordTmp = tokenizer.nextToken();
                double numWord = Double.parseDouble(numWor
dTmp);
                String key=className+"\t"+word;
                conditionalProbability.put(key,(numWord+1.
0)/wordSum.get(className));
                //System.out.println(className+"\t"+word+"
\t"+wordsProbably.get(key));
            }
            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
            conditionalProbability.put(key.toString(),1.0/
Double.parseDouble(entry.getValue().toString()));
        }
    }
    public static class Prediction_Mapper extends Mapper<L
ongWritable, Text, Text, Text> {
        public void setup(Context context)throws IOExcepti
on{
            Get_PriorProbability(); //先验概率
            Get_ConditionProbability(); //条件概率
        }
        private Text newKey = new Text();
        private Text newValue = new Text();
        @Override
        protected void map(LongWritable key, Text value, C
ontext context) throws IOException, InterruptedException {
            String[] lineValues = value.toString().split("
\\s");    //分词，按照空白字符切割
            String class_Name = lineValues[0];   //得到类
名
            String fileName = lineValues[1];    //得到文件
名
            for(Map.Entry<String, Double> entry : priorPro
bability.entrySet()){
                String className = entry.getKey();
                newKey.set(class_Name + "\t" + fileName);/
/新的键值的 key 为<类明  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<lineValues.length; i++){
                    String tempKey = className + "\t" + li
neValues[i];//构建临时键值对<class_word>,在 wordsProbably 表
中查找对应的概率
                    if(conditionalProbability.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(conditionalP
robability.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(conditionalP
robability.get(className));
                    }
                }
                newValue.set(className + "\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 Prediction_Reduce extends Reducer<
Text, Text, Text, Text> {
        Text newValue = new Text();
        public void reduce(Text key, Iterable<Text> values
, Context context) throws IOException, InterruptedExceptio
n{
            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.to
String());
                String[] result = value.toString().split("
\\s");
                String className=result[0];
                String probably=result[1];
                if(flag != true){//循环第一次
                    tempClass = className;//value.toString
().substring(0, index);
                    tempProbably = Double.parseDouble(prob
ably);
                    flag = true;
                }else{//否则当概率更大时就更新 tempClass 和
tempProbably
                    if(Double.parseDouble(probably) > temp
Probably){
                        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);
        }
    }
    @Override
    public int run(String[] strings) throws Exception {
        Configuration conf = getConf();
```

```
        FileSystem hdfs = FileSystem.get(conf);
        Path outputPath2 = new Path(Utils.Test_Prediction)
;
        if(hdfs.exists(outputPath2))
            hdfs.delete(outputPath2, true);
        Job job4 =Job.getInstance(conf, "Prediction");
        job4.setJarByClass(TestPrediction.class);
        job4.setMapperClass(Prediction_Mapper.class);
        job4.setCombinerClass(Prediction_Reduce.class);
        job4.setReducerClass(Prediction_Reduce.class);
        FileInputFormat.setInputDirRecursive(job4,true);
        job4.setOutputKeyClass(Text.class);//reduce 阶段的
输出的 key
        job4.setOutputValueClass(Text.class);//reduce 阶段
的输出的 value
        FileInputFormat.addInputPath(job4, new Path(Utils.
Test_Preparation));
        FileOutputFormat.setOutputPath(job4, new Path(Util
s.Test_Prediction));
        boolean result = job4.waitForCompletion(true);
        return (result ? 0 : 1);
    }
    public static void main(String[] args) throws Exceptio
n {
        int res = ToolRunner.run(new Configuration(),
                new TestPrediction(), args);
        System.exit(res);
    }
}
```

6 Evaluation.java 文件

```
package com.Bayes;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FSDataInputStream;
```

```
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URI;
import java.util.ArrayList;
public class Evaluation extends Configured implements Tool
 {
    public static void GetEvaluation(Configuration conf) t
hrows IOException {
        //读取 TextPrediction 输出的文档
        String classFilePath =Utils.Test_Prediction + "/pa
rt-r-00000";
        FileSystem fs = FileSystem.get(URI.create(classFil
ePath), conf);
        FSDataInputStream fsr = fs.open(new Path(classFile
Path));
        ArrayList<String> ClassNames = new ArrayList<>();
      //得到待分类的类名
        ArrayList<Integer> TruePositive = new ArrayList<>(
);    //TP,属于该类且被分到该类的数目
        ArrayList<Integer> FalseNegative = new ArrayList<>
();   //FN,属于该类但没分到该类的数目
        ArrayList<Integer> FalsePositive = new ArrayList<>
();   //FP,不属于该类但分到该类的数目
        ArrayList<Double> precision = new ArrayList<>();
      //Precision 精度:P = TP/(TP+FP)
        ArrayList<Double> recall = new ArrayList<>();
      //Recall 精度:   R = TP/(TP+FN)
        ArrayList<Double> F1 = new ArrayList<>();
      //P 和 R 的调和平均:F1 = 2PR/(P+R)
        BufferedReader reader = null;
```

```
        Integer temp = 0;       //下面用于计算时的暂时存储数
据
        try {
            reader = new BufferedReader(new InputStreamRea
der(fsr));
            String lineValue = null;
            while ((lineValue = reader.readLine()) != null
){
                //按站空白字符分词，分词后得到的数组中，前三项
依次为：真实类别名，文件名，预测类别名
                String[] values = lineValue.split("\\s");
                if (!ClassNames.contains(values[0])) {
                    ClassNames.add(values[0]);
                    TruePositive.add(0);
                    FalseNegative.add(0);
                    FalsePositive.add(0);
                }
                if (!ClassNames.contains(values[2])) {
                    ClassNames.add(values[2]);
                    TruePositive.add(0);
                    FalseNegative.add(0);
                    FalsePositive.add(0);
                }
                if (values[0].equals(values[2])){
                    temp = TruePositive.get(ClassNames.ind
exOf(values[2])) + 1;
                    TruePositive.set(ClassNames.indexOf(va
lues[2]), temp);
                }
                else {
                    temp = FalseNegative.get(ClassNames.in
dexOf(values[0])) + 1;
                    FalseNegative.set(ClassNames.indexOf(v
alues[0]), temp);
```

```
                    temp = FalsePositive.get(ClassNames.in
dexOf(values[2])) + 1;
                    FalsePositive.set(ClassNames.indexOf(v
alues[2]), temp);
                }
            }
            for (int i = 0; i < ClassNames.size(); i++) {
                int TP = TruePositive.get(i);
                int FP = FalsePositive.get(i);
                int FN = FalseNegative.get(i);
                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);
            }
            /*
             * 计算宏平均和微平均
             * 以计算 precision 为例
             * 宏平均的 precision：(p1+p2+...+pN)/N
             * 微平均的 precision：对应各项 PR 相加后再计算
precision
             * */
            double p_Sum_Ma = 0.0;
            double r_Sum_Ma = 0.0;
            double F1_Sum_Ma = 0.0;
            Integer TP_Sum_Mi = 0;
            Integer FN_Sum_Mi = 0;
            Integer FP_Sum_Mi = 0;
            int n = ClassNames.size();      //类的种类数量
            for (int i = 0; i < n; i++) {
                p_Sum_Ma += precision.get(i);
                r_Sum_Ma += recall.get(i);
                F1_Sum_Ma += F1.get(i);
```

```
                TP_Sum_Mi += TruePositive.get(i);
                FN_Sum_Mi += FalseNegative.get(i);
                FP_Sum_Mi += FalsePositive.get(i);
            }
            //宏平均
            double p_Ma = p_Sum_Ma / n;
            double r_Ma = r_Sum_Ma / n;
            double F1_Ma = F1_Sum_Ma / n;
            //微平均
            double p_Mi = TP_Sum_Mi * 1.0 / ( TP_Sum_Mi +
FP_Sum_Mi );;
            double r_Mi = TP_Sum_Mi * 1.0 / ( TP_Sum_Mi +
FN_Sum_Mi );
            double F1_Mi = 2 * p_Mi * r_Mi / ( p_Mi + r_Mi
 );
            for (int i = 0; i < n; i++) {
                System.out.println(ClassNames.get(i) + "\t
precision: " + precision.get(i).toString());
                System.out.println(ClassNames.get(i) + "\t
recall: " + recall.get(i).toString());
                System.out.println(ClassNames.get(i) + "\t
F1: " + F1.get(i).toString());
            }
            System.out.println("Macroaveraged(宏平
均) precision: "+ p_Ma );
            System.out.println("Macroaveraged(宏平
均) recall: "+ r_Ma );
            System.out.println("Macroaveraged(宏平
均) F1: "+ F1_Ma );
            System.out.println("Microaveraged(微平
均) precision: "+ p_Mi );
            System.out.println("Microaveraged(微平
均) recall: "+ r_Mi );
            System.out.println("Microaveraged(微平
均) F1: "+ F1_Mi );
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }finally {
            reader.close();
        }
    }
    @Override
    public int run(String[] strings) throws Exception {
        Configuration conf = getConf();
        GetEvaluation(conf);
        return 0;
    }
    public static void main(String[] args) throws Exceptio
n {
        int res = ToolRunner.run(new Configuration(),
                new Evaluation(), args);
        System.exit(res);
    }
}
```

五、数据集说明

本系统用到 4 个类别，分别是从 Country 文件夹下选取的 BRAZ、CZREP，和从 Industry 文件夹下选取的 I21000、I81502。将每个类中的文件随机选取 70%作为训练集，剩余的 30%作为测试集。具体的文档个数如下表所示：

用途 / 类别	训练集	测试集	总数
BRAZ	140	60	200
CZREP	89	38	127

I21000	85	36	121
I81502	90	38	128
总数	404	172	576

表 4-1：数据集类别与分布

六、程序运行说明

1. Job1

A．运行时 Web 页面监控图

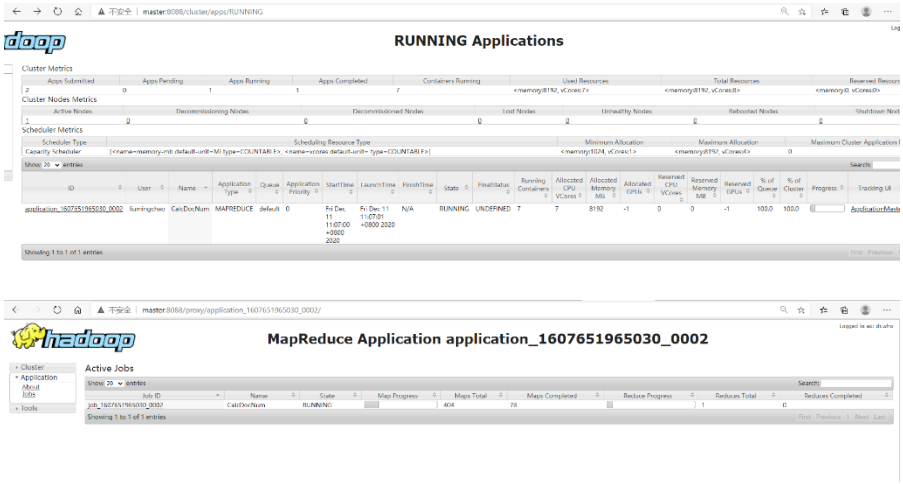

图 6.1.A:Job1 运行 web 监控图

B．运行截图

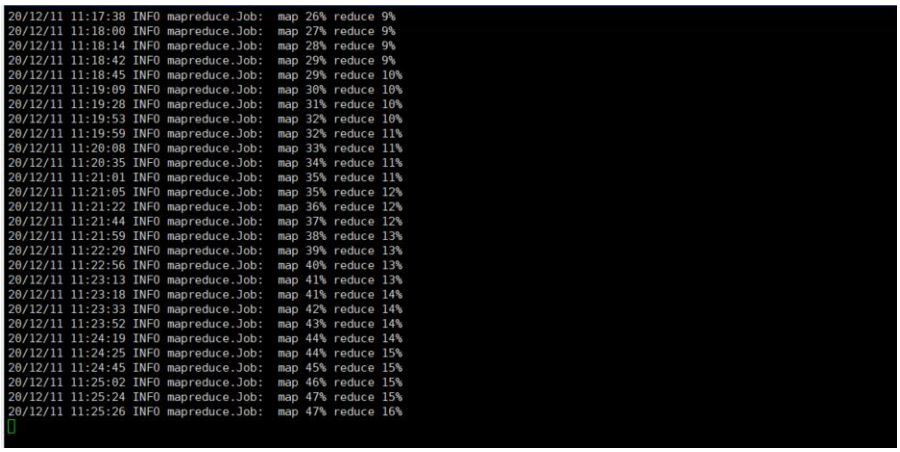

图 6.1.B:Job1 运行截图

C．map 和 reduce 数量截图

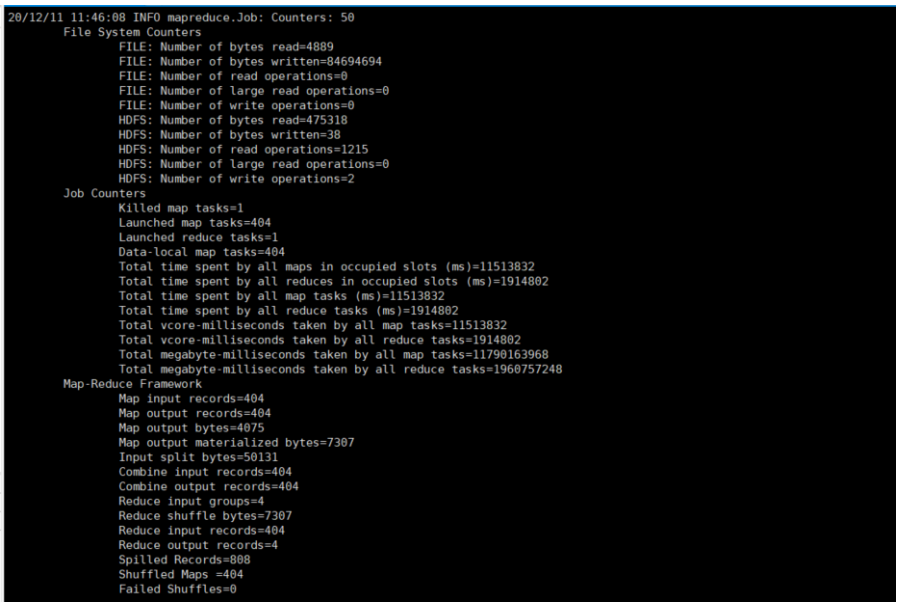

图 6.1.B:Job1 mapreduce 数量截图

由上图可知 Job1 共有 404 个 map 任务，1 个 reduce 任务。有 404 个 map 任务是因为训练集共有 404 个文件，每个文件大小都不超过 hadoop 的默认分片大小(128MB)，所以每个文件都不会分片，因此共有 404 个分片（每个文件为一个分片），每个分片对应一个 map 任务，共有 404 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 页面监控图

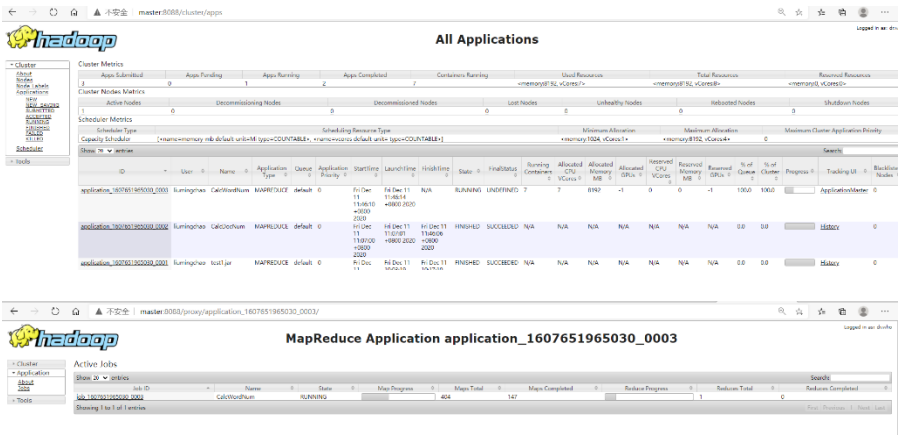

图 6.2.A:Job2 运行 web 监控图

B．运行截图

```
20/12/11 11:51:46 INFO mapreduce.Job:  map 13% reduce 0%
20/12/11 11:52:15 INFO mapreduce.Job:  map 14% reduce 0%
20/12/11 11:52:25 INFO mapreduce.Job:  map 15% reduce 0%
20/12/11 11:52:56 INFO mapreduce.Job:  map 16% reduce 0%
20/12/11 11:53:26 INFO mapreduce.Job:  map 17% reduce 0%
20/12/11 11:53:37 INFO mapreduce.Job:  map 18% reduce 0%
20/12/11 11:54:12 INFO mapreduce.Job:  map 19% reduce 0%
20/12/11 11:54:17 INFO mapreduce.Job:  map 19% reduce 6%
20/12/11 11:54:41 INFO mapreduce.Job:  map 20% reduce 6%
20/12/11 11:54:48 INFO mapreduce.Job:  map 20% reduce 7%
20/12/11 11:55:08 INFO mapreduce.Job:  map 21% reduce 7%
20/12/11 11:55:17 INFO mapreduce.Job:  map 22% reduce 7%
20/12/11 11:55:44 INFO mapreduce.Job:  map 23% reduce 7%
20/12/11 11:55:49 INFO mapreduce.Job:  map 23% reduce 8%
20/12/11 11:56:13 INFO mapreduce.Job:  map 24% reduce 8%
20/12/11 11:56:41 INFO mapreduce.Job:  map 25% reduce 8%
20/12/11 11:57:08 INFO mapreduce.Job:  map 26% reduce 8%
20/12/11 11:57:14 INFO mapreduce.Job:  map 26% reduce 9%
20/12/11 11:57:33 INFO mapreduce.Job:  map 27% reduce 9%
20/12/11 11:57:46 INFO mapreduce.Job:  map 28% reduce 9%
20/12/11 11:58:13 INFO mapreduce.Job:  map 29% reduce 9%
20/12/11 11:58:16 INFO mapreduce.Job:  map 29% reduce 10%
20/12/11 11:58:37 INFO mapreduce.Job:  map 30% reduce 10%
20/12/11 11:59:03 INFO mapreduce.Job:  map 31% reduce 10%
20/12/11 11:59:25 INFO mapreduce.Job:  map 32% reduce 10%
20/12/11 11:59:29 INFO mapreduce.Job:  map 32% reduce 11%
20/12/11 11:59:37 INFO mapreduce.Job:  map 33% reduce 11%
20/12/11 12:00:05 INFO mapreduce.Job:  map 34% reduce 11%
20/12/11 12:00:31 INFO mapreduce.Job:  map 35% reduce 11%
20/12/11 12:00:35 INFO mapreduce.Job:  map 35% reduce 12%
20/12/11 12:00:58 INFO mapreduce.Job:  map 36% reduce 12%
20/12/11 12:01:22 INFO mapreduce.Job:  map 37% reduce 12%
20/12/11 12:01:36 INFO mapreduce.Job:  map 38% reduce 12%
20/12/11 12:01:42 INFO mapreduce.Job:  map 38% reduce 13%
20/12/11 12:02:05 INFO mapreduce.Job:  map 39% reduce 13%
```

图 6.2.B:Job2 运行截图

C．map 和 reduce 数量截图

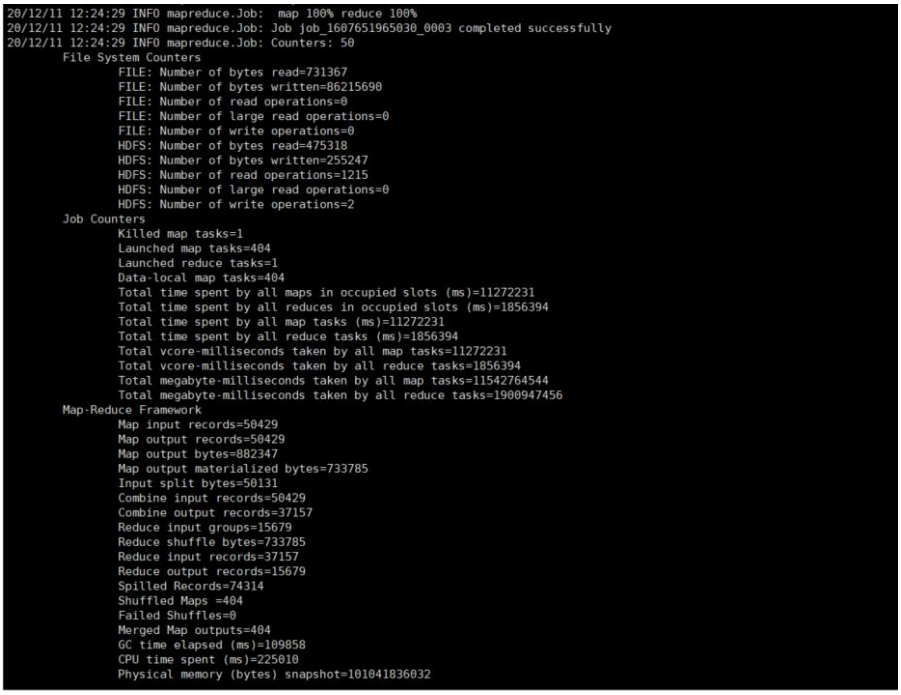

图 6.2.B:Job2 mapreduce 数量截图

由上图可知 Job1 共有 404 个 map 任务，1 个 reduce 任务。有 404 个 map 任务是因为训练集共有 404 个文件，每个文件大小都不超过 hadoop 的默认分片大小(128MB)，所以每个文件都不会分片，因此共有 404 个分片（每个文件为一个分片），每个分片对应一个 map 任务，共有 404 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 页面监控图

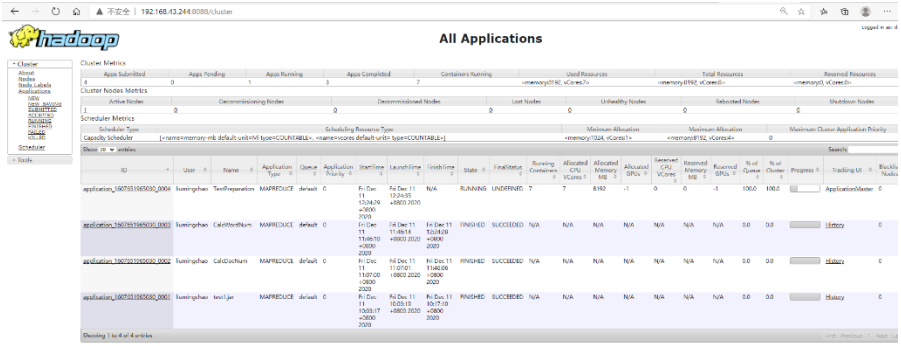

图 6.3.A:Job3 运行 web 监控图

B．运行截图

```
20/12/11 12:24:29 INFO client.RMProxy: Connecting to ResourceManager at master/192.168.43.244:8032
20/12/11 12:24:29 INFO input.FileInputFormat: Total input files to process : 172
20/12/11 12:24:29 INFO mapreduce.JobSubmitter: number of splits:172
20/12/11 12:24:29 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1607651965030_0004
20/12/11 12:24:29 INFO impl.YarnClientImpl: Submitted application application_1607651965030_0004
20/12/11 12:24:29 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/application_1607651965030_0004/
20/12/11 12:24:29 INFO mapreduce.Job: Running job: job_1607651965030_0004
20/12/11 12:24:44 INFO mapreduce.Job: Job job_1607651965030_0004 running in uber mode : false
20/12/11 12:24:44 INFO mapreduce.Job:  map 0% reduce 0%
20/12/11 12:25:18 INFO mapreduce.Job:  map 2% reduce 0%
20/12/11 12:25:19 INFO mapreduce.Job:  map 3% reduce 0%
20/12/11 12:25:49 INFO mapreduce.Job:  map 4% reduce 0%
20/12/11 12:25:53 INFO mapreduce.Job:  map 5% reduce 0%
20/12/11 12:25:54 INFO mapreduce.Job:  map 6% reduce 0%
20/12/11 12:25:55 INFO mapreduce.Job:  map 7% reduce 0%
20/12/11 12:26:23 INFO mapreduce.Job:  map 8% reduce 0%
20/12/11 12:26:29 INFO mapreduce.Job:  map 9% reduce 0%
20/12/11 12:26:30 INFO mapreduce.Job:  map 10% reduce 0%
20/12/11 12:26:57 INFO mapreduce.Job:  map 11% reduce 0%
20/12/11 12:27:03 INFO mapreduce.Job:  map 12% reduce 0%
20/12/11 12:27:05 INFO mapreduce.Job:  map 13% reduce 0%
20/12/11 12:27:07 INFO mapreduce.Job:  map 14% reduce 0%
20/12/11 12:27:33 INFO mapreduce.Job:  map 15% reduce 0%
20/12/11 12:27:41 INFO mapreduce.Job:  map 16% reduce 0%
20/12/11 12:27:42 INFO mapreduce.Job:  map 17% reduce 0%
20/12/11 12:28:09 INFO mapreduce.Job:  map 18% reduce 0%
20/12/11 12:28:15 INFO mapreduce.Job:  map 19% reduce 0%
20/12/11 12:28:18 INFO mapreduce.Job:  map 21% reduce 0%
20/12/11 12:28:47 INFO mapreduce.Job:  map 22% reduce 0%
20/12/11 12:28:50 INFO mapreduce.Job:  map 24% reduce 0%
20/12/11 12:28:52 INFO mapreduce.Job:  map 24% reduce 8%
20/12/11 12:29:18 INFO mapreduce.Job:  map 25% reduce 8%
20/12/11 12:29:19 INFO mapreduce.Job:  map 27% reduce 8%
20/12/11 12:29:23 INFO mapreduce.Job:  map 27% reduce 9%
20/12/11 12:29:47 INFO mapreduce.Job:  map 29% reduce 9%
20/12/11 12:29:48 INFO mapreduce.Job:  map 30% reduce 9%
20/12/11 12:29:54 INFO mapreduce.Job:  map 30% reduce 10%
20/12/11 12:30:15 INFO mapreduce.Job:  map 31% reduce 10%
20/12/11 12:30:16 INFO mapreduce.Job:  map 33% reduce 10%
20/12/11 12:30:18 INFO mapreduce.Job:  map 33% reduce 11%
20/12/11 12:30:44 INFO mapreduce.Job:  map 34% reduce 11%
20/12/11 12:30:46 INFO mapreduce.Job:  map 35% reduce 11%
20/12/11 12:30:49 INFO mapreduce.Job:  map 35% reduce 12%
20/12/11 12:31:08 INFO mapreduce.Job:  map 36% reduce 12%
20/12/11 12:31:13 INFO mapreduce.Job:  map 37% reduce 12%
20/12/11 12:31:16 INFO mapreduce.Job:  map 38% reduce 12%
20/12/11 12:31:19 INFO mapreduce.Job:  map 38% reduce 13%
20/12/11 12:31:37 INFO mapreduce.Job:  map 39% reduce 13%
20/12/11 12:31:41 INFO mapreduce.Job:  map 40% reduce 13%
```

图 6.3.B:Job3 运行截图

C．map 和 reduce 数量截图

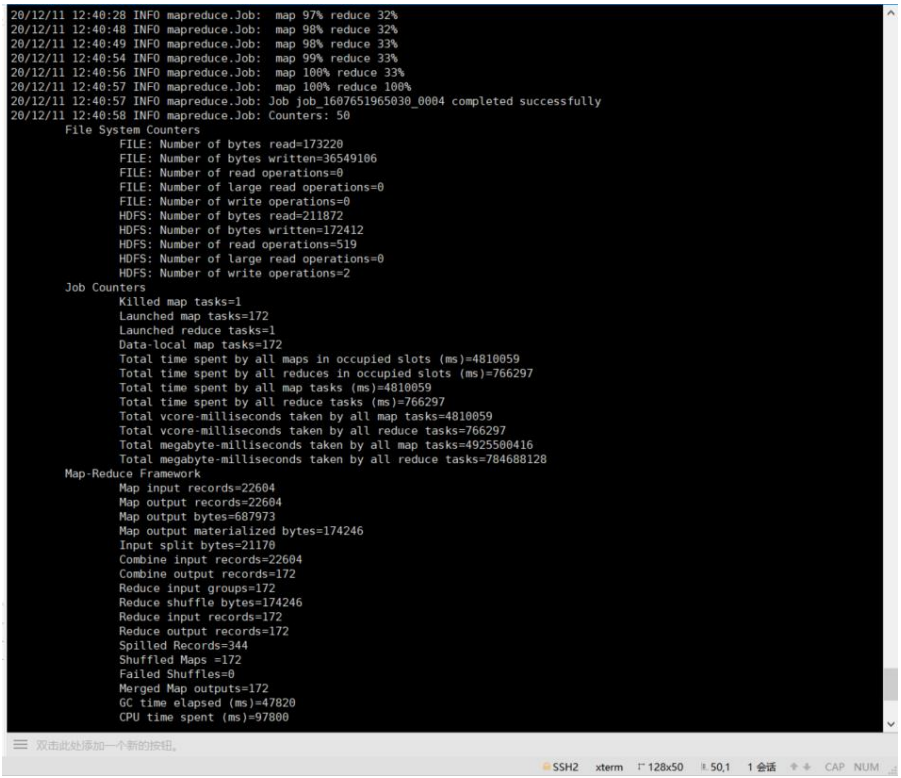

图 6.3.B:Job3 mapreduce 数量截图

由上图可知 Job3 共有 127 个 map 任务，1 个 reduce 任务。有 127 个 map 任务是因为测试集共有 127 个文件，每个文件大小都不超过 hadoop 的默认分片大小(128MB)，所以每个文件都不会分片，因此共有 127 个分片（每个文件为一个分片），每个分片对应一个 map 任务，共有 127 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 页面监控图

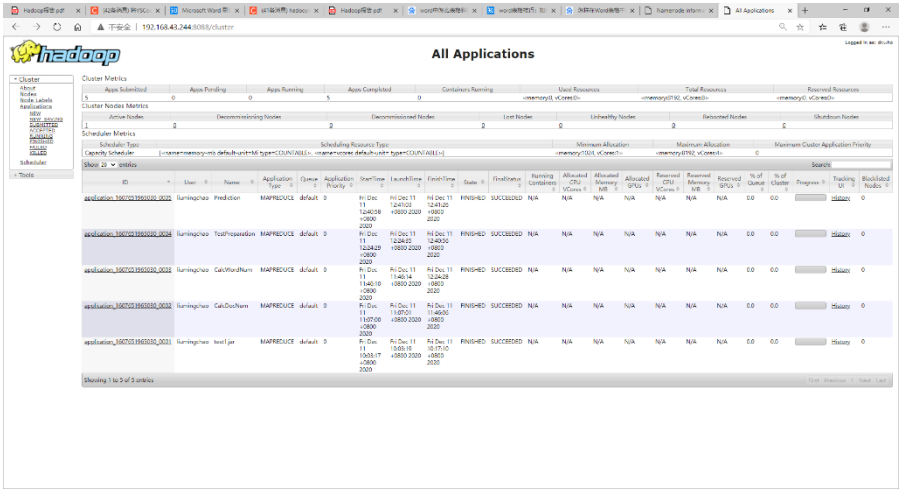

图 6.4.A:Job4 运行 web 监控图

B．运行截图

```
20/12/11 12:40:58 INFO client.RMProxy: Connecting to ResourceManager at master/192.168.43.244:8032
20/12/11 12:40:58 INFO input.FileInputFormat: Total input files to process : 1
20/12/11 12:40:58 INFO mapreduce.JobSubmitter: number of splits:1
20/12/11 12:40:58 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1607651965030_0005
20/12/11 12:40:58 INFO impl.YarnClientImpl: Submitted application application_1607651965030_0005
20/12/11 12:40:58 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/application_1607651965030_0005/
20/12/11 12:40:58 INFO mapreduce.Job: Running job: job_1607651965030_0005
20/12/11 12:41:12 INFO mapreduce.Job: Job job_1607651965030_0005 running in uber mode : false
20/12/11 12:41:12 INFO mapreduce.Job:  map 0% reduce 0%
20/12/11 12:41:20 INFO mapreduce.Job:  map 100% reduce 0%
20/12/11 12:41:27 INFO mapreduce.Job:  map 100% reduce 100%
```

图 6.4.B:Job4 运行截图

C．map 和 reduce 数量截图

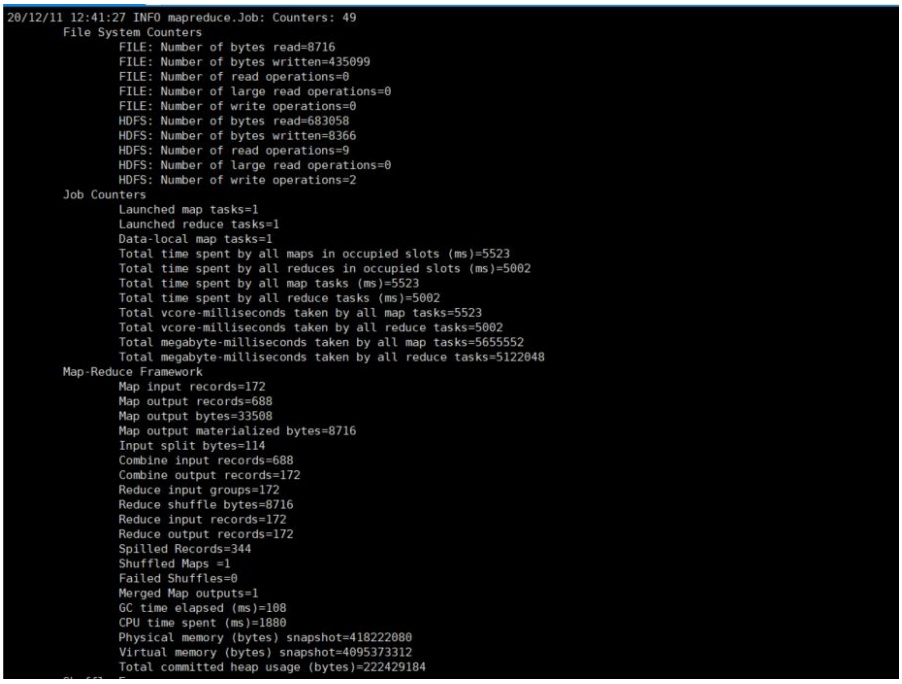

图 6.4.B:Job4 mapreduce 数量截图

由上图可知 Job3 共有 1 个 map 任务，1 个 reduce 任务。有 1 个 map 任务是 job4 的输入是读取 Job3 的输出文件，Job3 只有一个输出文件，该文件大小都不超过 hadoop 的默认分片大小(128MB)，所以不会分片，故该文件就是一个分片，一个分片对应一个 map 任务，共有 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 presicion，recall 和 F1 值。

运行 Evaluation.java 文件可以得到每个类别的 precision、recall、F1 值，以及宏平均和微平均下的 precision、recall、F1 值。以此来评估该分类器的可靠性，运行结果如下图所示：

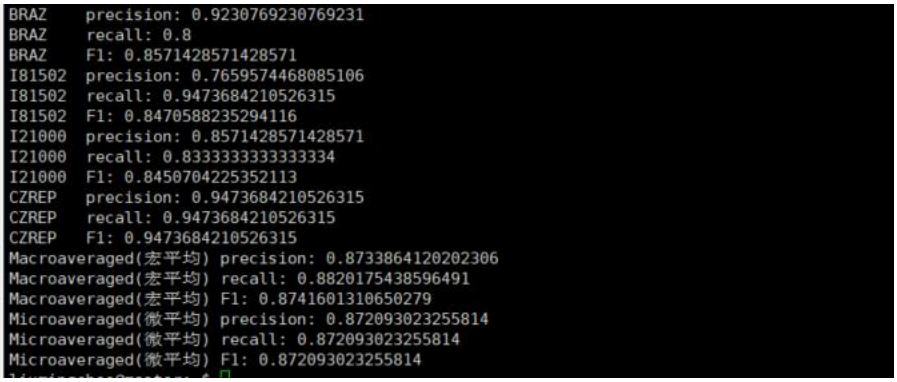

图 7.1：评估模型的 precision、recall、F1 数值图

下表中列出了相应的 precision、recall、F1 值：

标准 / 类别	Precision	Recall	F1
BRAZ	92.30%	80.00%	85.71%
CZREP	94.74%	94.74%	94.74%
I81502	76.60%	94.73%	84.71
I21000	85.71%	83.33%	84.51%
宏平均	87.34%	88.20%	87.42%
微平均	87.21%	87.20%	87.21%

表 7.1：模型评估数值图

由表中数据可以看出，朴素贝叶斯分类器虽然做了很强的先验假设简化了模型，大大减少了需要估计的参数个数，但是最终模型对于文本分类效果还是很好的。同时该方法也有提升空间，未来可以通过词项归一化、去掉停用词等操作来进一步提高贝叶斯分类器的效果。