目录

1. 贝叶斯分类器理论介绍

在机器学习中，朴素贝叶斯分类器是一系列以假设特征之间强（朴素）独立下运用贝叶斯定理为基础的简单概率分类器。

朴素贝叶斯自 20 世纪 50 年代已广泛研究。在 20 世纪 60 年代初就以另外一个名称引入到文本信息检索界中，并仍然是文本分类的一种热门（基准）方法，文本分类是以词频为特征判断文件所属类别或其他（如垃圾邮件、合法性、体育或政治等等）的问题。通过适当的预处理，它可以与这个领域更先进的方法（包括支持向量机）相竞争。它在自动医疗诊断中也有应用。

理论上，概率模型分类器是一个条件概率模型。

（1）朴素贝叶斯分类器的条件概率为：

$$\hat{P}(t|c)=\frac{T_{\mathrm{ct}}+1}{\sum_{t'\in V}T_{ct'}+B}$$

其中：

t ：表示测试集文件中的单词

c ：表示训练集中的类

T_{ct}：表示测试集文件中的单词 t 在训练集中类 c 里出现的总次数

$\sum_{t'\in V}T_{ct'}$：表示训练集中类 c 里的单词总数

B ：整个训练集文档 V 中的不同单词总数

条件概率模型做了平滑处理，防止测试集文件中的单词 t 在训练集中类 c 里没有出现的情况。

（2）朴素贝叶斯分类器的先验概率为：

$$\hat{P}(c)=\frac{\sum_{t'\in V}T_{ct'}}{\sum_{t'\in V}T_{\mathrm{vt'}}}$$

$\sum_{t'\in V}T_{ct'}$：表示训练集中类 c 里的单词总数

$\sum_{t'\in V}T_{\mathrm{vt'}}$：表示训练集文档 V 中的单词总数

这是基于单词的先验概率，是在基于文件的先验概率上的优化，可以减少整个训练集文档中有很多类别，但类别中只有很少的文件的情况的影响。

（3）根据朴素贝叶斯分类器的独立性假设的前提，可得到如下分类规则：

$$C_{\mathrm{map}}=\arg\max_{c\in C}[\log\hat{P}(c)+\sum_{1\le k\le n_d}\log\hat{P}(t|c)]$$

对训练集中类的集合分别计算上述数值，取最大的值的那个类就是预测的结果。

2. 贝叶斯分类器的 MapReduce 算法设计

整个贝叶斯分类器的训练和测试过程均是使用 MapReduce 来实现的。下图为贝叶斯分类器结构流程图。

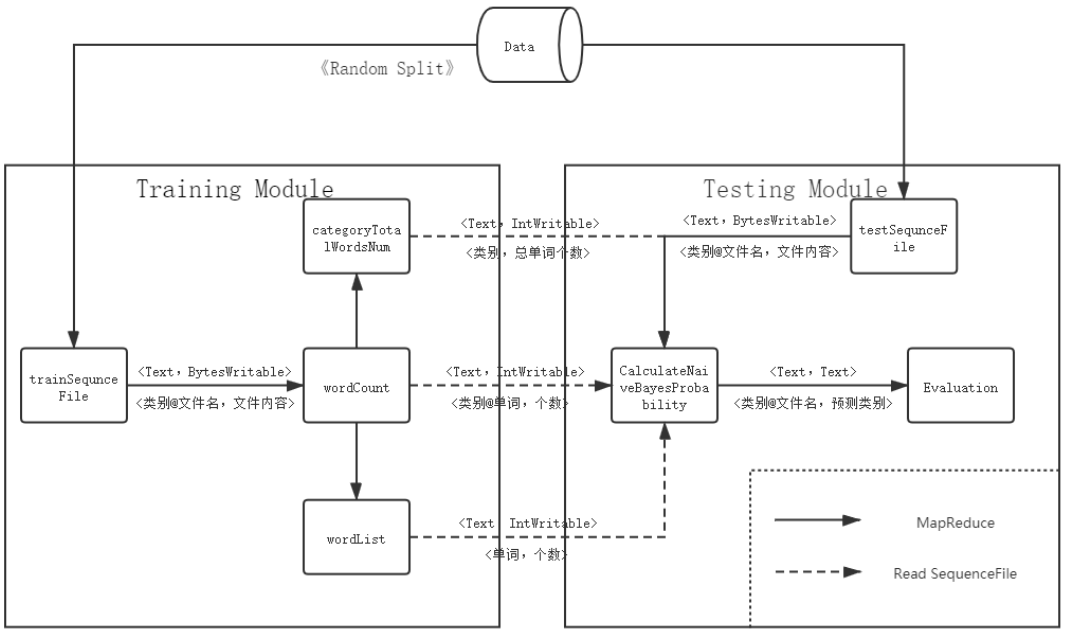


图 2.1 贝叶斯分类器结构流程图

贝叶斯分类器主要分为训练和测试两个大的模块。首先将数据按照一定比例随机划分为训练集和测试集。在训练阶段主要是进行一些信息的统计，以便使用贝叶斯公式计算概率。主要统计的信息为每个类别每个单词出现的次数（wordCount）、每个类别的总单词个数（categoryTotalWordsNum）以及总单词类别数（wordList）。在测试阶段则是读入训练阶段的信息预测是每个类别的概率，选取概率最大的作为预测类别，最后评估模型计算 Precision、Recall 和 F1 三个指标。

2.1 训练阶段

训练阶段主要包括 4 个 MapReduce 程序，分别是序列化文件（trainSequenceFile）、统计每个类别每个单词的个数（wordCount）、统计每个类别的总单词个数（categoryTotalWordsNum）和统计所有单词的种类个数（wordList）。具体的 MapReduce 设计如下所示：

- 序列化文件（SequenceFile）

(input)<NullWritable, BytesWritable> → **map** → <Text, BytesWritable>**(output)**
(input)<null, 文件内容> → **map** → <类别@文件名, 文件内容>**(output)**

上述第一行输入输出用的是类型表示，第二行用的具体的内容表示。“类别@文件名”中

的“@”是分隔符。在 SequenceFile 中只用了 map 将一行的文件内容转为 key-value 对即可。

- wordCount

(**input**)<Text, BytesWritable> → **map** → <Text, BytesWritable> → **combine** → <Text, IntWritable> → **reduce** → <Text, IntWritable>(**output**)
(**input**)<类别@文件名，文件内容> → **map** → <类别@单词名，1> → **combine** → <类别@单词，个数> → **reduce** → <类别@单词，个数>(**output**)

这就是官网的 wordCount 例子，只需要将 key 变为“类别@单词”。因为统计的是每个类别每个单词出现的次数。

- categoryTotalWordsNum

(**input**)<Text, IntWritable> → **map** → <Text, IntWritable> → **combine** → <Text, IntWritable> → **reduce** → <Text, IntWritable>(**output**)
(**input**)<类别@单词，个数> → **map** → <类别，单词个数> → **combine** → <类别，总单词个数> → **reduce** → <类别，总单词个数>(**output**)

统计每个类别的总单词个数。

- wordlist

(**input**)<Text, IntWritable> → **map** → <Text, IntWritable> → **combine** → <Text, IntWritable> → **reduce** → <Text, IntWritable>(**output**)
(**input**)<类别@单词，个数> → **map** → <单词，个数> → **combine** → <单词，总个数> → **reduce** → <单词，总个数>(**output**)

2.2 测试阶段

测试阶段主要由 3 个 MapReduce 程序组成，分别是序列化文件（testSequenceFile）、计算概率并预测类别（CalculateNaiveBayesProbability）和评估三个指标（Evaluation）。

- testSequenceFile

这个的过程和训练阶段的一致。

- CalculateNaiveBayesProbability

(**input**)<Text, BytesWritable> → **map** → <Text, DoubleWritable> → **reduce** → <Text, Text>(**output**)
(**input**)<类别@文件名，文件内容> → **map** → <类别@文件名@预测类别，概率> → **reduce** → <类别@文件名，预测类别>(**output**)

这个程序在 map 阶段会读入训练阶段统计的三个信息，如上图 2.1 所示。然后计算每个文件预测为每个类的概率，生成“类别@文件名@预测类别，概率”这样的键值对交给 reduce。最后 reduce 取最大的概率作为最终的预测类别，输出“类别@文件名，预测类别”这样的键值对。

- Evaluation

(**input**)<Text, Text> → **map** → <Text, DoubleWritable> → **combine** → <Text, DoubleWritable> → **reduce** → <Text, DoubleWritable>(**output**)

(input)<类别@文件名，预测类别> → **map** → <指标，一个类别的分数> → **combine** → <指标，所有类别的平均分数> → **reduce** → <指标，所有类别的平均分数>**(output)**

计算 Precision、Recall、F1 三个指标

3. 源代码清单

Main.java

```java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.util.ToolRunner;

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

/**
 * @className: Main
 * @description:  整个朴素贝叶斯算法的启动类
 * @author: dahongdou
 * @date: 2020/10/22
 **/
public class Main {
    /**
     * trainFiles  训练文件集合
     * testFiles  测试文件集合
     */
    private static List<String> trainFiles = new ArrayList<String>(), testFiles = new
ArrayList<String>();
    /**
     * proportion  训练集与测试集比例
     */
    private static double proportion = 0.8;
    /**
     * outputPath  输出路径（由输入参数的最后一个给出）
     */
    private static String outputPath;

    private static void randomSplitFiles(FileSystem fs, Path dirPath) {
        try {
            //当前目录下总的文件集合
            FileStatus[] status = fs.listStatus(dirPath);

            //训练集个数
            int trainFilesNum = (int)(status.length * proportion);

            //随机获得训练集下标
            Set<Integer> trainFilesIndex = new HashSet<Integer>();
            Random random = new Random();
            while(trainFilesIndex.size() < trainFilesNum) {
                trainFilesIndex.add(random.nextInt(status.length));
            }

            //分别将文件路径加入训练集和测试集
            for(int i=0; i<status.length; i++){
                if(trainFilesIndex.contains(i)){
                    trainFiles.add(status[i].getPath().toString());
                }else{
                    testFiles.add(status[i].getPath().toString());
                }
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     *  合并两个路径，没完成，先这样吧。。。
     * @param path1
     * @param path2
     * @return
     */
    private static String mergePath(String path1, String path2) {
        return path1 + path2;
//          return Path.mergePaths(new Path(path1), new Path(path2)).toString();
    }

    public static void main(String[] args) throws Exception {
        if (args.length == 0) {
            System.out.println("请输入类别");
            return;
        }
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(conf);

        //将所有的类别按照 proportion 比例将文件随机划分
        for(int i=0; i<args.length-1; i++){
            randomSplitFiles(fs, new Path(args[i]));
        }

        //设置输出路径为 args 的最后一个参数
        outputPath = args[args.length-1];

        //将小文件打包成 sequenceFile
        trainFiles.add(mergePath(outputPath, "trainSequenceFile"));
        ToolRunner.run(conf, new MySequenceFile(), trainFiles.toArray(new String[0]));

        testFiles.add(mergePath(outputPath, "testSequenceFile"));
        ToolRunner.run(conf, new MySequenceFile(), testFiles.toArray(new String[0]));

        //统计每个类别每个单词出现的次数
        ToolRunner.run(conf, new WordCount(), new String[]{mergePath(outputPath,
"trainSequenceFile"), mergePath(outputPath, "wordCount")});

        //统计每个类别的单词总数
        ToolRunner.run(conf, new CountTotalWordsOfCategory(), new
String[]{mergePath(outputPath, "wordCount"), mergePath(outputPath, "categoryTotalWordsNum")});

        //统计单词列表
        ToolRunner.run(conf, new WordList(), new String[]{mergePath(outputPath, "wordCount"),
mergePath(outputPath, "wordList")});

        conf.set("wordCount", mergePath(outputPath, "wordCount"));
        conf.set("wordList", mergePath(outputPath, "wordList"));
        conf.set("categoryTotalWordsNum", mergePath(outputPath, "categoryTotalWordsNum"));

        //预测分类结果
        ToolRunner.run(conf, new CalculateNaiveBayesProbability(), new
String[]{mergePath(outputPath, "testSequenceFile"), mergePath(outputPath, "probability")});

        //评估
        ToolRunner.run(conf, new Evaluation(), new String[]{mergePath(outputPath, "probability"),
mergePath(outputPath, "evaluation")});
    }
}
```

MyFileInputFormat.java

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

import java.io.IOException;

/**
  * @className: MyFileInputFormat
  * @description: 自定义文件输入格式，主要是控制生成的 key-value 对。
  * @author: dahongdou
  * @date: 2020/10/21
  **/
public class MyFileInputFormat extends FileInputFormat<NullWritable, BytesWritable> {
    @Override
    public RecordReader<NullWritable, BytesWritable> createRecordReader(
                InputSplit inputSplit, TaskAttemptContext taskAttemptContext)
                throws IOException, InterruptedException {
            MyFileRecordReader reader = new MyFileRecordReader();
            reader.initialize(inputSplit, taskAttemptContext);
            return reader;
    }

    @Override
    protected boolean isSplitable(JobContext context, Path file) {return false; }
}

/**
  * @className: MyFileRecordReader
  * @description: 定义生成的 value 对为文件内容。
  * @author: dahongdou
  * @date: 2020/10/21
```

```
 **/
class MyFileRecordReader extends
        RecordReader<NullWritable, BytesWritable> {

    private FileSplit fileSplit;
    private Configuration conf;
    private BytesWritable value = new BytesWritable();
    private boolean processed = false;

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

    @Override
    public boolean nextKeyValue() throws IOException, InterruptedException {
        if (!processed) {
            byte[] contents = new byte[(int) fileSplit.getLength()];
            Path file = fileSplit.getPath();
            FileSystem fs = file.getFileSystem(conf);
            FSDataInputStream in = null;
            try {
                in = fs.open(file);
                IOUtils.readFully(in, contents, 0, contents.length);
                value.set(contents, 0, contents.length);
            } finally {
                IOUtils.closeStream(in);
            }
            processed = true;
            return true;
        }
        return false;
    }

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

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

```java
    }

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

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

MySequenceFile.java

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

import java.io.IOException;

/**
  * @className: MySequenceFile
  * @description:  将小文件打包成 sequenceFile。
  * @author: dahongdou
  * @date: 2020/10/21
  **/
public class MySequenceFile extends Configured implements Tool {

    static class FileMapper extends Mapper<NullWritable, BytesWritable, Text, BytesWritable> {
        /**
```

```
         *  写入 sequenceFile 的小文件的 key 值，形式为“类别@文件名”
         */
        private Text filenameKey;

        @Override
        protected void setup(Context context)
                throws IOException, InterruptedException {
            InputSplit split = context.getInputSplit();
            Path path = ((FileSplit) split).getPath();
            String filename = path.getName();
            String classname = path.getParent().getName();
            filenameKey = new Text(classname + "@" + filename);
            //System.out.println(classname + "@" + filename);
        }

        @Override
        protected void map(NullWritable key, BytesWritable value, Context context) throws
IOException, InterruptedException {
            context.write(filenameKey, value);
        }
    }

    public int run(String[] args) throws Exception {
        Configuration conf = this.getConf();
        Job job = Job.getInstance(conf, "SequenceFile");

        job.setJarByClass(MySequenceFile.class);
        job.setMapperClass(FileMapper.class);

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

        //最后一个参数为输出路径
        for(int i=0; i<args.length-1; i++){
            FileInputFormat.addInputPath(job, new Path(args[i]));
        }
        FileOutputFormat.setOutputPath(job, new Path(args[args.length-1]));

        return job.waitForCompletion(true) ? 0 : 1;
    }

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

WordCount.java

```

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

import java.io.IOException;
import java.util.StringTokenizer;

/**
  * @className: WordCount
  * @description:  统计每个类每个单词出现的次数
  * @author: dahongdou
  * @date: 2020/10/20
  **/
public class WordCount extends Configured implements Tool {

        public static class TokenizerMapper extends Mapper<Text, BytesWritable, Text, IntWritable> {
                private final static IntWritable one = new IntWritable(1);
                private Text word = new Text();

                @Override
                public void map(Text key, BytesWritable value, Context context) throws IOException,
InterruptedException {
                        String classname = key.toString().split("@")[0];
                        String content = new String(value.getBytes(),0,value.getLength());
```

```
            StringTokenizer itr = new StringTokenizer(content);
            while (itr.hasMoreTokens()) {
                word.set(classname + "@" + itr.nextToken());
                context.write(word, one);
            }
        }
    }

    public static class IntSumReducer 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);
        }
    }

    public int run(String[] args) throws Exception {
        Configuration conf = this.getConf();
        Job job = Job.getInstance(conf, "word count");

        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);

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


        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
```

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

CountTotalWordsOfCategory.java

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

import java.io.IOException;

/**
  * @className: CountTotalWordsOfCategory
  * @description:  统计每个类别的总单词个数
  * @author: dahongdou
  * @date: 2020/10/21
  **/
public class CountTotalWordsOfCategory extends Configured implements Tool {
        public static class CountTotalWordsOfCategoryMapper
                    extends Mapper<Text, IntWritable, Text, IntWritable> {
            @Override
            protected void map(Text key, IntWritable value, Context context) throws IOException,
InterruptedException {
                    String category = key.toString().split("@")[0];
                    context.write(new Text(category), value);
            }
        }

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

    public int run(String[] args) throws Exception {
        Configuration conf = this.getConf();
        Job job = Job.getInstance(conf, "Count the total number of category words");

        job.setJarByClass(CountTotalWordsOfCategory.class);
        job.setMapperClass(CountTotalWordsOfCategoryMapper.class);
        job.setCombinerClass(CountTotalWordsOfCategoryReducer.class);
        job.setReducerClass(CountTotalWordsOfCategoryReducer.class);

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

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        if (args.length == 0) {
            args = new String[]{"output/wordcount", "output/totalWordsOfCategory"};
        }
        int res = ToolRunner.run(new Configuration(), new CountTotalWordsOfCategory(), args);
        System.exit(res);
    }
}
```

WordList.java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
```

```java
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.SequenceFileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

import java.io.IOException;

/**
 * @className: WordList
 * @description: 获得所有类别的单词列表
 * @author: dahongdou
 * @date: 2020/10/21
 **/
public class WordList extends Configured implements Tool {

    public static class WordListMapper extends Mapper<Text, IntWritable, Text, IntWritable> {
        @Override
        protected void map(Text key, IntWritable value, Context context) throws IOException,
InterruptedException {
            String word = key.toString().split("@")[1];
            context.write(new Text(word), value);
        }
    }
    public static class WordListReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        @Override
        protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
            int sum = 0;
            for(IntWritable val : values) {
                sum += val.get();
            }
            context.write(key, new IntWritable(sum));
        }
    }
    public int run(String[] args) throws Exception {
        Configuration conf = this.getConf();
        Job job = Job.getInstance(conf, "Statistics word list");

        job.setJarByClass(WordList.class);
        job.setMapperClass(WordListMapper.class);
        job.setCombinerClass(WordListReducer.class);
        job.setReducerClass(WordListReducer.class);

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

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        return job.waitForCompletion(true) ? 0 : 1;
    }
    public static void main(String[] args) throws Exception {
        if (args.length == 0) {
            args = new String[]{"output/wordCount", "output/wordList"};
        }
        int res = ToolRunner.run(new Configuration(), new WordList(), args);
        System.exit(res);
    }
}
```

CalculateNaiveBayesProbability.java

```java
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.*;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.SequenceFileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import java.io.IOException;
import java.util.*;

/**
  * @className: CalculateNaiveBayesProbability
  * @description:  计算每个类别每个单词的朴素贝叶斯概率
  * @author: dahongdou
  * @date: 2020/10/21
  **/
public class CalculateNaiveBayesProbability extends Configured implements Tool {

      public static class CalculateNaiveBayesProbabilityMapper extends Mapper<Text, BytesWritable,
Text, DoubleWritable> {
            /**
              * wordSet  所有单词集合
              */
            private Set<String> wordSet;
            /**
              * categoryTotalWordsNum  类别的所有单词数目
              * key(String)  类别
              * value(Integer)  单词数目
              */
            private Map<String, Integer> categoryTotalWordsNum;
            /**
              * categorySet  类别集合
              */
            private Set<String> categorySet;
            /**
              * categoryWordProbability  每个类别中每个单词的概率
              * key(String)  类别@单词
              * value(Double)  概率
              *
              *  计算公式:
              */
            private Map<String, Double> categoryWordProbability;
            /**
              * B  总单词类别个数（不重复）
              */
            private int B;
            /**
              * S  总单词个数
              */
            private int S;
            private Map<String, Integer> getMapFromFile(Configuration conf, Path path) throws
```

```
IOException {
			SequenceFile.Reader reader = new SequenceFile.Reader(conf,
SequenceFile.Reader.file(path));

			Text key = new Text();
			IntWritable value = new IntWritable();
			Map<String, Integer> map = new HashMap<String, Integer>();
			while(reader.next(key, value)) {
				map.put(key.toString(), value.get());
			}
			reader.close();
			return map;
		}

		@Override
		protected void setup(Context context) throws IOException, InterruptedException {
			Configuration conf = context.getConfiguration();
			Map<String, Integer> wordCount = getMapFromFile(context.getConfiguration(), new
Path(conf.get("wordCount")+"/part-r-00000"));
			categoryTotalWordsNum = getMapFromFile(context.getConfiguration(), new
Path(conf.get("categoryTotalWordsNum")+"/part-r-00000"));
			categorySet = categoryTotalWordsNum.keySet();
			wordSet = getMapFromFile(context.getConfiguration(), new
Path(conf.get("wordList")+"/part-r-00000")).keySet();
			B = wordSet.size();
			for(Integer num : categoryTotalWordsNum.values()){
				S += num;
			}

			//根据朴素贝叶斯公式计算每个单词的条件概率，以便后面直接使用
			categoryWordProbability = new HashMap<String, Double>();
			for(Map.Entry<String, Integer> entry : wordCount.entrySet()) {
				String category = entry.getKey().split("@")[0];
				Double property = Math.log10((entry.getValue() +
1.0)/(categoryTotalWordsNum.get(category) + B*1.0));

				//乘以先验概率，取 log 所以是加法运算
				property += Math.log10(categoryTotalWordsNum.get(category) * 1.0) / (S * 1.0);
				categoryWordProbability.put(entry.getKey(), property);
			}
		}

		@Override
		protected void map(Text key, BytesWritable value, Context context) throws IOException,
InterruptedException {
            String content = new String(value.getBytes(), 0, value.getLength());
            StringTokenizer itr = new StringTokenizer(content);
            //String category = key.toString().split("@")[0];
            while (itr.hasMoreTokens()) {
                String word = itr.nextToken();

                //forecastCategory 预测为哪个类别，每个类别都需要预测得到一个概率。
                for(String forecastCategory : categorySet) {
                    Double probability = 0.0;

                    //用 forecastCategory 和 word 组成 key，判断训练集这个预测类别是否存
在这个单词。是，则直接取概率；否，则设定次数为 1.
                    String forecastCategoryWordKey = forecastCategory + "@" + word;
                    if(categoryWordProbability.containsKey(forecastCategoryWordKey)) {
                        probability = categoryWordProbability.get(forecastCategoryWordKey);
                    }else {
                        probability =
Math.log10(1.0/(categoryTotalWordsNum.get(forecastCategory) + B*1.0));
                        probability +=
Math.log10(categoryTotalWordsNum.get(forecastCategory) * 1.0) / (S * 1.0);
                    }
                    context.write(new Text(key.toString() + "@" + forecastCategory), new
DoubleWritable(probability));
                }
            }
        }
    }

    public static class CalculateNaiveBayesProbabilityReducer extends Reducer<Text,
DoubleWritable, Text, Text> {
        /**
         * forecastResult  预测的结果
         * key(String)  类别@文件
         * value(String)  预测的类别
         */
        private Map<String, String> forecastResult;
        /**
         * forecastMaxNum  预测的最大值（用来控制预测结果，预测的最大值的类别为预测
类别）
         * key(String)  类别@文件
         * value(String)  预测概率
         */
        private Map<String, Double> forecastMaxNum;
        @Override
        protected void setup(Context context) throws IOException, InterruptedException {
            forecastResult = new HashMap<String, String>();
            forecastMaxNum = new HashMap<String, Double>();
        }

        @Override
        protected void reduce(Text key, Iterable<DoubleWritable> values, Context context) throws
IOException, InterruptedException {
            Double probability = 0.0;
            for(DoubleWritable val : values) {
                probability += val.get();
            }

            String categoryFilename = key.toString().split("@")[0] + "@" +
key.toString().split("@")[1];
            if(!forecastMaxNum.containsKey(categoryFilename)) {
                forecastMaxNum.put(categoryFilename, probability);
                forecastResult.put(categoryFilename, key.toString().split("@")[2]);
            }else if(forecastMaxNum.get(categoryFilename) < probability) {
                forecastMaxNum.put(categoryFilename, probability);
                forecastResult.put(categoryFilename, key.toString().split("@")[2]);
            }
//            context.write(key, new DoubleWritable(probability));
        }

        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException {
            for(Map.Entry<String, String> entry : forecastResult.entrySet()) {
                context.write(new Text(entry.getKey()), new Text(entry.getValue()));
            }
        }
    }

    public int run(String[] args) throws Exception {
        Configuration conf = this.getConf();
        Job job = Job.getInstance(conf, "Calculate NaiveBayes Probability");

        job.setJarByClass(CalculateNaiveBayesProbability.class);
        job.setMapperClass(CalculateNaiveBayesProbabilityMapper.class);
//        job.setCombinerClass(CalculateNaiveBayesProbabilityReducer.class);
        job.setReducerClass(CalculateNaiveBayesProbabilityReducer.class);
```

```java
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(DoubleWritable.class);
        job.setInputFormatClass(SequenceFileInputFormat.class);
        job.setOutputFormatClass(SequenceFileOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        return job.waitForCompletion(true) ? 0 : 1;
    }

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

Evaluation.java

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

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

/**
 * @className: Evaluation
 * @description:  评估分类效果，三个指标 precision, recall, F1
 * @author: dahongdou
 * @date: 2020/10/23
 **/
public class Evaluation extends Configured implements Tool {
    public static class EvaluationMapper extends Mapper<Text, Text, Text, DoubleWritable> {
        /**
         * TP（实际为正样本预测为正样本）、FN（实际为正样本预测为负样本）、FP（实
际为负样本预测为正样本）
         * key（String） 类别
         * value（Integer） 个数
         */
        private Map<String, Integer> TP, FN, FP;

        @Override
        protected void setup(Context context) throws IOException, InterruptedException {
            TP = new HashMap<String, Integer>();
            FN = new HashMap<String, Integer>();
            FP = new HashMap<String, Integer>();
        }

        @Override
        protected void map(Text key, Text value, Context context) throws IOException,
InterruptedException {
            //真实类别和预测类别
            String category = key.toString().split("@")[0];
            String forecastCategory = value.toString();

            //设 a 为真实类别，b 为预测类别。
            //若 a==b，TP[a]++;
            //若 a!=b，FN[a]++、FP[b]++;
            if(category.equals(forecastCategory)) {
                try {
                    TP.put(category, TP.get(category) + 1);
                }catch (Exception e) {
                    TP.put(category, 1);
                }
            }else {
                try {
                    FN.put(category, FN.get(category) + 1);
                }catch (Exception e) {
                    FN.put(category, 1);
                }
                try {
                    FP.put(forecastCategory, FP.get(forecastCategory) + 1);
                }catch (Exception e) {
                    FP.put(forecastCategory, 1);
                }
            }
        }

        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException {
            for(String key : TP.keySet()) {
                double tp = TP.get(key);
                double fn = FN.containsKey(key) ? FN.get(key) : 0;
                double fp = FP.containsKey(key) ? FP.get(key) : 0;

                //计算 precision、recall、F1 三个指标
                double precision = tp / (tp + fp);
                double recall = tp / (tp + fn);
                double F1 = 2*tp / (2*tp + fp + fn);

                context.write(new Text("precision"), new DoubleWritable(precision));
                context.write(new Text("recall"), new DoubleWritable(recall));
                context.write(new Text("F1"), new DoubleWritable(F1));
            }
        }
    }

    public static class EvaluationReducer extends Reducer<Text, DoubleWritable, Text,
DoubleWritable> {
        @Override
        protected void reduce(Text key, Iterable<DoubleWritable> values, Context context) throws
IOException, InterruptedException {
            double sum = 0.0;
            int num = 0;
            for(DoubleWritable val : values){
                sum += val.get();
                num++;
            }
            context.write(key, new DoubleWritable(sum / num));
        }
    }

    public int run(String[] args) throws Exception {
        Configuration conf = this.getConf();
        Job job = Job.getInstance(conf, "Evaluation");

        job.setJarByClass(Evaluation.class);
        job.setMapperClass(EvaluationMapper.class);
        job.setCombinerClass(EvaluationReducer.class);
        job.setReducerClass(EvaluationReducer.class);

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

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        return job.waitForCompletion(true) ? 0 : 1;
    }

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

4. 数据集说明

该工程选择的是 Country 的类别，在具体的类别选取上使用了两种方法。按照 8：2 的比例随机划分训练集和测试集。

1. 文件个数大于 80 个的类别

{"ALB":81,"ARG":108,"AUSTR":305,"BELG":154,"BRAZ":200,"CANA":263,"CHINA":255,"CZREP":127,"EEC":182,"FIN":83,"FRA":358,"GFR":257,"HKONG":108,"INDIA":326,"INDON":137,"ISRAEL":145,"ITALY":131,"JAP":470,"MALAY":99,"MEX":144,"NETH":125,"PHLNS":132,"POL":127,"RUSS":148,"SAFR":147,"SINGP":137,"SKOREA":83,"THAIL":81,"UK":790,"USA":3137}

2. 文件个数大于 80 小于 100 个的类别

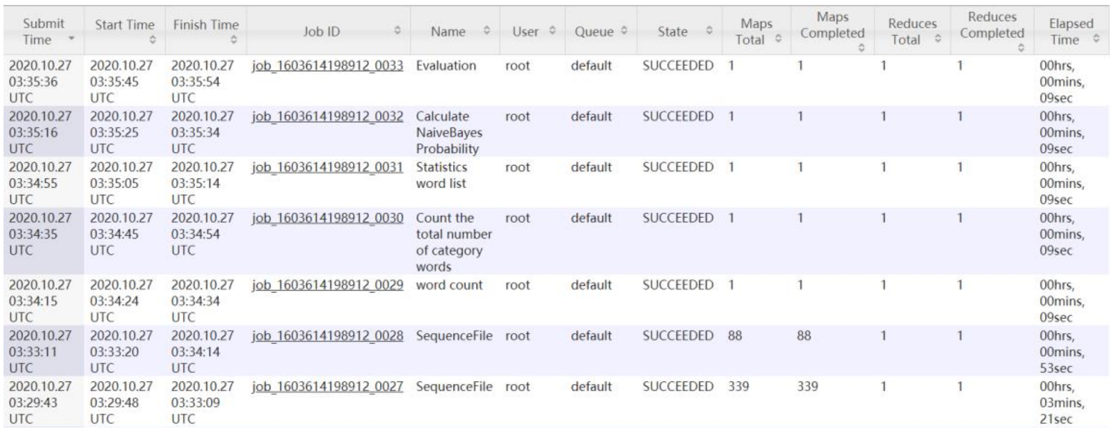
{"ALB":81,"FIN":83,"MALAY":99,"SKOREA":83,"THAIL":81}

5. 程序运行说明

本实验使用的是 hadoop 单机伪分布式模式。启动的模块为一个 namenode、一个 datanode、一个 secondary namenode、resource manager、node manage 以及 jobhistory。

Main 类是一个启动类，传入的参数形式为“类别 类别 类别 … 输出地址”。

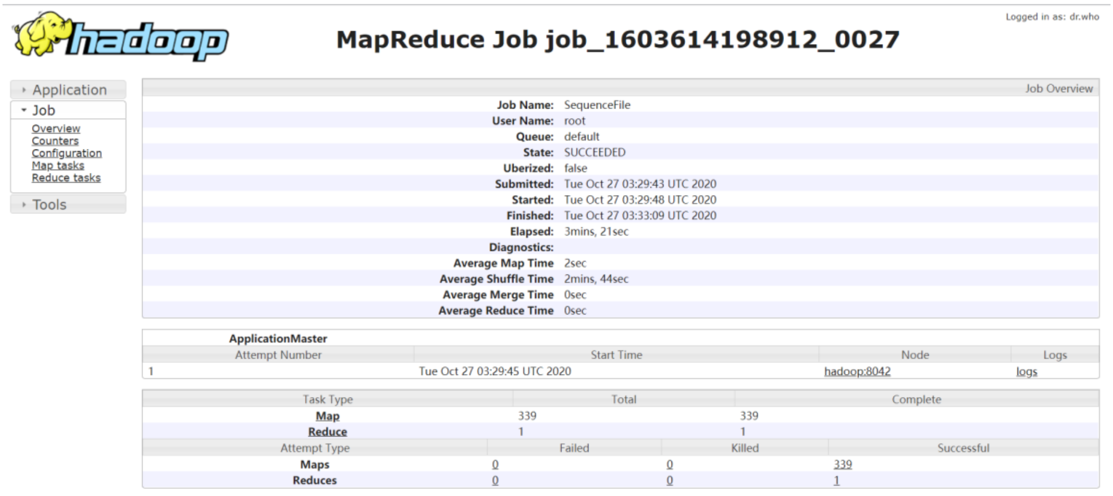
每运行一次程序会生成 7 个 job（因为有 7 个 MapReduce 程序），每个 job 的 map 和 reduce 个数如下图所示：



Submit Time	Start Time	Finish Time	Job ID	Name	User	Queue	State	Maps Total	Maps Completed	Reduces Total	Reduces Completed	Elapsed Time
2020.10.27 03:35:36 UTC	2020.10.27 03:35:45 UTC	2020.10.27 03:35:54 UTC	job_1603614198912_0033	Evaluation	root	default	SUCCEEDED	1	1	1	1	00hrs, 00mins, 09sec
2020.10.27 03:35:16 UTC	2020.10.27 03:35:25 UTC	2020.10.27 03:35:34 UTC	job_1603614198912_0032	Calculate NaiveBayes Probability	root	default	SUCCEEDED	1	1	1	1	00hrs, 00mins, 09sec
2020.10.27 03:34:55 UTC	2020.10.27 03:35:05 UTC	2020.10.27 03:35:14 UTC	job_1603614198912_0031	Statistics word list	root	default	SUCCEEDED	1	1	1	1	00hrs, 00mins, 09sec
2020.10.27 03:34:35 UTC	2020.10.27 03:34:45 UTC	2020.10.27 03:34:54 UTC	job_1603614198912_0030	Count the total number of category words	root	default	SUCCEEDED	1	1	1	1	00hrs, 00mins, 09sec
2020.10.27 03:34:15 UTC	2020.10.27 03:34:24 UTC	2020.10.27 03:34:34 UTC	job_1603614198912_0029	word count	root	default	SUCCEEDED	1	1	1	1	00hrs, 00mins, 09sec
2020.10.27 03:33:11 UTC	2020.10.27 03:33:20 UTC	2020.10.27 03:34:14 UTC	job_1603614198912_0028	SequenceFile	root	default	SUCCEEDED	88	88	1	1	00hrs, 00mins, 53sec
2020.10.27 03:29:43 UTC	2020.10.27 03:29:48 UTC	2020.10.27 03:33:09 UTC	job_1603614198912_0027	SequenceFile	root	default	SUCCEEDED	339	339	1	1	00hrs, 03mins, 21sec

图 5.1 job 结果图

如图所示，除了序列化文件 job 外，其他 job 的 map 和 reduce 个数都为 1。因为本程序全程采用的 SequenceFile 文件格式，所以不存在小文件。而由于文件的大小小于 block，所以没有分片，因此后面的 job 都只有一个 map。对于序列化文件 job 则是有多少个文件就有多少个 map。



MapReduce Job job_1603614198912_0027

Job Overview

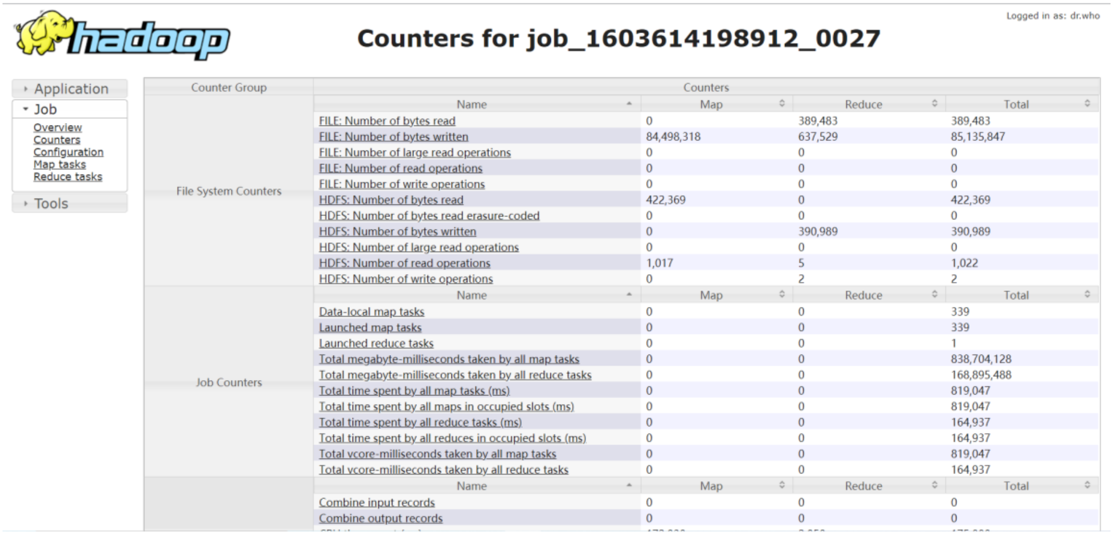
Job Name:	SequenceFile
User Name:	root
Queue:	default
State:	SUCCEEDED
Uberized:	false
Submitted:	Tue Oct 27 03:29:43 UTC 2020
Started:	Tue Oct 27 03:29:48 UTC 2020
Finished:	Tue Oct 27 03:33:09 UTC 2020
Elapsed:	3mins, 21sec
Diagnostics:	
Average Map Time	2sec
Average Shuffle Time	2mins, 44sec
Average Merge Time	0sec
Average Reduce Time	0sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Tue Oct 27 03:29:45 UTC 2020	hadoop:8042	logs

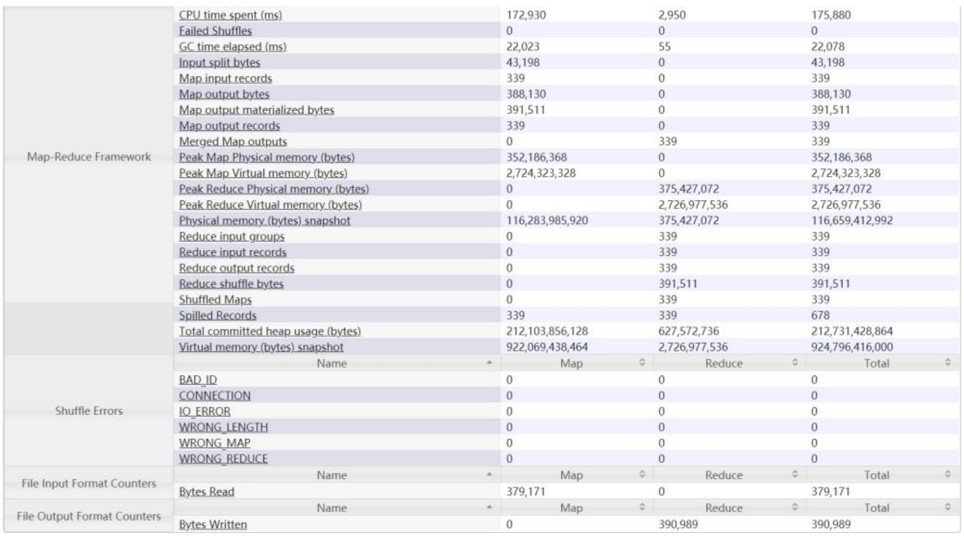
Task Type	Total	Complete
Map	339	339
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	339
Reduces	0	0	1



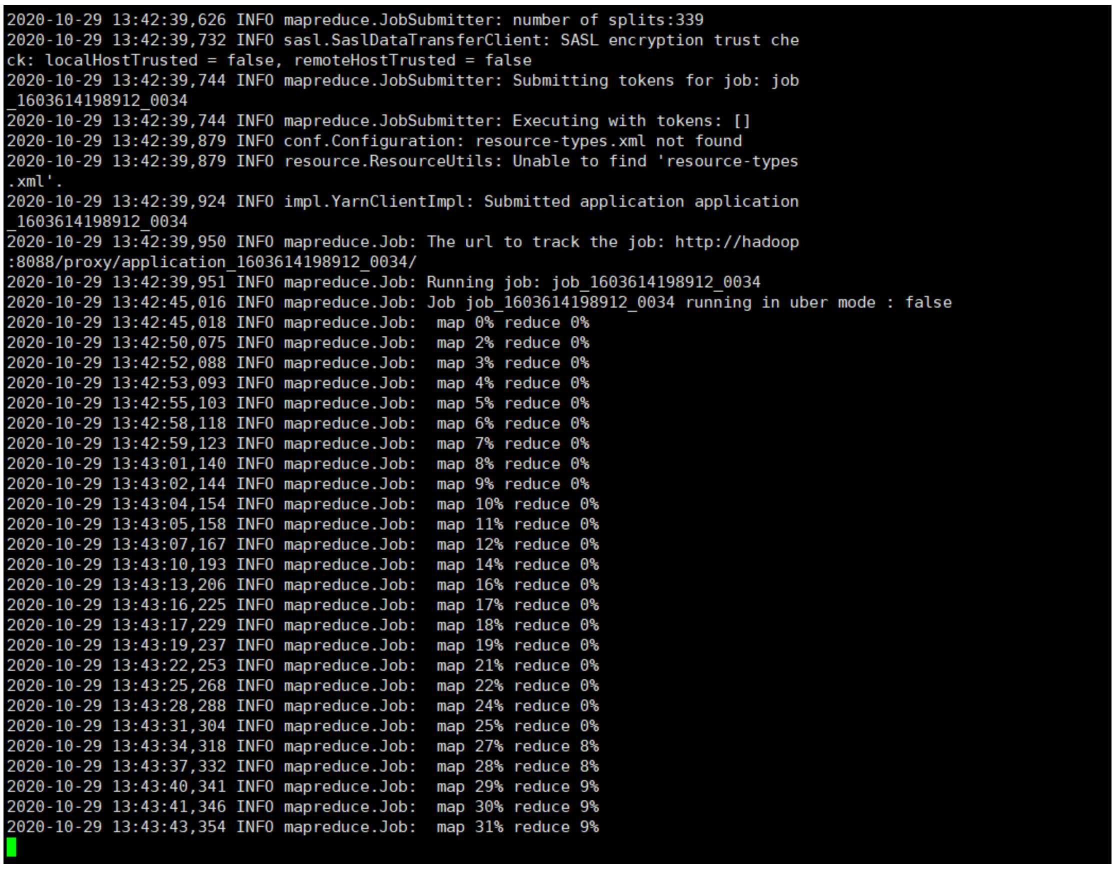
Counters for job_1603614198912_0027

Counter Group	Name	Map	Reduce	Total
File System Counters	FILE: Number of bytes read	0	389,483	389,483
	FILE: Number of bytes written	84,498,318	637,529	85,135,847
	FILE: Number of large read operations	0	0	0
	FILE: Number of read operations	0	0	0
	FILE: Number of write operations	0	0	0
	HDFS: Number of bytes read	422,369	0	422,369
	HDFS: Number of bytes read erasure-coded	0	0	0
	HDFS: Number of bytes written	0	390,989	390,989
	HDFS: Number of large read operations	0	0	0
	HDFS: Number of read operations	1,017	5	1,022
	HDFS: Number of write operations	0	2	2
Job Counters	Data-local map tasks	0	0	339
	Launched map tasks	0	0	339
	Launched reduce tasks	0	0	1
	Total megabyte-milliseconds taken by all map tasks	0	0	838,704,128
	Total megabyte-milliseconds taken by all reduce tasks	0	0	168,895,488
	Total time spent by all map tasks (ms)	0	0	819,047
	Total time spent by all maps in occupied slots (ms)	0	0	819,047
	Total time spent by all reduce tasks (ms)	0	0	164,937
	Total time spent by all reduces in occupied slots (ms)	0	0	164,937
	Total vcore-milliseconds taken by all map tasks	0	0	819,047
	Total vcore-milliseconds taken by all reduce tasks	0	0	164,937
	Combine input records	0	0	0
	Combine output records	0	0	0



Map-Reduce Framework	CPU time spent (ms)	172,930	2,950	175,880
	Failed Shuffles	0	0	0
	GC time elapsed (ms)	22,023	55	22,078
	Input split bytes	43,198	0	43,198
	Map input records	339	0	339
	Map output bytes	388,130	0	388,130
	Map output materialized bytes	391,511	0	391,511
	Map output records	339	0	339
	Merged Map outputs	0	339	339
	Peak Map Physical memory (bytes)	352,186,368	0	352,186,368
	Peak Map Virtual memory (bytes)	2,724,323,328	0	2,724,323,328
	Peak Reduce Physical memory (bytes)	0	375,427,072	375,427,072
	Peak Reduce Virtual memory (bytes)	0	2,726,977,536	2,726,977,536
	Physical memory (bytes) snapshot	116,283,985,920	375,427,072	116,659,412,992
	Reduce input groups	0	339	339
	Reduce input records	0	339	339
	Reduce output records	0	339	339
	Reduce shuffle bytes	0	391,511	391,511
	Shuffled Maps	0	339	339
	Spilled Records	339	339	678
	Total committed heap usage (bytes)	212,103,856,128	627,572,736	212,731,428,864
	Virtual memory (bytes) snapshot	922,069,438,464	2,726,977,536	924,796,416,000
Shuffle Errors	Name	Map	Reduce	Total
	BAD_ID	0	0	0
	CONNECTION	0	0	0
	IO_ERROR	0	0	0
	WRONG_LENGTH	0	0	0
	WRONG_MAP	0	0	0
	WRONG_REDUCE	0	0	0
File Input Format Counters	Name	Map	Reduce	Total
	Bytes Read	379,171	0	379,171
File Output Format Counters	Name	Map	Reduce	Total
	Bytes Written	0	390,989	390,989

图 5.2 Web 页面的作业监控截图



```
2020-10-29 13:42:39,626 INFO mapreduce.JobSubmitter: number of splits:339
2020-10-29 13:42:39,732 INFO sasl.SaslDataTransferClient: SASL encryption trust che
ck: localHostTrusted = false, remoteHostTrusted = false
2020-10-29 13:42:39,744 INFO mapreduce.JobSubmitter: Submitting tokens for job: job
_1603614198912_0034
2020-10-29 13:42:39,744 INFO mapreduce.JobSubmitter: Executing with tokens: []
2020-10-29 13:42:39,879 INFO conf.Configuration: resource-types.xml not found
2020-10-29 13:42:39,879 INFO resource.ResourceUtils: Unable to find 'resource-types
.xml'.
2020-10-29 13:42:39,924 INFO impl.YarnClientImpl: Submitted application application
_1603614198912_0034
2020-10-29 13:42:39,950 INFO mapreduce.Job: The url to track the job: http://hadoop
:8088/proxy/application_1603614198912_0034/
2020-10-29 13:42:39,951 INFO mapreduce.Job: Running job: job_1603614198912_0034
2020-10-29 13:42:45,016 INFO mapreduce.Job: Job job_1603614198912_0034 running in uber mode : false
2020-10-29 13:42:45,018 INFO mapreduce.Job:  map 0% reduce 0%
2020-10-29 13:42:50,075 INFO mapreduce.Job:  map 2% reduce 0%
2020-10-29 13:42:52,088 INFO mapreduce.Job:  map 3% reduce 0%
2020-10-29 13:42:53,093 INFO mapreduce.Job:  map 4% reduce 0%
2020-10-29 13:42:55,103 INFO mapreduce.Job:  map 5% reduce 0%
2020-10-29 13:42:58,118 INFO mapreduce.Job:  map 6% reduce 0%
2020-10-29 13:42:59,123 INFO mapreduce.Job:  map 7% reduce 0%
2020-10-29 13:43:01,140 INFO mapreduce.Job:  map 8% reduce 0%
2020-10-29 13:43:02,144 INFO mapreduce.Job:  map 9% reduce 0%
2020-10-29 13:43:04,154 INFO mapreduce.Job:  map 10% reduce 0%
2020-10-29 13:43:05,158 INFO mapreduce.Job:  map 11% reduce 0%
2020-10-29 13:43:07,167 INFO mapreduce.Job:  map 12% reduce 0%
2020-10-29 13:43:10,193 INFO mapreduce.Job:  map 14% reduce 0%
2020-10-29 13:43:13,206 INFO mapreduce.Job:  map 16% reduce 0%
2020-10-29 13:43:16,225 INFO mapreduce.Job:  map 17% reduce 0%
2020-10-29 13:43:17,229 INFO mapreduce.Job:  map 18% reduce 0%
2020-10-29 13:43:19,237 INFO mapreduce.Job:  map 19% reduce 0%
2020-10-29 13:43:22,253 INFO mapreduce.Job:  map 21% reduce 0%
2020-10-29 13:43:25,268 INFO mapreduce.Job:  map 22% reduce 0%
2020-10-29 13:43:28,288 INFO mapreduce.Job:  map 24% reduce 0%
2020-10-29 13:43:31,304 INFO mapreduce.Job:  map 25% reduce 0%
2020-10-29 13:43:34,318 INFO mapreduce.Job:  map 27% reduce 8%
2020-10-29 13:43:37,332 INFO mapreduce.Job:  map 28% reduce 8%
2020-10-29 13:43:40,341 INFO mapreduce.Job:  map 29% reduce 9%
2020-10-29 13:43:41,346 INFO mapreduce.Job:  map 30% reduce 9%
2020-10-29 13:43:43,354 INFO mapreduce.Job:  map 31% reduce 9%
```

图 5.3 程序运行截图

6. 实验结果分析

表 6.1 评估表

	Precision	Recall	F1
文件个数大于 80	0.90946553228791	0.53818081922001	0.63129355768104
大于 80 小于 100	0.98888888888888	0.98823529411764	0.98822510822510

从上表中我们可以看出选取文件个数大于 80 小于 100 的类型来训练模型的评估结果非

常的高，而第一种选法的 F1 知州 0.63。从两者对比中可以看出类别分布不均会导致模型的评估结果差.