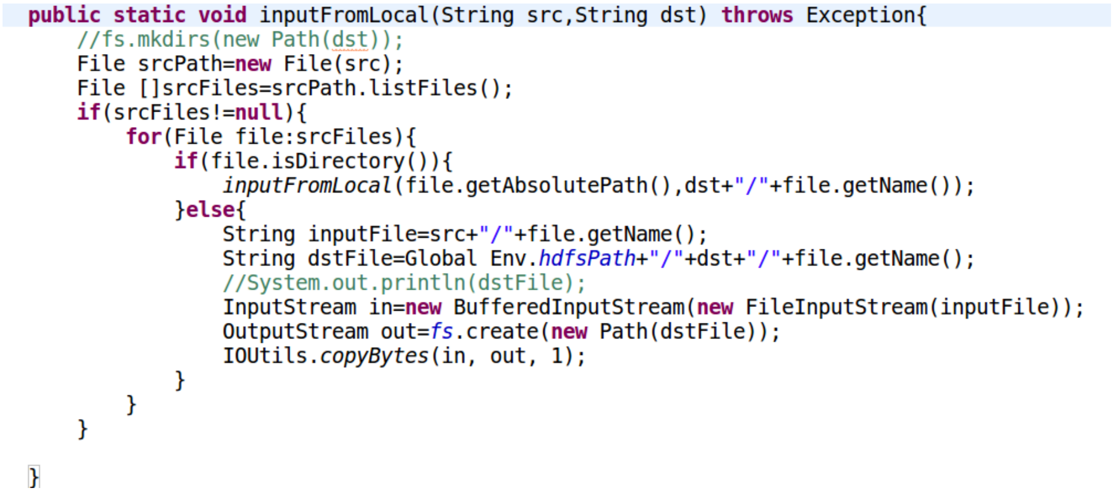
一 Project 内容

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

1.1：将本地文件读入 hdfs 文件系统

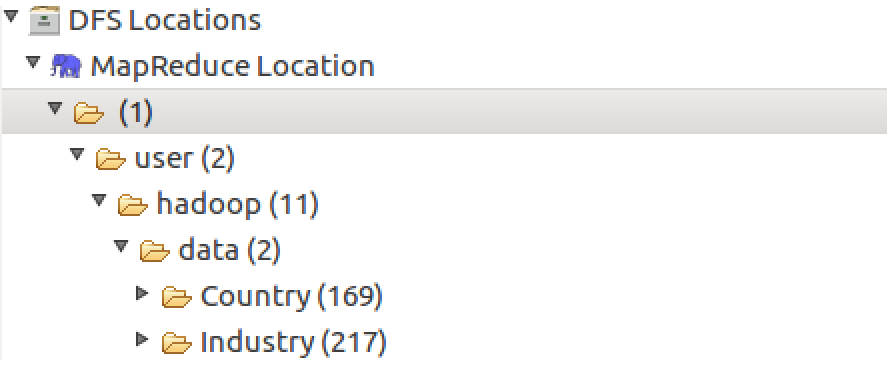
将本地文件系统上的 data 文件夹递归复制到 hdfs 文件系统，方法的参数分别为本地路径"/home/hadoop/workspace/data/NBCorpus/Industry"，与 hdfs 文件系统下指定的路径 usr/hadoop/data。

Class Path: tools/InputFromLocal.java



```java
public static void inputFromLocal(String src,String dst) throws Exception{
    //fs.mkdirs(new Path(dst));
    File srcPath=new File(src);
    File []srcFiles=srcPath.listFiles();
    if(srcFiles!=null){
        for(File file:srcFiles){
            if(file.isDirectory()){
                inputFromLocal(file.getAbsolutePath(),dst+"/"+file.getName());
            }else{
                String inputFile=src+"/"+file.getName();
                String dstFile=Global Env.hdfsPath+"/"+dst+"/"+file.getName();
                //System.out.println(dstFile);
                InputStream in=new BufferedInputStream(new FileInputStream(inputFile));
                OutputStream out=fs.create(new Path(dstFile));
                IOUtils.copyBytes(in, out, 1);
            }
        }
    }

}
```

导入结果如下：

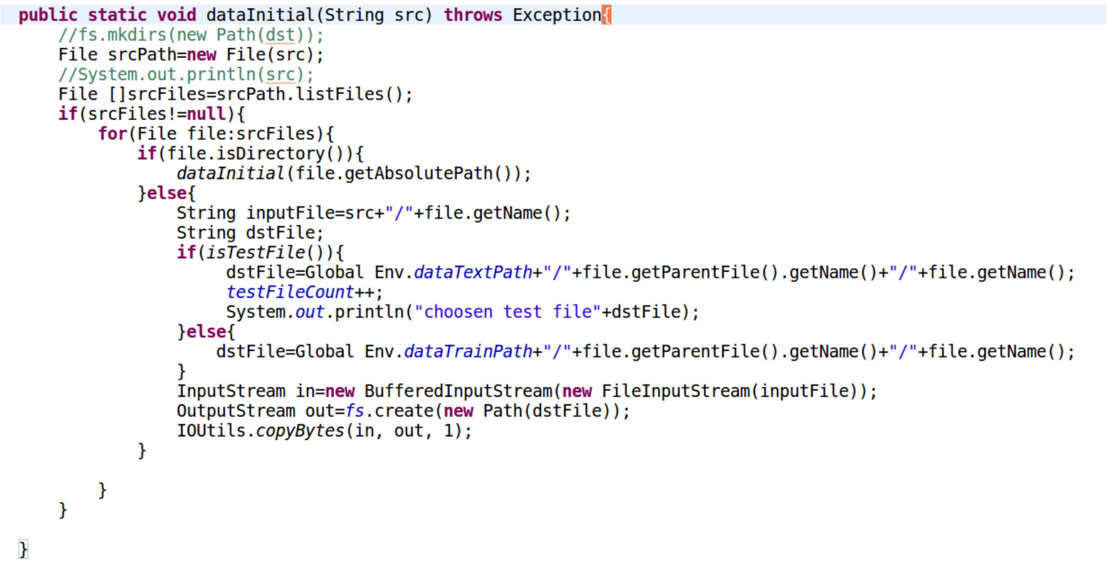


```
▼ DFS Locations
  ▼ MapReduce Location
    ▼ (1)
      ▼ user (2)
        ▼ hadoop (11)
          ▼ data (2)
            ▶ Country (169)
            ▶ Industry (217)
```

1.2：从文件中分出测试集与训练集

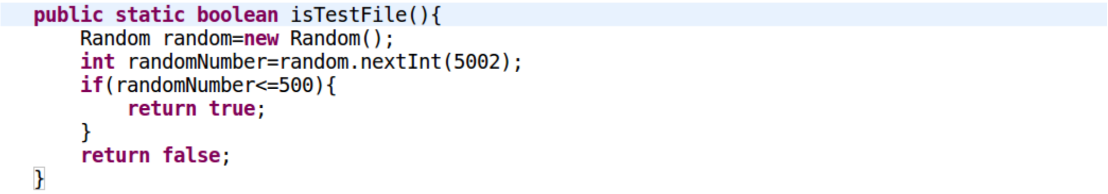
Industry 目录下的文件总数为 4999，随机从数据集中随机抽取 10%作为测试文件，90%作为训练文件

Class Path: tools/DataInitialize.java



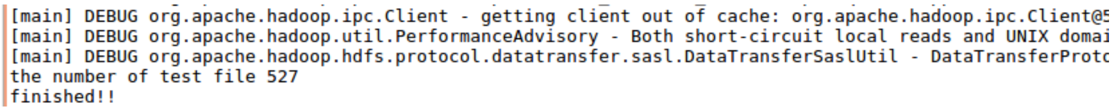
```
public static void dataInitial(String src) throws Exception{
    //fs.mkdirs(new Path(dst));
    File srcPath=new File(src);
    //System.out.println(src);
    File []srcFiles=srcPath.listFiles();
    if(srcFiles!=null){
        for(File file:srcFiles){
            if(file.isDirectory()){
                dataInitial(file.getAbsolutePath());
            }else{
                String inputFile=src+"/"+file.getName();
                String dstFile;
                if(isTestFile()){
                     dstFile=Global Env.dataTextPath+"/"+file.getParentFile().getName()+"/"+file.getName();
                     testFileCount++;
                     System.out.println("choosen test file"+dstFile);
                }else{
                    dstFile=Global Env.dataTrainPath+"/"+file.getParentFile().getName()+"/"+file.getName();
                }
                InputStream in=new BufferedInputStream(new FileInputStream(inputFile));
                OutputStream out=fs.create(new Path(dstFile));
                IOUtils.copyBytes(in, out, 1);
            }

        }
    }

}
```

其中选择方法使用随机数实现



```
public static boolean isTestFile(){
    Random random=new Random();
    int randomNumber=random.nextInt(5002);
    if(randomNumber<=500){
        return true;
    }
    return false;
}
```

结果如下：



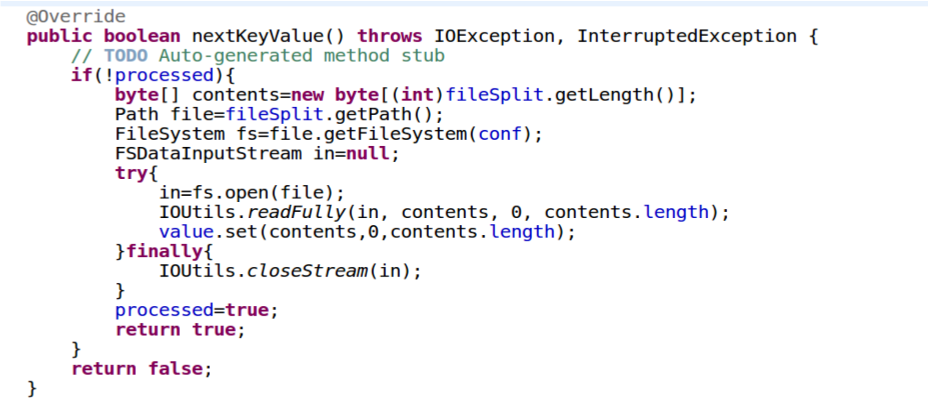
```
[main] DEBUG org.apache.hadoop.ipc.Client - getting client out of cache: org.apache.hadoop.ipc.Client@5
[main] DEBUG org.apache.hadoop.util.PerformanceAdvisory - Both short-circuit local reads and UNIX domai
[main] DEBUG org.apache.hadoop.hdfs.protocol.datatransfer.sasl.DataTransferSaslUtil - DataTransferProtc
the number of test file 527
finished!!
```

1.3：实现 SequenceFile

由于 Industry 目录是由许多小文件组成，为了加快处理速度先将小文件合成为一个 SequenceFile。

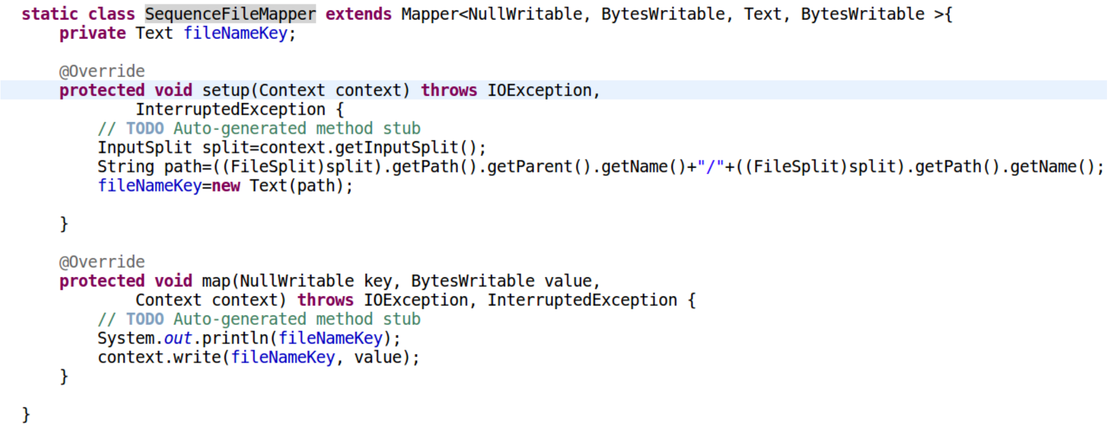
首先，定义 WholeFileRecodReader，读取整个文件的值作为一个 record 的值。其主要方法 nextKeyValue 实现如下：

Class Path：tools/WholeFileRecodReader.java



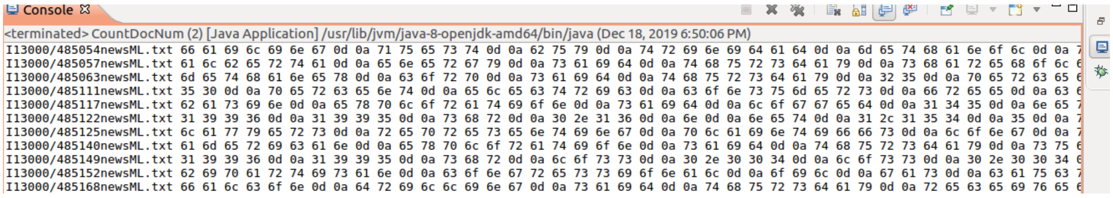
```
@Override
public boolean nextKeyValue() throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    if(!processed){
        byte[] contents=new byte[(int)fileSplit.getLength()];
        Path file=fileSplit.getPath();
        FileSystem fs=file.getFileSystem(conf);
        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;
}
```

其次，将多个文件打包成一个SequenceFile，其中SequenceFileMapper实现如下。
map函数中将key设置为父路径名+文件名，即类名+文件名。
Class Path: tools/SmallFilesToSequenceFileConverter.java



```java
static class SequenceFileMapper extends Mapper<NullWritable, BytesWritable, Text, BytesWritable >{
    private Text fileNameKey;

    @Override
    protected void setup(Context context) throws IOException,
            InterruptedException {
        // TODO Auto-generated method stub
        InputSplit split=context.getInputSplit();
        String path=((FileSplit)split).getPath().getParent().getName()+"/"+((FileSplit)split).getPath().getName();
        fileNameKey=new Text(path);

    }

    @Override
    protected void map(NullWritable key, BytesWritable value,
            Context context) throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        System.out.println(fileNameKey);
        context.write(fileNameKey, value);
    }

}
```

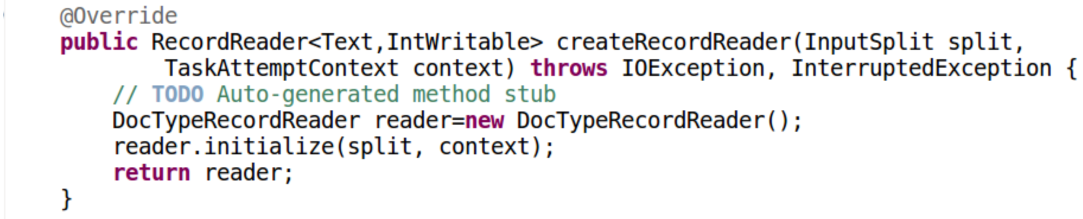
获取的SequenceFile可通过命令行或代码读取，这里实现read函数读取输出的SequenceFile如下：



```
<terminated> CountDocNum (2) [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (Dec 18, 2019 6:50:06 PM)
I13000/485054newsML.txt 66 61 69 6c 69 6e 67 0d 0a 71 75 65 73 74 0d 0a 62 75 79 0d 0a 74 72 69 6e 69 64 61 64 0d 0a 6d 65 74 68 61 6e 6f 6c 0d 0a
I13000/485057newsML.txt 61 6c 62 65 72 74 61 0d 0a 65 6e 65 72 67 79 0d 0a 73 61 69 64 0d 0a 74 68 75 72 73 64 61 79 0d 0a 73 68 61 72 65 68 6f 6c
I13000/485063newsML.txt 6d 65 74 68 61 6e 65 78 0d 0a 63 6f 72 70 0d 0a 73 61 69 64 0d 0a 74 68 75 72 73 64 61 79 0d 0a 32 35 0d 0a 70 65 72 63 65
I13000/485111newsML.txt 35 30 0d 0a 70 65 72 63 65 6e 74 0d 0a 65 6c 65 63 74 72 69 63 0d 0a 63 6f 6e 73 75 6d 65 72 73 0d 0a 66 72 65 65 0d 0a 63
I13000/485117newsML.txt 62 61 73 69 6e 0d 0a 65 78 70 6c 6f 72 61 74 69 6f 6e 0d 0a 73 61 69 64 0d 0a 6c 6f 67 67 65 64 0d 0a 31 34 35 0d 0a 6e 65
I13000/485122newsML.txt 31 39 39 36 0d 0a 31 39 39 35 0d 0a 73 68 72 0d 0a 30 2e 31 36 0d 0a 6e 0d 0a 6e 65 74 0d 0a 31 2c 31 35 34 0d 0a 35 0d 0a
I13000/485125newsML.txt 6c 61 77 79 65 72 73 0d 0a 72 65 70 72 65 73 65 6e 74 69 6e 67 0d 0a 70 6c 61 69 6e 74 69 66 66 73 0d 0a 6c 6f 6e 67 0d 0a
I13000/485140newsML.txt 61 6d 65 72 69 63 61 6e 0d 0a 65 78 70 6c 6f 72 61 74 69 6f 6e 0d 0a 73 61 69 64 0d 0a 74 68 75 72 73 64 61 79 0d 0a 73 75
I13000/485149newsML.txt 31 39 39 36 0d 0a 31 39 39 35 0d 0a 73 68 72 0d 0a 6c 6f 73 73 0d 0a 30 2e 30 30 34 0d 0a 6c 6f 73 73 0d 0a 30 2e 30 30 34
I13000/485152newsML.txt 62 69 70 61 72 74 69 73 61 6e 0d 0a 63 6f 6e 67 72 65 73 73 69 6f 6e 61 6c 0d 0a 6f 69 6c 0d 0a 67 61 73 0d 0a 63 61 75 63
I13000/485168newsML.txt 66 61 6c 63 6f 6e 0d 0a 64 72 69 6c 6c 69 6e 67 0d 0a 73 61 69 64 0d 0a 74 68 75 72 73 64 61 79 0d 0a 72 65 63 65 69 76 65
```

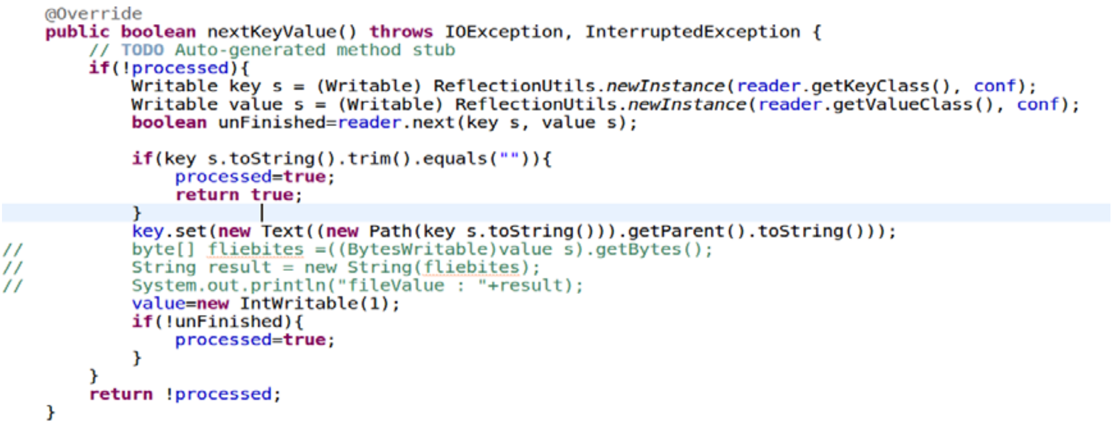
1.4：训练先验概率

首先，定义DocTypeInputFormat.java与DocTypeRecordReader.java，在DocTypeInputFormat的createRecordReader函数中生成DocTypeRecordReader的实例，代码如下：



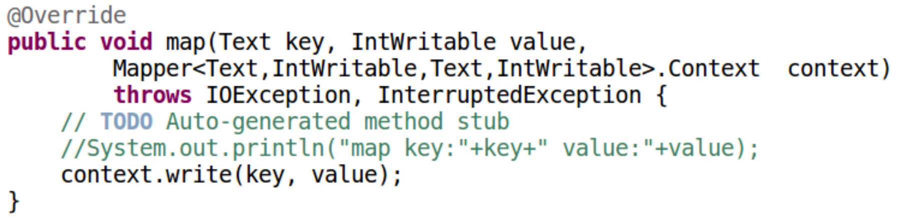
```java
@Override
public RecordReader<Text,IntWritable> createRecordReader(InputSplit split,
        TaskAttemptContext context) throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    DocTypeRecordReader reader=new DocTypeRecordReader();
    reader.initialize(split, context);
    return reader;
}
```

在DocTypeRecordReader对于每个文件，将<文件所在的目录类别，1>作为输出，具体实现在DocTypeRecordReader的nextKeyValue()方法中。
由于在SequenceFile将key设置为了父路径名+文件名，这里直接获取父路径即可。



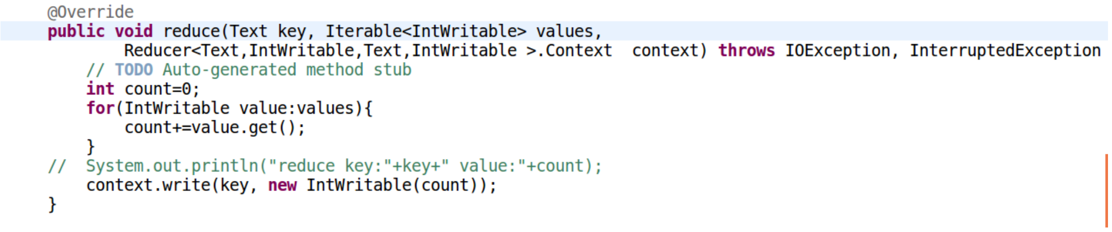
```java
    @Override
    public boolean nextKeyValue() throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        if(!processed){
            Writable key_s = (Writable) ReflectionUtils.newInstance(reader.getKeyClass(), conf);
            Writable value_s = (Writable) ReflectionUtils.newInstance(reader.getValueClass(), conf);
            boolean unFinished=reader.next(key_s, value_s);

            if(key_s.toString().trim().equals("")){
                processed=true;
                return true;
            }
            key.set(new Text((new Path(key_s.toString())).getParent().toString()));
//          byte[] fliebites =((BytesWritable)value_s).getBytes();
//          String result = new String(fliebites);
//          System.out.println("fileValue : "+result);
            value=new IntWritable(1);
            if(!unFinished){
                processed=true;
            }
        }
        return !processed;
    }
```

最后在 CountDocType.java 中实现 Map 与 Reduce
在内部类 CountTypeNumMapper.java 的 map 方法中直接将<key,value>原样输出。



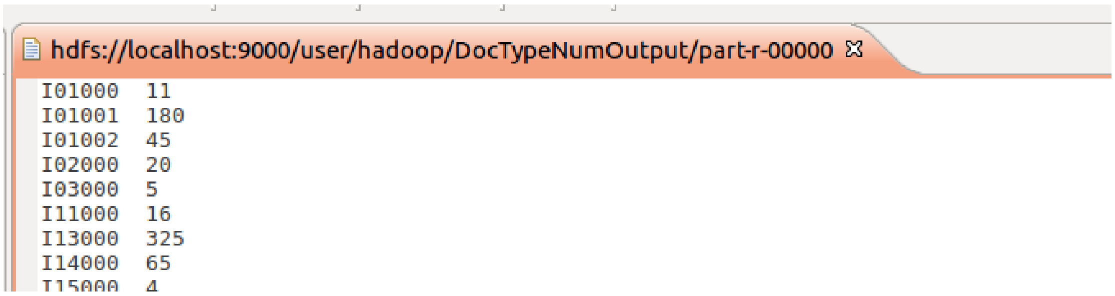
```
@Override
public void map(Text key, IntWritable value,
        Mapper<Text,IntWritable,Text,IntWritable>.Context  context)
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    //System.out.println("map key:"+key+" value:"+value);
    context.write(key, value);
}
```

在内部类 CountTypeNumReduce.java 的 Reduce 方法中对 map 中的 value 进行求和：



```
    @Override
    public void reduce(Text key, Iterable<IntWritable> values,
            Reducer<Text,IntWritable,Text,IntWritable >.Context  context) throws IOException, InterruptedException
        // TODO Auto-generated method stub
        int count=0;
        for(IntWritable value:values){
            count+=value.get();
        }
    //  System.out.println("reduce key:"+key+" value:"+count);
        context.write(key, new IntWritable(count));
    }
```

最终，作业输出文件中每一行的格式为类名+文档总数。输出如下：

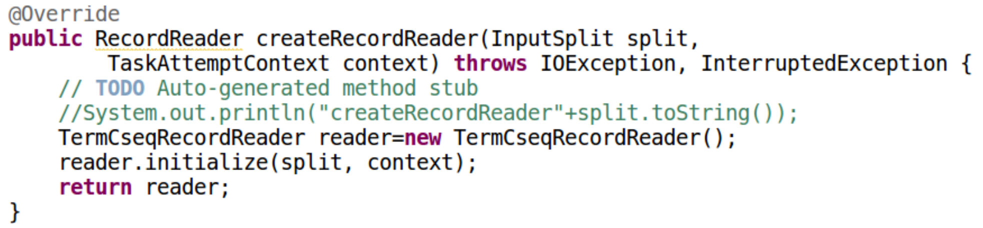


hdfs://localhost:9000/user/hadoop/DocTypeNumOutput/part-r-00000

```
I01000  11
I01001  180
I01002  45
I02000  20
I03000  5
I11000  16
I13000  325
I14000  65
I15000  4
```

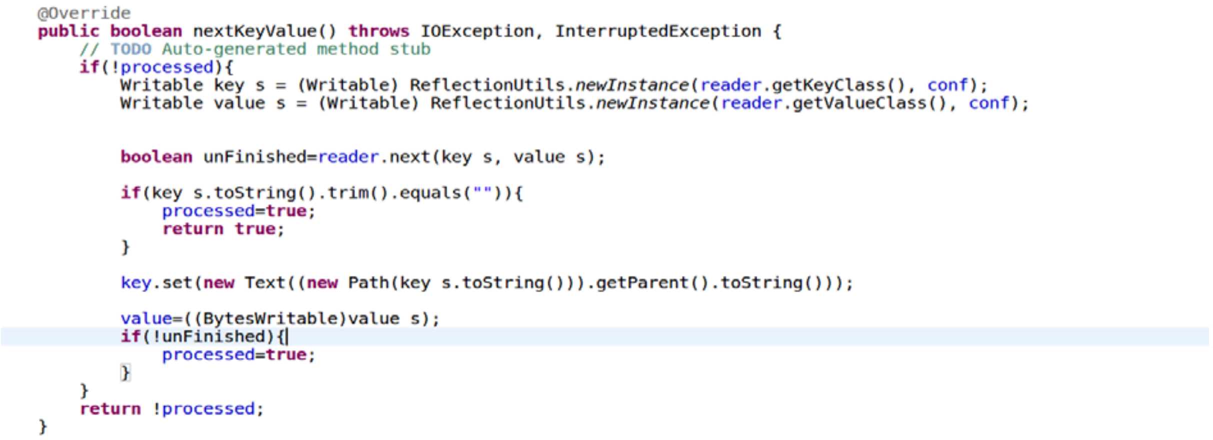
1.5：训练条件概率

首先定义 TermCNumInputFromat.java 和 TermCseqRecordReader.java
在 TermCNumInputFromat.java 中生成 TermCseqRecordReader 的实例如下：



```
@Override
public RecordReader createRecordReader(InputSplit split,
        TaskAttemptContext context) throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    //System.out.println("createRecordReader"+split.toString());
    TermCseqRecordReader reader=new TermCseqRecordReader();
    reader.initialize(split, context);
    return reader;
}
```

在 TermCseqRecordReader 中，将<文件父路径名，文件二进制流>作为输出。
TermCseqRecordReader 的 nextKeyValue()函数实现如下：

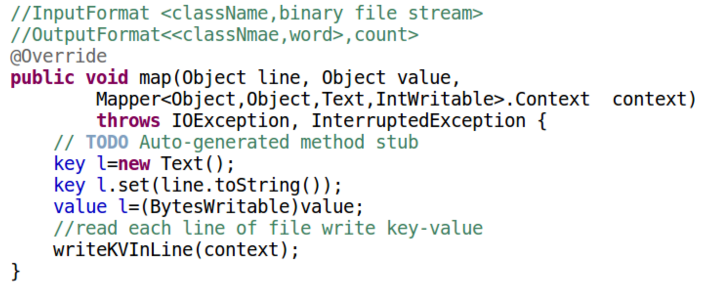


```
@Override
public boolean nextKeyValue() throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    if(!processed){
        Writable key s = (Writable) ReflectionUtils.newInstance(reader.getKeyClass(), conf);
        Writable value s = (Writable) ReflectionUtils.newInstance(reader.getValueClass(), conf);


        boolean unFinished=reader.next(key s, value s);

        if(key s.toString().trim().equals("")){
            processed=true;
            return true;
        }

        key.set(new Text((new Path(key s.toString())).getParent().toString()));

        value=((BytesWritable)value s);
        if(!unFinished){
            processed=true;
        }
    }
    return !processed;
}
```

最后在 CountTermC.java 中实现 CountTermcSeqNumMapper.java 与 CountTermCNumReduce.java。

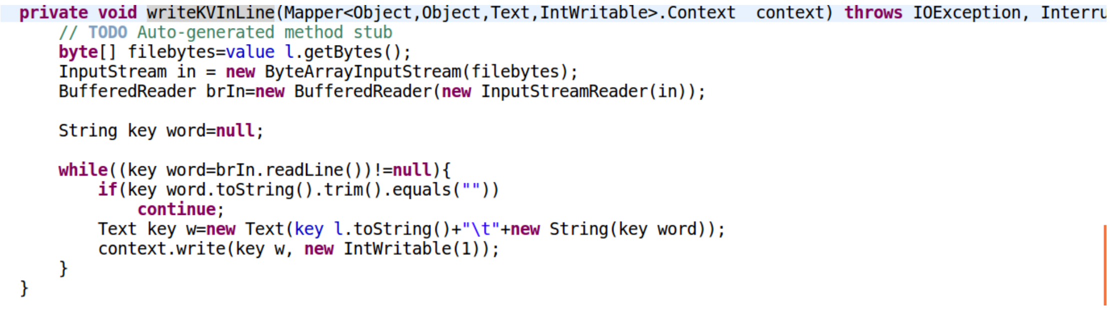
在 CountTermcSeqNumMapper.java 中将读取 value 中的文件二进制流，将<类名-单词，IntWritable(1)>作为输出。

map 函数代码如下：



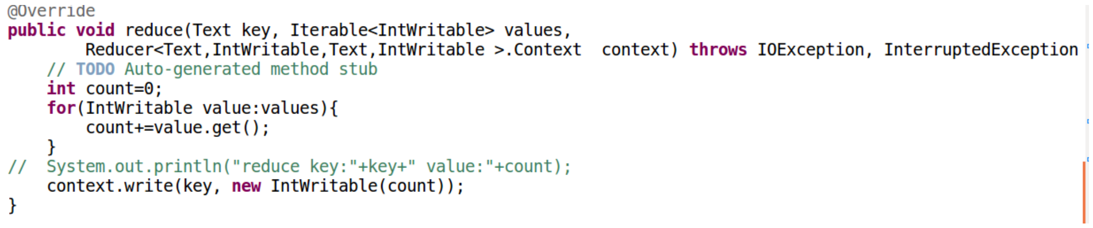
```
//InputFormat <className,binary file stream>
//OutputFormat<<classNmae,word>,count>
@Override
public void map(Object line, Object value,
        Mapper<Object,Object,Text,IntWritable>.Context  context)
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    key l=new Text();
    key l.set(line.toString());
    value l=(BytesWritable)value;
    //read each line of file write key-value
    writeKVInLine(context);
}
```

writeKVInLine()函数读取每一行，在上下文中写入 key-value 对：<<classNmae,word>,count>



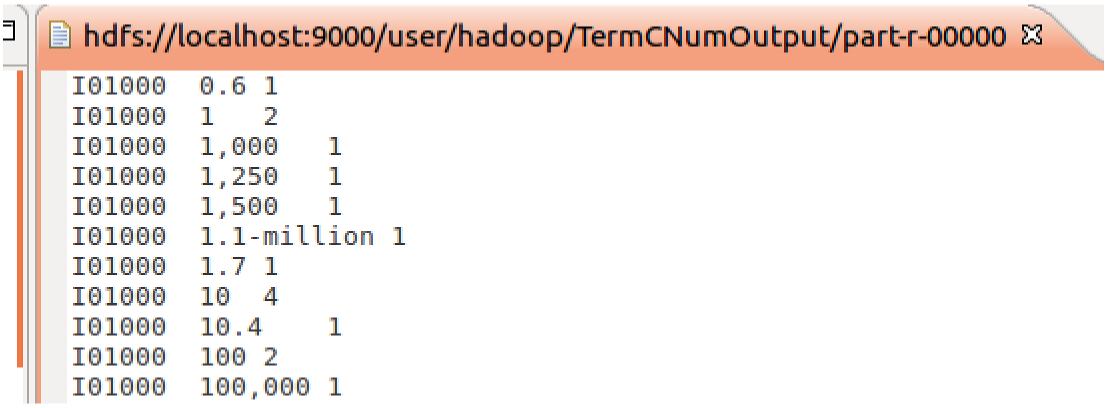
```
private void writeKVInLine(Mapper<Object,Object,Text,IntWritable>.Context  context) throws IOException, Interru
    // TODO Auto-generated method stub
    byte[] filebytes=value l.getBytes();
    InputStream in = new ByteArrayInputStream(filebytes);
    BufferedReader brIn=new BufferedReader(new InputStreamReader(in));

    String key word=null;

    while((key word=brIn.readLine())!=null){
        if(key word.toString().trim().equals(""))
            continue;
        Text key w=new Text(key l.toString()+"\t"+new String(key word));
        context.write(key w, new IntWritable(1));
    }
}
```

在 CountTermCNumReduce.java 中将相同 className-word 的 key 对应的 value 累加，计算出每个单词在每一类中的出现频率，reduce 方法代码如下：



```
@Override
public void reduce(Text key, Iterable<IntWritable> values,
        Reducer<Text,IntWritable,Text,IntWritable >.Context  context) throws IOException, InterruptedException
    // TODO Auto-generated method stub
    int count=0;
    for(IntWritable value:values){
        count+=value.get();
    }
//  System.out.println("reduce key:"+key+" value:"+count);
    context.write(key, new IntWritable(count));
}
```

最终，输出格式为<类名 单词名 单词频数>，如下图：



hdfs://localhost:9000/user/hadoop/TermCNumOutput/part-r-00000

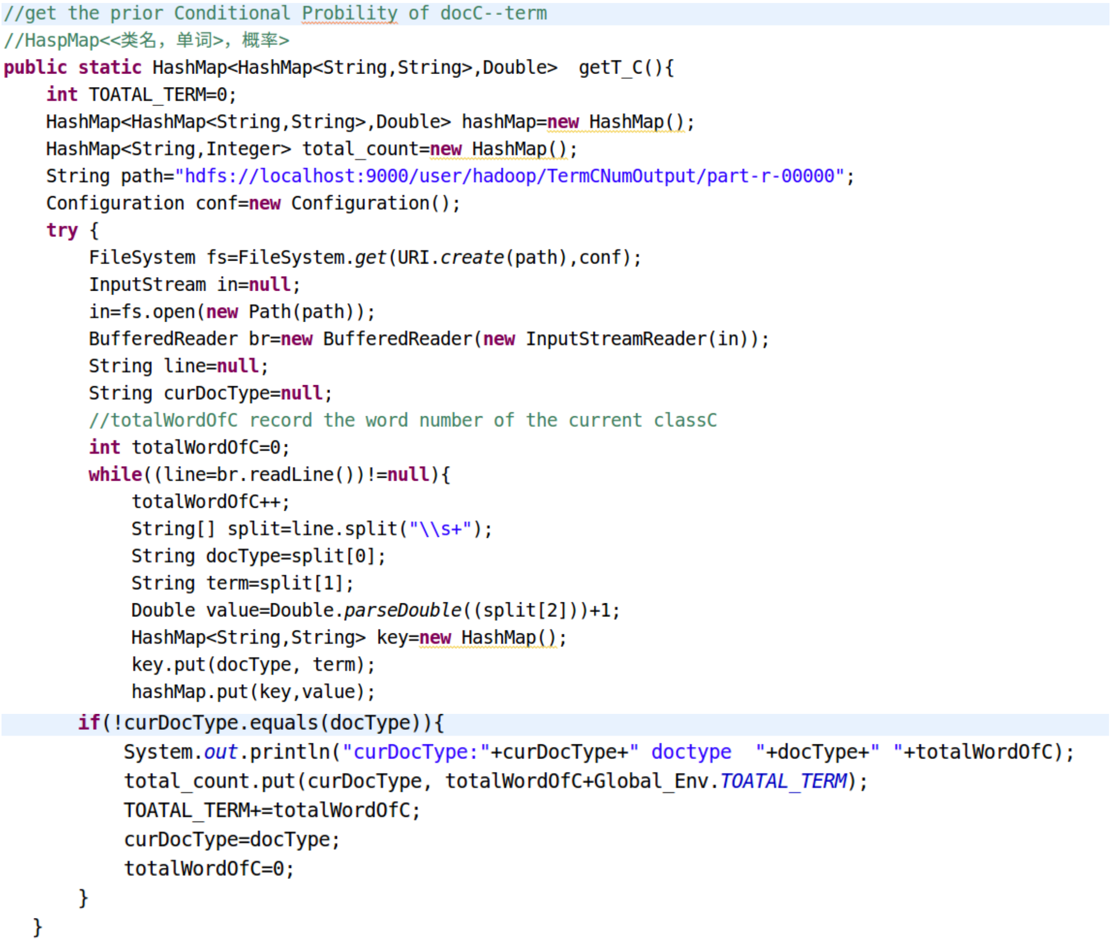
```
I01000  0.6 1
I01000  1   2
I01000  1,000   1
I01000  1,250   1
I01000  1,500   1
I01000  1.1-million 1
I01000  1.7 1
I01000  10  4
I01000  10.4    1
I01000  100 2
I01000  100,000 1
```

至此，条件概率训练完毕。

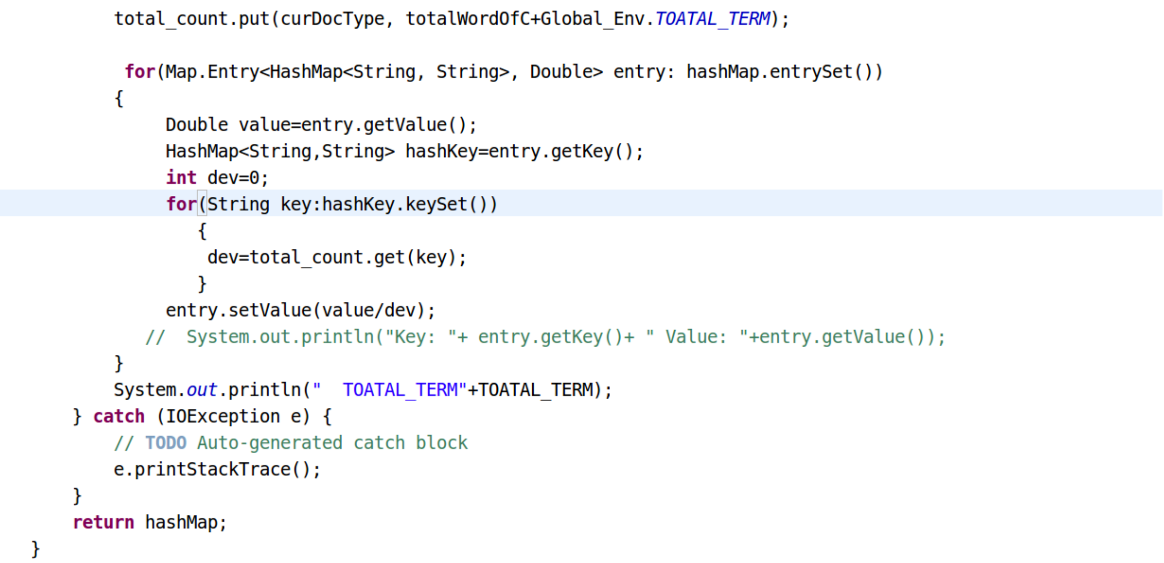
2：用输出的模型对测试集文档进行分类测试。

2.1：计算每个单词属于文档 C 的概率： $\log \hat{P}(t_k|c)$

在 ConditionalProbility.java 中实现 getT_C() 方法，其返回值为 HashMap<HashMap<String,String>,Double>，对应着<<类名，单词>，概率>。其代码如下：



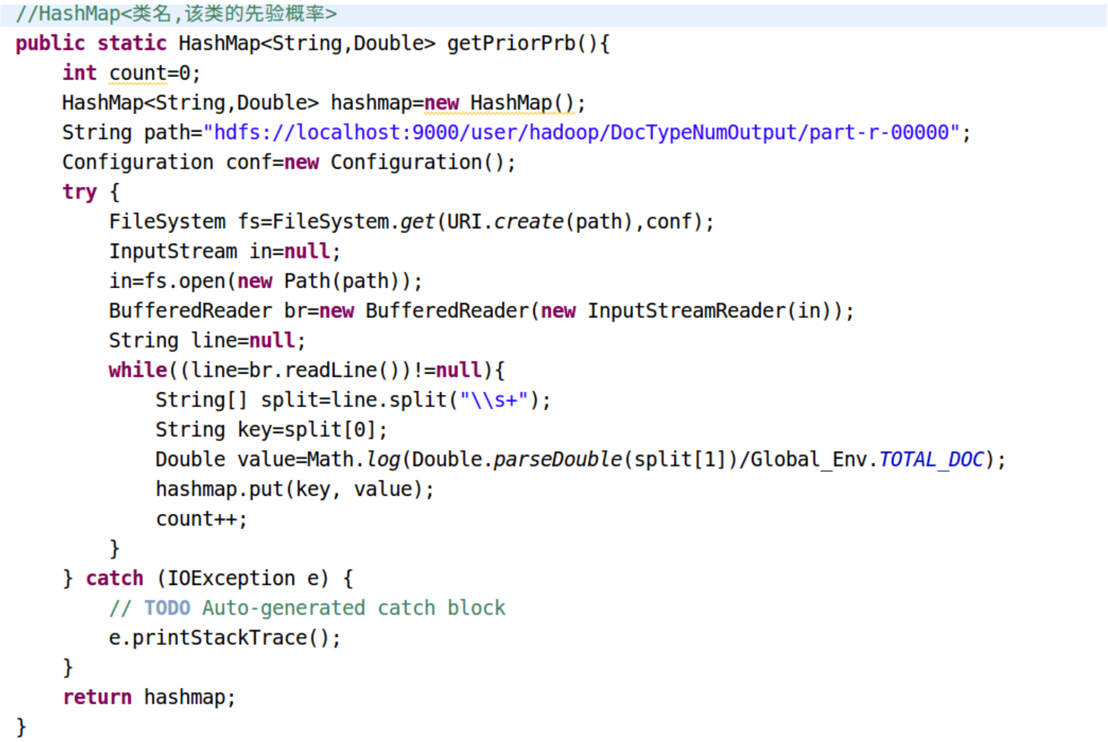
```
//get the prior Conditional Probility of docC--term
//HaspMap<<类名，单词>，概率>
public static HashMap<HashMap<String,String>,Double>  getT_C(){
    int TOATAL_TERM=0;
    HashMap<HashMap<String,String>,Double> hashMap=new HashMap();
    HashMap<String,Integer> total_count=new HashMap();
    String path="hdfs://localhost:9000/user/hadoop/TermCNumOutput/part-r-00000";
    Configuration conf=new Configuration();
    try {
        FileSystem fs=FileSystem.get(URI.create(path),conf);
        InputStream in=null;
        in=fs.open(new Path(path));
        BufferedReader br=new BufferedReader(new InputStreamReader(in));
        String line=null;
        String curDocType=null;
        //totalWordOfC record the word number of the current classC
        int totalWordOfC=0;
        while((line=br.readLine())!=null){
            totalWordOfC++;
            String[] split=line.split("\\s+");
            String docType=split[0];
            String term=split[1];
            Double value=Double.parseDouble((split[2]))+1;
            HashMap<String,String> key=new HashMap();
            key.put(docType, term);
            hashMap.put(key,value);
      if(!curDocType.equals(docType)){
          System.out.println("curDocType:"+curDocType+" doctype  "+docType+" "+totalWordOfC);
          total_count.put(curDocType, totalWordOfC+Global_Env.TOATAL_TERM);
          TOATAL_TERM+=totalWordOfC;
          curDocType=docType;
          totalWordOfC=0;
      }
    }
```



```
        total_count.put(curDocType, totalWordOfC+Global_Env.TOATAL_TERM);

         for(Map.Entry<HashMap<String, String>, Double> entry: hashMap.entrySet())
        {
            Double value=entry.getValue();
            HashMap<String,String> hashKey=entry.getKey();
            int dev=0;
            for(String key:hashKey.keySet())
               {
                dev=total_count.get(key);
               }
            entry.setValue(value/dev);
          //  System.out.println("Key: "+ entry.getKey()+ " Value: "+entry.getValue());
        }
        System.out.println("  TOATAL_TERM"+TOATAL_TERM);
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return hashMap;
}
```

2.2：计算文档 C 的先验概率：$\log \hat{P}(c)$

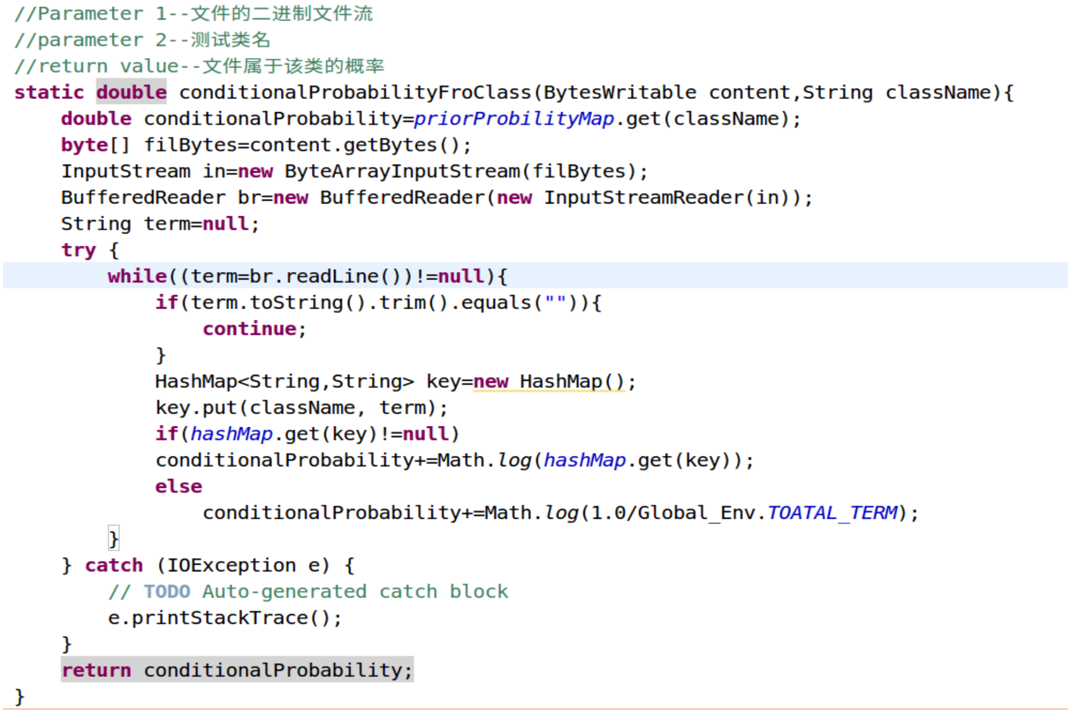
在 priorProbability.java 中实现 getPriorPrb()方法，该方法返回一个 HashMap<String,Double>,对应<类名,该类的先验概率>，其代码如下：



```java
//HashMap<类名,该类的先验概率>
public static HashMap<String,Double> getPriorPrb(){
    int count=0;
    HashMap<String,Double> hashmap=new HashMap();
    String path="hdfs://localhost:9000/user/hadoop/DocTypeNumOutput/part-r-00000";
    Configuration conf=new Configuration();
    try {
        FileSystem fs=FileSystem.get(URI.create(path),conf);
        InputStream in=null;
        in=fs.open(new Path(path));
        BufferedReader br=new BufferedReader(new InputStreamReader(in));
        String line=null;
        while((line=br.readLine())!=null){
            String[] split=line.split("\\s+");
            String key=split[0];
            Double value=Math.log(Double.parseDouble(split[1])/Global_Env.TOTAL_DOC);
            hashmap.put(key, value);
            count++;
        }
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return hashmap;
}
```

2.3：实现 conditionalProbabilityFroClass()函数

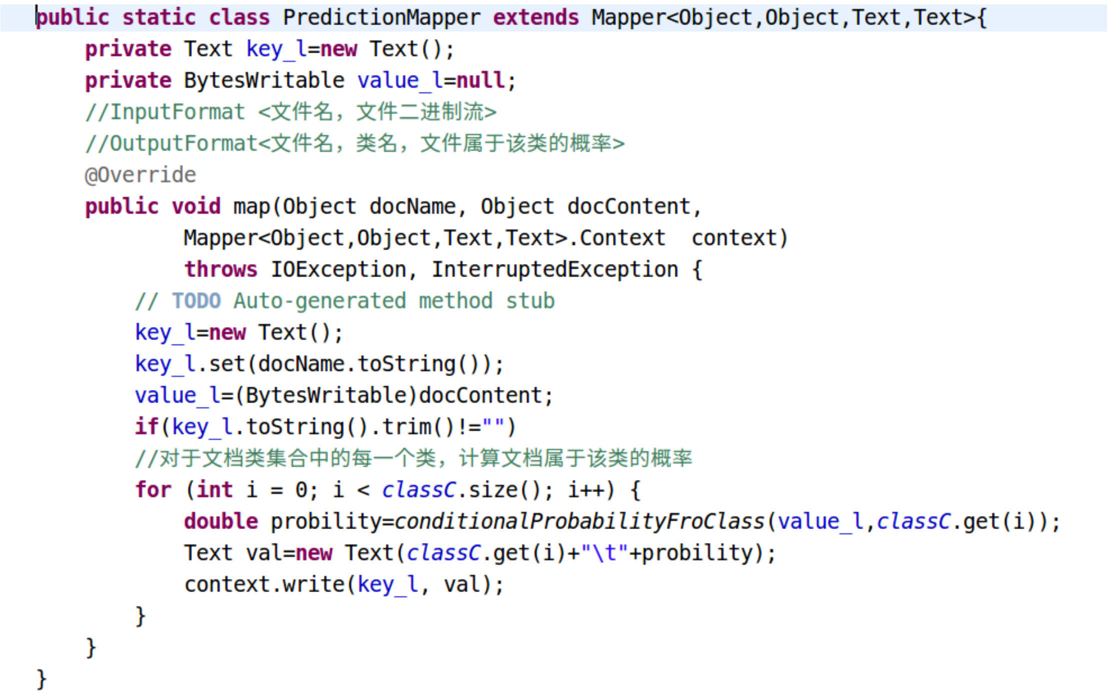
在 Prediction.java 中实现 conditionalProbabilityFroClass()函数。该函数的输入参数为一个文件的二进制文件流与类名,返回该文件属于该类的概率 P(class|doc) 。



```java
//Parameter 1--文件的二进制文件流
//parameter 2--测试类名
//return value--文件属于该类的概率
static double conditionalProbabilityFroClass(BytesWritable content,String className){
    double conditionalProbability=priorProbilityMap.get(className);
    byte[] filBytes=content.getBytes();
    InputStream in=new ByteArrayInputStream(filBytes);
    BufferedReader br=new BufferedReader(new InputStreamReader(in));
    String term=null;
    try {
        while((term=br.readLine())!=null){
            if(term.toString().trim().equals("")){
                continue;
            }
            HashMap<String,String> key=new HashMap();
            key.put(className, term);
            if(hashMap.get(key)!=null)
            conditionalProbability+=Math.log(hashMap.get(key));
            else
                conditionalProbability+=Math.log(1.0/Global_Env.TOATAL_TERM);
        }
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return conditionalProbability;
}
```

2.4：实现 PredictionMapper.class

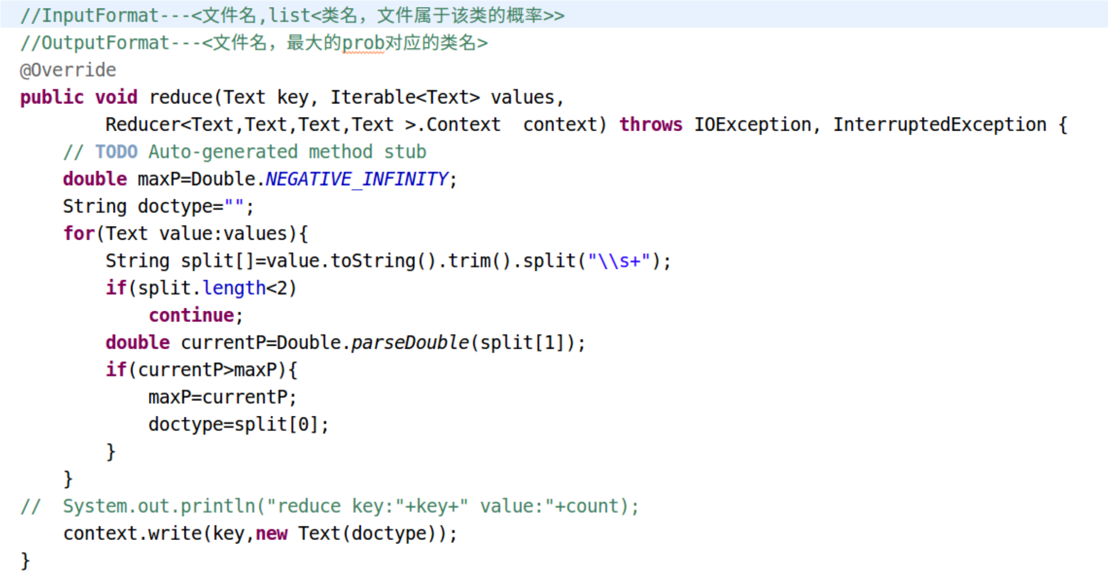
PredictionMapper 的输入为<文件名， 文件二进制流>，输出为<文件名,<类名,文件属于该类的概率>>，对于文档类集合中的每一个类，计算文档属于该类的概率，其实现如下。



```
public static class PredictionMapper extends Mapper<Object,Object,Text,Text>{
    private Text key_l=new Text();
    private BytesWritable value_l=null;
    //InputFormat <文件名，文件二进制流>
    //OutputFormat<文件名，类名，文件属于该类的概率>
    @Override
    public void map(Object docName, Object docContent,
            Mapper<Object,Object,Text,Text>.Context  context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        key_l=new Text();
        key_l.set(docName.toString());
        value_l=(BytesWritable)docContent;
        if(key_l.toString().trim()!="")
        //对于文档类集合中的每一个类，计算文档属于该类的概率
        for (int i = 0; i < classC.size(); i++) {
            double probility=conditionalProbabilityFroClass(value_l,classC.get(i));
            Text val=new Text(classC.get(i)+"\t"+probility);
            context.write(key_l, val);
        }
    }
}
```

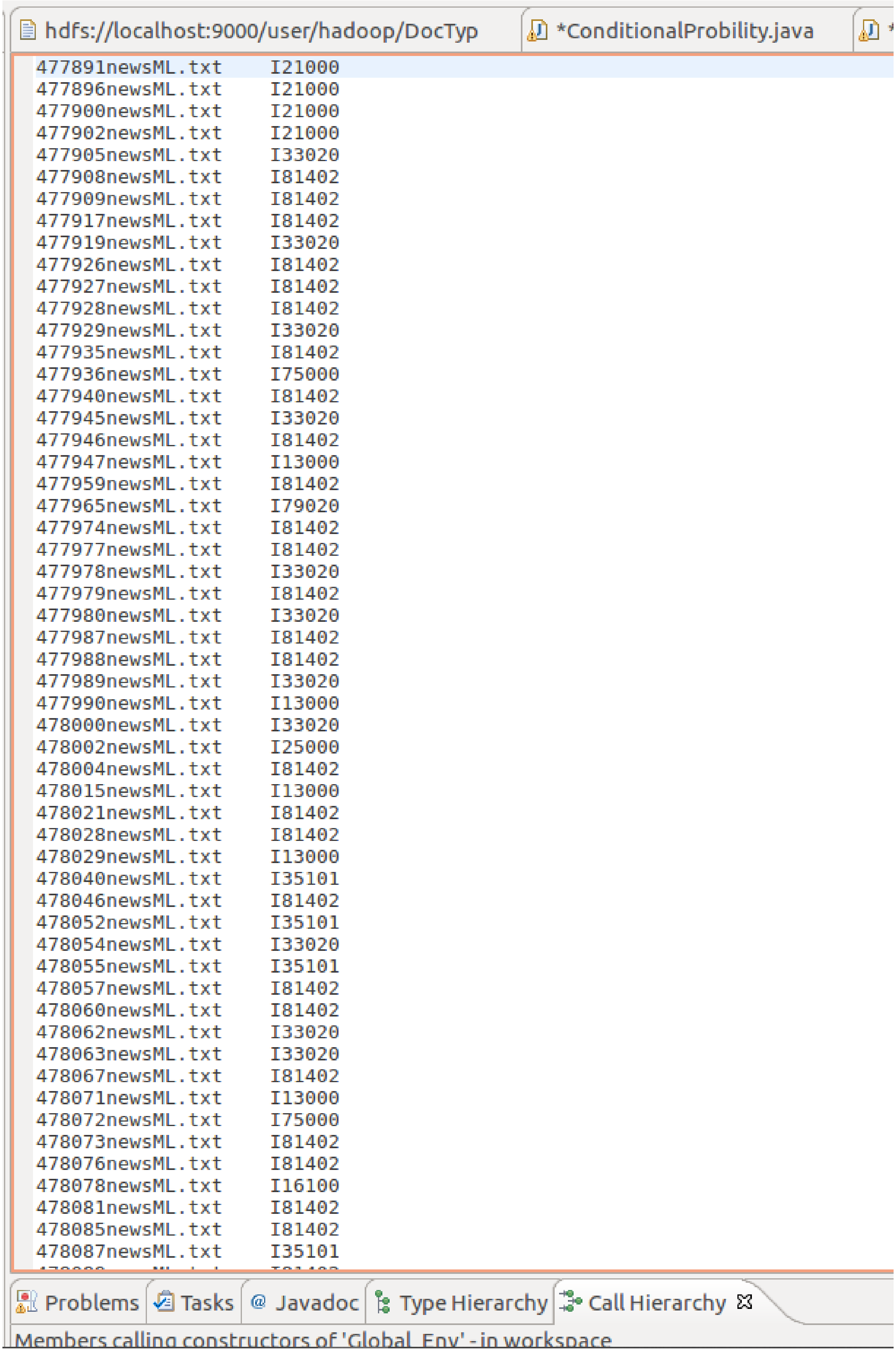
2.5：实现 PreductionReduce.class

PreductionReduce 中 reduce 的输入为<文件名,list<类名，文件属于该类的概率>>,对于每个文件,找到最大的 prob,输出<文件名,最大的 prob 对应的类名>。



```
//InputFormat---<文件名,list<类名，文件属于该类的概率>>
//OutputFormat---<文件名，最大的prob对应的类名>
@Override
public void reduce(Text key, Iterable<Text> values,
        Reducer<Text,Text,Text,Text >.Context  context) throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    double maxP=Double.NEGATIVE_INFINITY;
    String doctype="";
    for(Text value:values){
        String split[]=value.toString().trim().split("\\s+");
        if(split.length<2)
            continue;
        double currentP=Double.parseDouble(split[1]);
        if(currentP>maxP){
            maxP=currentP;
            doctype=split[0];
        }
    }
//  System.out.println("reduce key:"+key+" value:"+count);
    context.write(key,new Text(doctype));
}
```

2.6：测试文档的分类结果如下


hdfs://localhost:9000/user/hadoop/DocTyp
*ConditionalProbility.java
Problems
Tasks
Javadoc
Type Hierarchy
Call Hierarchy

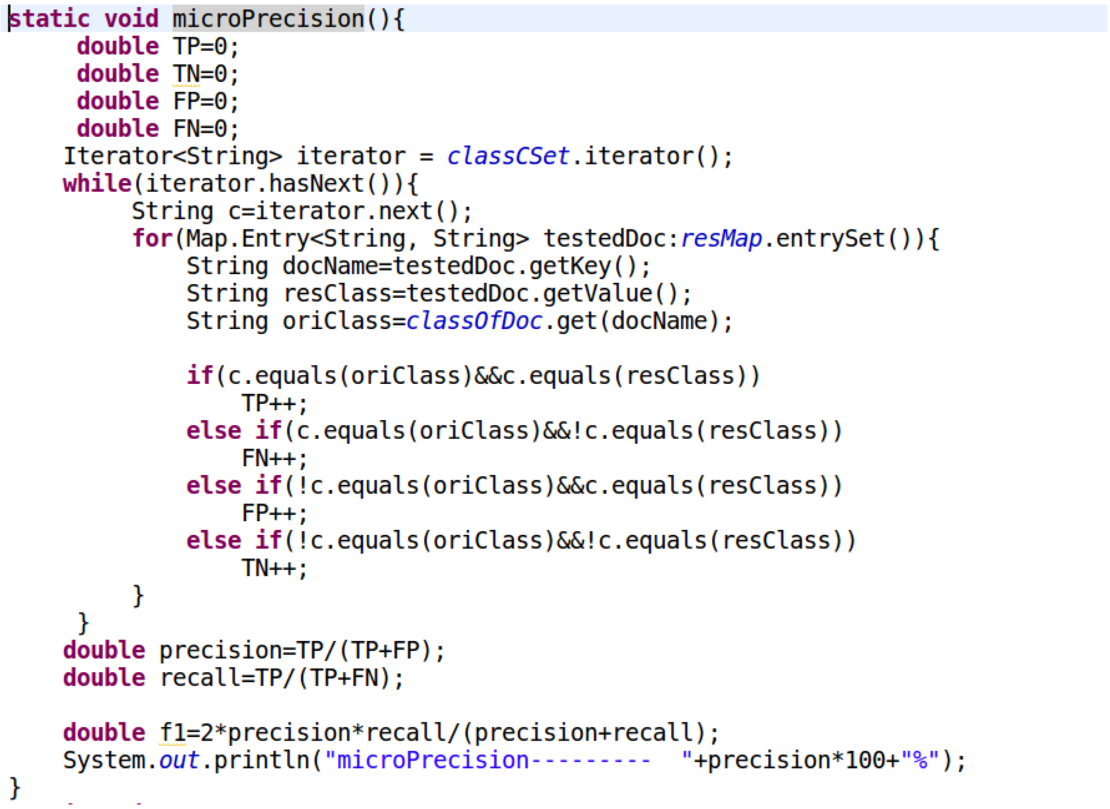
```
477891newsML.txt     I21000
477896newsML.txt     I21000
477900newsML.txt     I21000
477902newsML.txt     I21000
477905newsML.txt     I33020
477908newsML.txt     I81402
477909newsML.txt     I81402
477917newsML.txt     I81402
477919newsML.txt     I33020
477926newsML.txt     I81402
477927newsML.txt     I81402
477928newsML.txt     I81402
477929newsML.txt     I33020
477935newsML.txt     I81402
477936newsML.txt     I75000
477940newsML.txt     I81402
477945newsML.txt     I33020
477946newsML.txt     I81402
477947newsML.txt     I13000
477959newsML.txt     I81402
477965newsML.txt     I79020
477974newsML.txt     I81402
477977newsML.txt     I81402
477978newsML.txt     I33020
477979newsML.txt     I81402
477980newsML.txt     I33020
477987newsML.txt     I81402
477988newsML.txt     I81402
477989newsML.txt     I33020
477990newsML.txt     I13000
478000newsML.txt     I33020
478002newsML.txt     I25000
478004newsML.txt     I81402
478015newsML.txt     I13000
478021newsML.txt     I81402
478028newsML.txt     I81402
478029newsML.txt     I13000
478040newsML.txt     I35101
478046newsML.txt     I81402
478052newsML.txt     I35101
478054newsML.txt     I33020
478055newsML.txt     I35101
478057newsML.txt     I81402
478060newsML.txt     I81402
478062newsML.txt     I33020
478063newsML.txt     I33020
478067newsML.txt     I81402
478071newsML.txt     I13000
478072newsML.txt     I75000
478073newsML.txt     I81402
478076newsML.txt     I81402
478078newsML.txt     I16100
478081newsML.txt     I81402
478085newsML.txt     I81402
478087newsML.txt     I35101
```

Members calling constructors of 'Global_Env' - in workspace

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

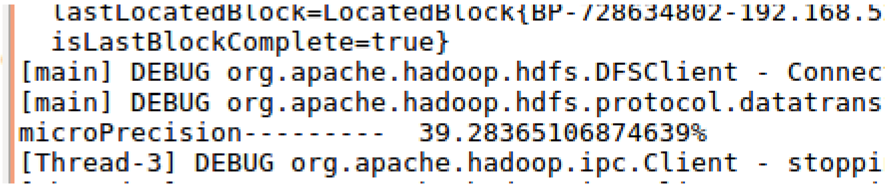
3.1：microPrecision

在 Judge.java 中实现 microPrecision()计算其 Micoraverage Precision



```
static void microPrecision(){
     double TP=0;
     double TN=0;
     double FP=0;
     double FN=0;
    Iterator<String> iterator = classCSet.iterator();
    while(iterator.hasNext()){
         String c=iterator.next();
         for(Map.Entry<String, String> testedDoc:resMap.entrySet()){
             String docName=testedDoc.getKey();
             String resClass=testedDoc.getValue();
             String oriClass=classOfDoc.get(docName);

             if(c.equals(oriClass)&&c.equals(resClass))
                 TP++;
             else if(c.equals(oriClass)&&!c.equals(resClass))
                 FN++;
             else if(!c.equals(oriClass)&&c.equals(resClass))
                 FP++;
             else if(!c.equals(oriClass)&&!c.equals(resClass))
                 TN++;
         }
     }
    double precision=TP/(TP+FP);
    double recall=TP/(TP+FN);

    double f1=2*precision*recall/(precision+recall);
    System.out.println("microPrecision---------  "+precision*100+"%");
}
```

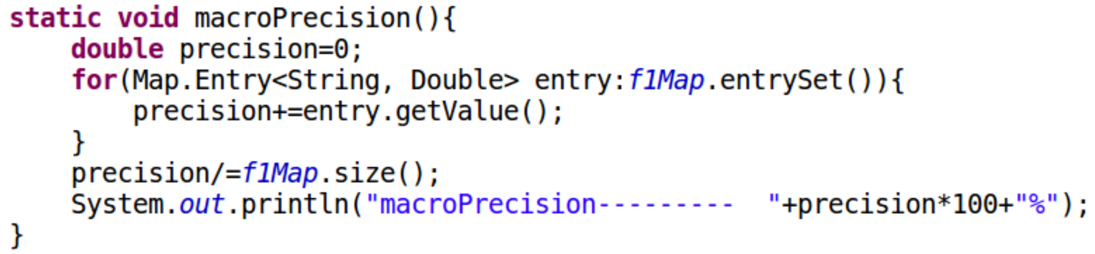
Micoraverage Precision 输出如图：



```
  lastLocatedBlock=LocatedBlock{BP-728634802-192.168.5
  isLastBlockComplete=true}
[main] DEBUG org.apache.hadoop.hdfs.DFSClient - Connec
[main] DEBUG org.apache.hadoop.hdfs.protocol.datatrans
microPrecision---------  39.28365106874639%
[Thread-3] DEBUG org.apache.hadoop.ipc.Client - stoppi
```

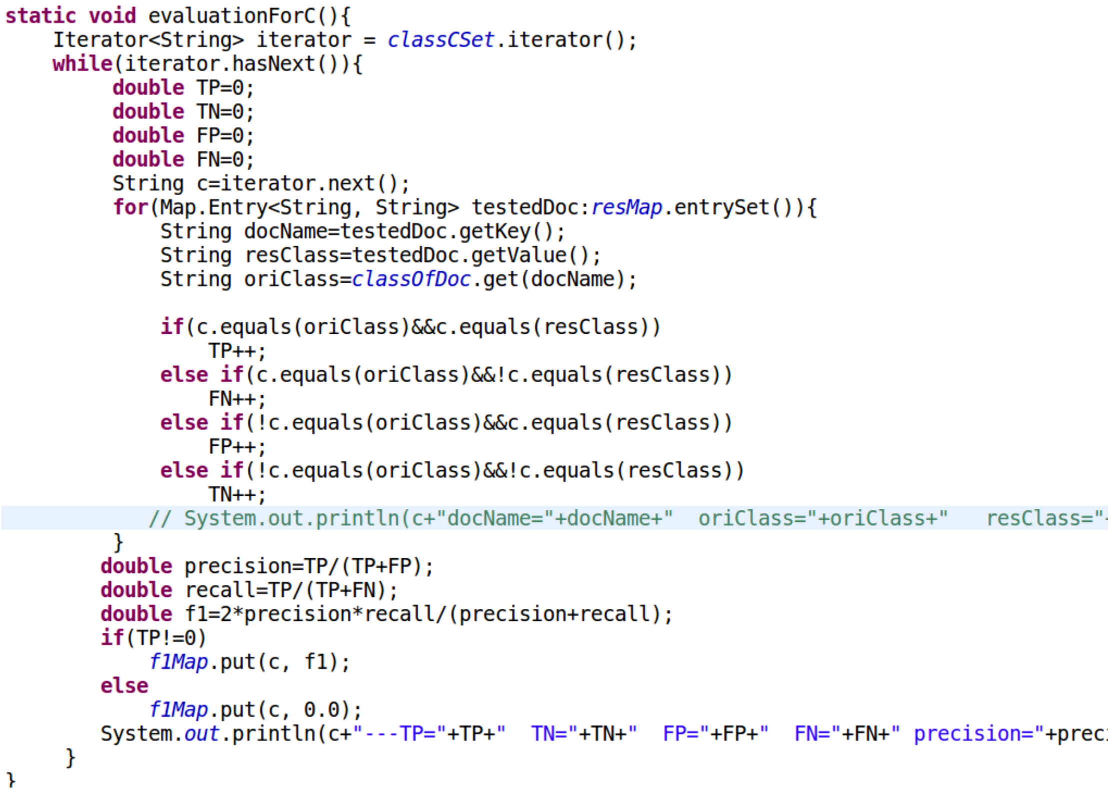
3.1：macroPrecision

在 Judge.java 中实现 macroPrecision()计算其 Macoraverage Precision



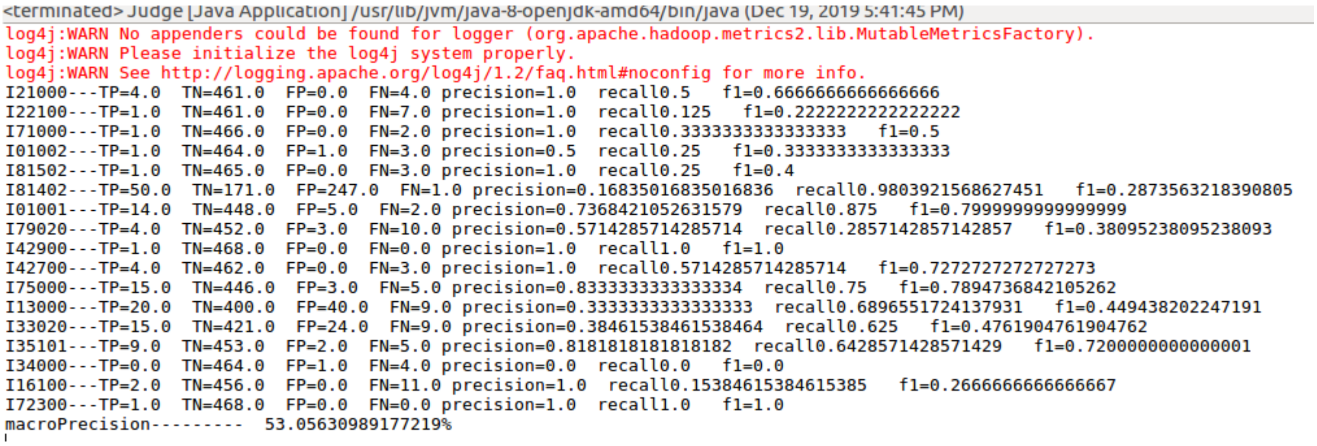
```
static void macroPrecision(){
    double precision=0;
    for(Map.Entry<String, Double> entry:f1Map.entrySet()){
        precision+=entry.getValue();
    }
    precision/=f1Map.size();
    System.out.println("macroPrecision---------  "+precision*100+"%");
}
```

其中 f1Map 为每个类所对应的 f1 值，其键值对结构为<类名，f1>，代码实现如下：



```
static void evaluationForC(){
    Iterator<String> iterator = classCSet.iterator();
    while(iterator.hasNext()){
         double TP=0;
         double TN=0;
         double FP=0;
         double FN=0;
         String c=iterator.next();
         for(Map.Entry<String, String> testedDoc:resMap.entrySet()){
             String docName=testedDoc.getKey();
             String resClass=testedDoc.getValue();
             String oriClass=classOfDoc.get(docName);

             if(c.equals(oriClass)&&c.equals(resClass))
                 TP++;
             else if(c.equals(oriClass)&&!c.equals(resClass))
                 FN++;
             else if(!c.equals(oriClass)&&c.equals(resClass))
                 FP++;
             else if(!c.equals(oriClass)&&!c.equals(resClass))
                 TN++;
            // System.out.println(c+"docName="+docName+"  oriClass="+oriClass+"   resClass="
         }
        double precision=TP/(TP+FP);
        double recall=TP/(TP+FN);
        double f1=2*precision*recall/(precision+recall);
        if(TP!=0)
            f1Map.put(c, f1);
        else
            f1Map.put(c, 0.0);
        System.out.println(c+"---TP="+TP+"  TN="+TN+"  FP="+FP+"  FN="+FN+" precision="+prec
    }
}
```

Macoraverage Precision 输出如图：



```
<terminated> Judge [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (Dec 19, 2019 5:41:45 PM)
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.lib.MutableMetricsFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
I21000---TP=4.0  TN=461.0  FP=0.0  FN=4.0 precision=1.0  recall0.5   f1=0.6666666666666666
I22100---TP=1.0  TN=461.0  FP=0.0  FN=7.0 precision=1.0  recall0.125   f1=0.2222222222222222
I71000---TP=1.0  TN=466.0  FP=0.0  FN=2.0 precision=1.0  recall0.3333333333333333   f1=0.5
I01002---TP=1.0  TN=464.0  FP=1.0  FN=3.0 precision=0.5  recall0.25   f1=0.3333333333333333
I81502---TP=1.0  TN=465.0  FP=0.0  FN=3.0 precision=1.0  recall0.25   f1=0.4
I81402---TP=50.0  TN=171.0  FP=247.0  FN=1.0 precision=0.16835016835016836  recall0.9803921568627451   f1=0.2873563218390805
I01001---TP=14.0  TN=448.0  FP=5.0  FN=2.0 precision=0.7368421052631579  recall0.875   f1=0.7999999999999999
I79020---TP=4.0  TN=452.0  FP=3.0  FN=10.0 precision=0.5714285714285714  recall0.2857142857142857   f1=0.38095238095238093
I42900---TP=1.0  TN=468.0  FP=0.0  FN=0.0 precision=1.0  recall1.0   f1=1.0
I42700---TP=4.0  TN=462.0  FP=0.0  FN=3.0 precision=1.0  recall0.5714285714285714   f1=0.7272727272727273
I75000---TP=15.0  TN=446.0  FP=3.0  FN=5.0 precision=0.8333333333333334  recall0.75   f1=0.7894736842105262
I13000---TP=20.0  TN=400.0  FP=40.0  FN=9.0 precision=0.3333333333333333  recall0.6896551724137931   f1=0.449438202247191
I33020---TP=15.0  TN=421.0  FP=24.0  FN=9.0 precision=0.38461538461538464  recall0.625   f1=0.4761904761904762
I35101---TP=9.0  TN=453.0  FP=2.0  FN=5.0 precision=0.8181818181818182  recall0.6428571428571429   f1=0.7200000000000001
I34000---TP=0.0  TN=464.0  FP=1.0  FN=4.0 precision=0.0  recall0.0   f1=0.0
I16100---TP=2.0  TN=456.0  FP=0.0  FN=11.0 precision=1.0  recall0.15384615384615385   f1=0.2666666666666667
I72300---TP=1.0  TN=468.0  FP=0.0  FN=0.0 precision=1.0  recall1.0   f1=1.0
macroPrecision---------  53.05630989177219%
```

以上为 project 的全部内容。

二 贝叶斯分类器理论介绍

首先假设已经有分好类的N篇文档:(d1,c1)、(d2,c2)、(d3,c3)……(dn,cn)。di 表示第 i 篇文档，ci 表示第 i 个类别。目标是：寻找一个分类器，这个分类器能够：当丢给它一篇新文档d，它就输出 d 最有可能属于哪个类别 c。

朴素贝叶斯分类器是一个概率分类器。假设现有的类别 C={c1,c2,……cm}。给定一篇文档d，这个问题用数学公式表示如下：

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d)$$

c^ 就是：在所有的类别 C={c1，c2，……cm} 中，使得：条件概率 P(c|d) 取最大值的类别。使用贝叶斯公式，将转换成如下形式：

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d) = \underset{c \in C}{\operatorname{argmax}} \frac{P(d|c)P(c)}{P(d)}$$

对类别 C 中的每个类型，计算 [p(d|c)*p(c)]/p(d) 的值，然后选取最大值对应的那个类型 ci ，该 ci 就是最优解 c^，因此，可以忽略掉分母 p(d)，变成如下形式：

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d) = \underset{c \in C}{\operatorname{argmax}} P(d|c)P(c)$$

这个公式由两部分组成,前面那部分 P(d|c) 称为似然函数,后面那部分 P(c) 称为先验概率。

文档 d，文档 d 的每个特征表示为：d={f1,f2,f3……fn}，那么这里的特征 fi 其实就是单词 wi 出现的频率（次数），公式转化成如下形式：

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \overbrace{P(f_1, f_2, \ldots., f_n|c)}^{\text{likelihood}} \overbrace{P(c)}^{\text{prior}}$$

对文档 d 做个假设：假设各个特征之间是相互独立的。那么 p(f1,f2……fn|c)=p(f1|c)*p(f2|c)*……*p(fn|c)，公式转化成如下形式：

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{f \in F} P(f|c)$$

由于每个概率值很小若干个很小的概率值直接相乘，得到的结果会越来越小。为了避免计算过程出现下溢，引入对数函数 Log，然后使用每个单词 wi 出现频率作为特征，得到如下公式：

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} \log P(c) + \sum_{i \in positions} \log P(w_i|c)$$

训练朴素贝叶斯的过程其实就是计算先验概率和似然函数的过程。

①先验概率 P(c)的计算

P(c)的意思是：在所有的文档中，类别为c的文档出现的概率有多大？假设训练数据中一共有Ndoc篇文档,只要数一下类别c的文档有多少个就能计算p(c)了，类别c的文档共有Nc篇，先验概率的计算公式如下：

$$\hat{P}(c)=\frac{N_c}{N_{doc}}$$

②似然函数P(wi|c)的计算

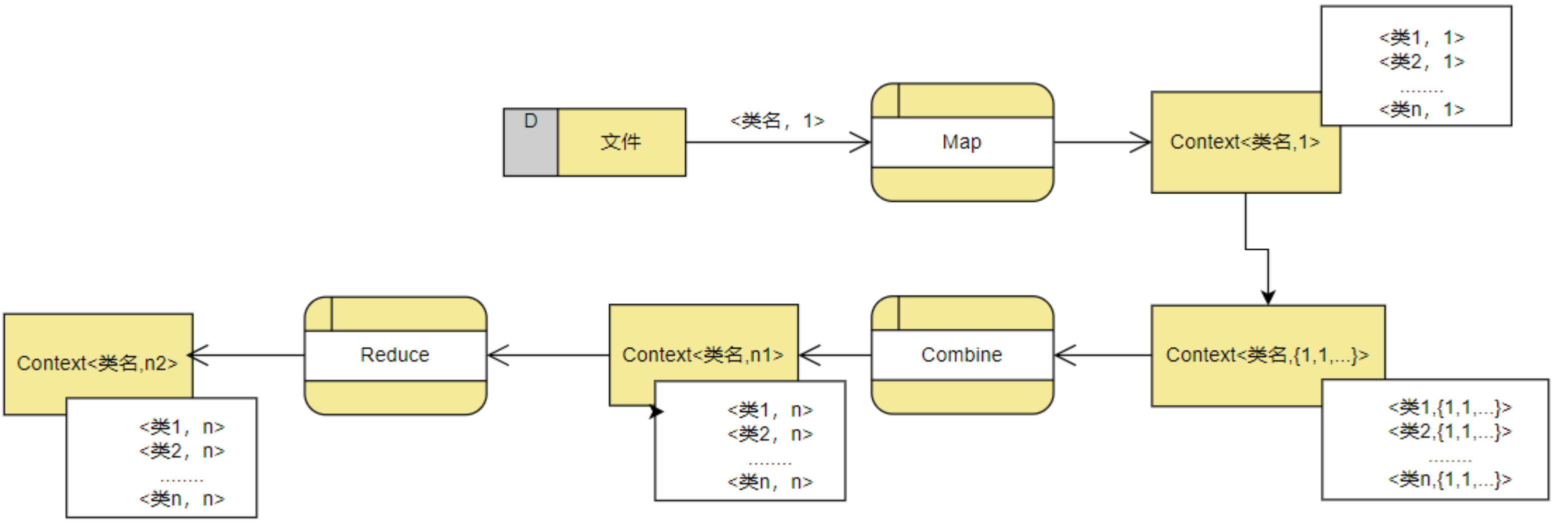
对于文档d中的每个单词wi，找到训练数据集中所有类别为c的文档，数一数单词wi在这些文档（类别为c）中出现的次数：count(wi,c)，然后再数一数训练数据集中类别为c的文档一共有多少个单词。计算 二者之间的比值，就是似然函数的值。似然函数计算公式如下：

$$\hat{P}(w_i|c)=\frac{count(w_i,c)}{\sum_{w\in V} count(w,c)}$$

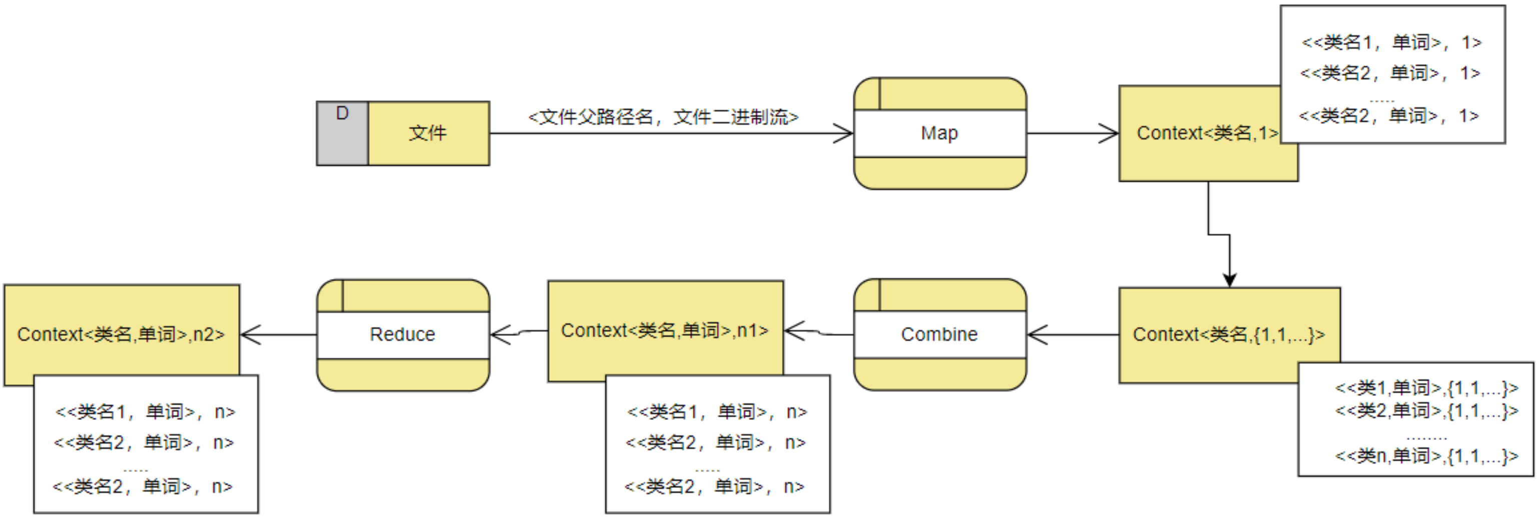
将“出现次数加1”。似然函数公式变成如下形式：

$$\hat{P}(w_i|c)=\frac{count(w_i,c)+1}{\sum_{w\in V}(count(w,c)+1)}=\frac{count(w_i,c)+1}{\left(\sum_{w\in V} count(w,c)\right)+|V|}$$

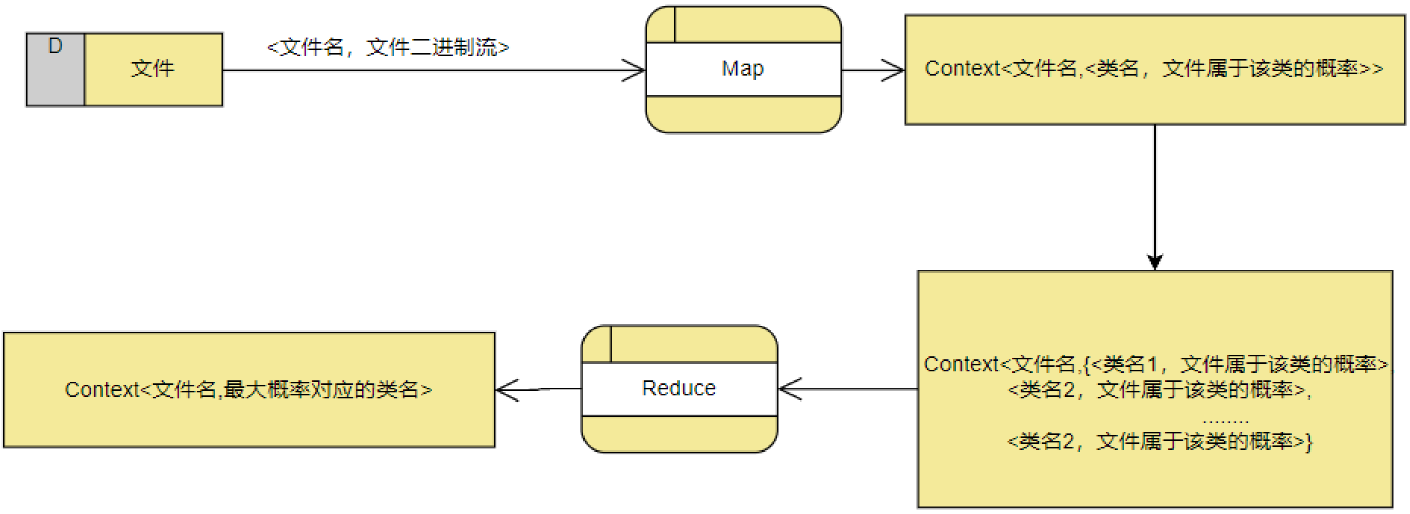
三 贝叶斯分类器训练的MapReduce算法设计



计算每个类型中文档数目的MapReduce job数据流图上如图所示，RecordReader输出的键值为<类名，1>。Map的输入为<类名，1>，Map中不做处理，直接输出<类名，1>。Reduce的输入为<类名，{文件数1，文件数2，...，文件数n}>，Reduce中对其进行求和，输出为<类名，文件总数>。



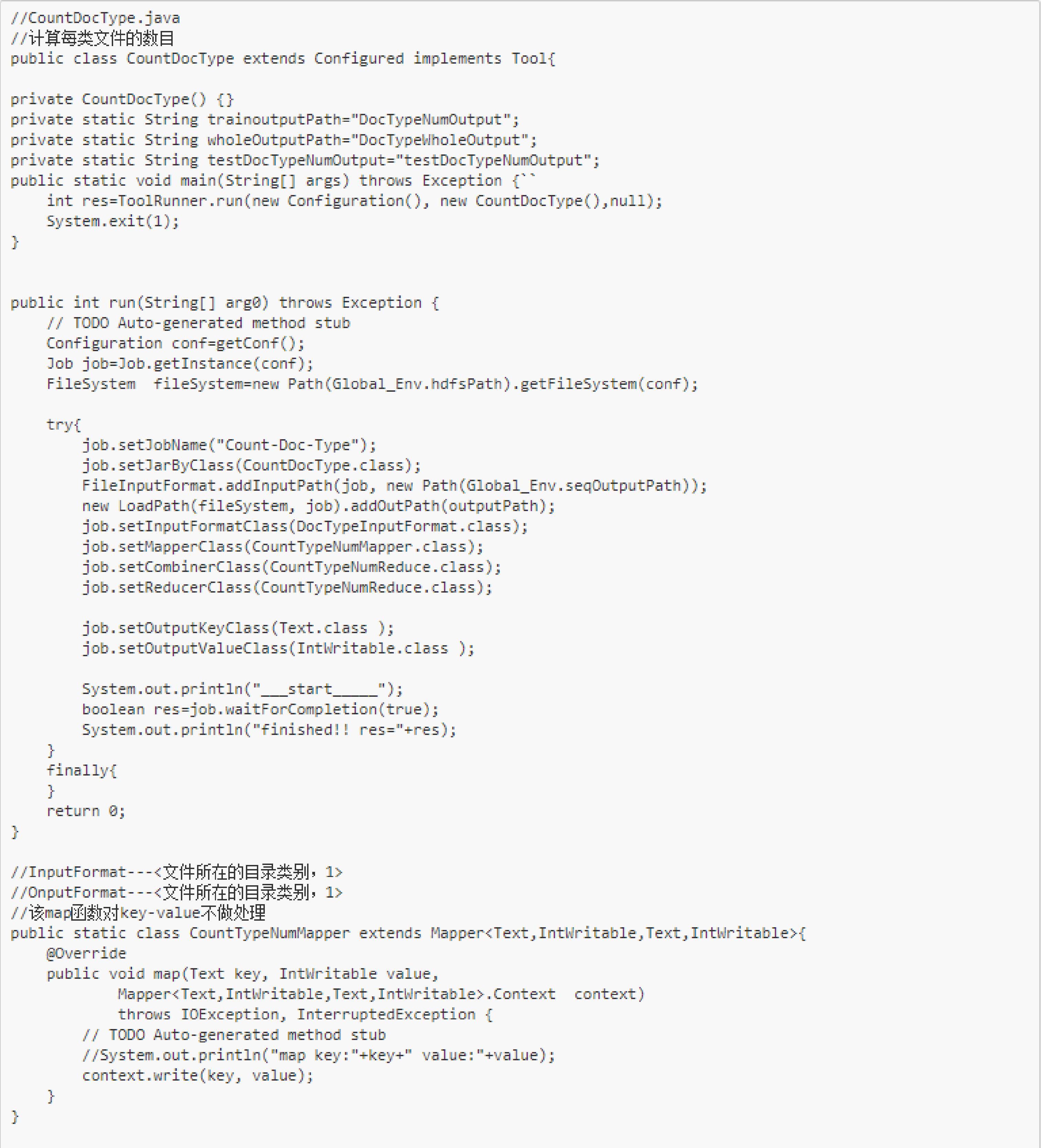
计算每个文件中每个单词出现频率的 MapReduce job 的数据流图如上图所示，使用 SequenceFile 作为输入，将每个文件作为一个 record 处理，Record Reader 输出的键值为<文件父路径名，文件二进制流>即<文件类名，文件二进制流>，即为 Map 的输入，map 的输出为<<类名,单词>,1>。Reduce 的输入为<<类名，单词>，{count1，count2，...，countn}>，Reduce 对其求和，输出为<<类名，单词>，单词数量>。


D
文件
<文件名，文件二进制流>
Map
Context<文件名,<类名，文件属于该类的概率>>
Context<文件名,{<类名1，文件属于该类的概率>
<类名2，文件属于该类的概率>,
....
<类名2，文件属于该类的概率>}
Reduce
Context<文件名,最大概率对应的类名>

预测 Map Reduce job 的数据流图如上图所示，使用 SequenceFile 作为输入，将每个文件作为一个 record 处理，Record Reader 输出的键值为<文件名，文件二进制流>,即为 Map 的输入,map 的输出为<文件名,<类名,文件属于该类的概率>>。Reduce 的输入为<文件名,<类名，list{文件属于该类的概率>}>，最终输出为<文件名,最大概率对应的类名>。

四：源代码清单

1：统计每类文件数目



```
//CountDocType.java
//计算每类文件的数目
public class CountDocType extends Configured implements Tool{

private CountDocType() {}
private static String trainoutputPath="DocTypeNumOutput";
private static String wholeOutputPath="DocTypeWholeOutput";
private static String testDocTypeNumOutput="testDocTypeNumOutput";
public static void main(String[] args) throws Exception {``
    int res=ToolRunner.run(new Configuration(), new CountDocType(),null);
    System.exit(1);
}


public int run(String[] arg0) throws Exception {
    // TODO Auto-generated method stub
    Configuration conf=getConf();
    Job job=Job.getInstance(conf);
    FileSystem  fileSystem=new Path(Global_Env.hdfsPath).getFileSystem(conf);

    try{
        job.setJobName("Count-Doc-Type");
        job.setJarByClass(CountDocType.class);
        FileInputFormat.addInputPath(job, new Path(Global_Env.seqOutputPath));
        new LoadPath(fileSystem, job).addOutPath(outputPath);
        job.setInputFormatClass(DocTypeInputFormat.class);
        job.setMapperClass(CountTypeNumMapper.class);
        job.setCombinerClass(CountTypeNumReduce.class);
        job.setReducerClass(CountTypeNumReduce.class);

        job.setOutputKeyClass(Text.class );
        job.setOutputValueClass(IntWritable.class );

        System.out.println("___start_____");
        boolean res=job.waitForCompletion(true);
        System.out.println("finished!! res="+res);
    }
    finally{
    }
    return 0;
}

//InputFormat---<文件所在的目录类别，1>
//OnputFormat---<文件所在的目录类别，1>
//该map函数对key-value不做处理
public static class CountTypeNumMapper extends Mapper<Text,IntWritable,Text,IntWritable>{
    @Override
    public void map(Text key, IntWritable value,
            Mapper<Text,IntWritable,Text,IntWritable>.Context  context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        //System.out.println("map key:"+key+" value:"+value);
        context.write(key, value);
    }
}
//InputFormat---<文件所在的目录类别，1>
//OnputFormat---<文件所在的目录类别，该类别下文件总数>
public static class CountTypeNumReduce extends Reducer<Text,IntWritable,Text,IntWritable>{
    @Override
    public void reduce(Text key, Iterable<IntWritable> values,
            Reducer<Text,IntWritable,Text,IntWritable >.Context  context) throws IOException, InterruptedException
        // TODO Auto-generated method stub
        int count=0;
        for(IntWritable value:values){
            count+=value.get();
        }
    //  System.out.println("reduce key:"+key+" value:"+count);
        context.write(key, new IntWritable(count));
    }



}

//读取输出文件，统计文件类型,返回list<文件类型>
public static List readDocType(){
    List<String> list=new ArrayList();
     Configuration conf = new Configuration();
    String path = trainoutputPath+"/part-r-00000";
    FileSystem fs;
    int totalNum=0;
    try {
        fs = FileSystem.get(URI.create(path),conf);
        InputStream in=null;
        in=fs.open(new Path(path));
        BufferedReader br=new BufferedReader(new InputStreamReader(in));
        String line=null;
        while((line=br.readLine())!=null){
            String[] split=line.split("\\s+");
            String docType=split[0];
            String docNum=split[1];
            totalNum+=Integer.parseInt(docNum);
            list.add(docType);
        }
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    System.out.println("total doc number="+totalNum);
    return list;
}
}
```

```
//DocTypeInputFormat.java
public class DocTypeInputFormat extends FileInputFormat<Text, IntWritable>{
    @Override
    public RecordReader<Text,IntWritable> createRecordReader(InputSplit split,
            TaskAttemptContext context) throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        DocTypeRecordReader reader=new DocTypeRecordReader();
        reader.initialize(split, context);
        return reader;
    }

    @Override
    protected boolean isSplitable(JobContext context, Path filename) {
        // TODO Auto-generated method stub
        return false;
    }
}
```

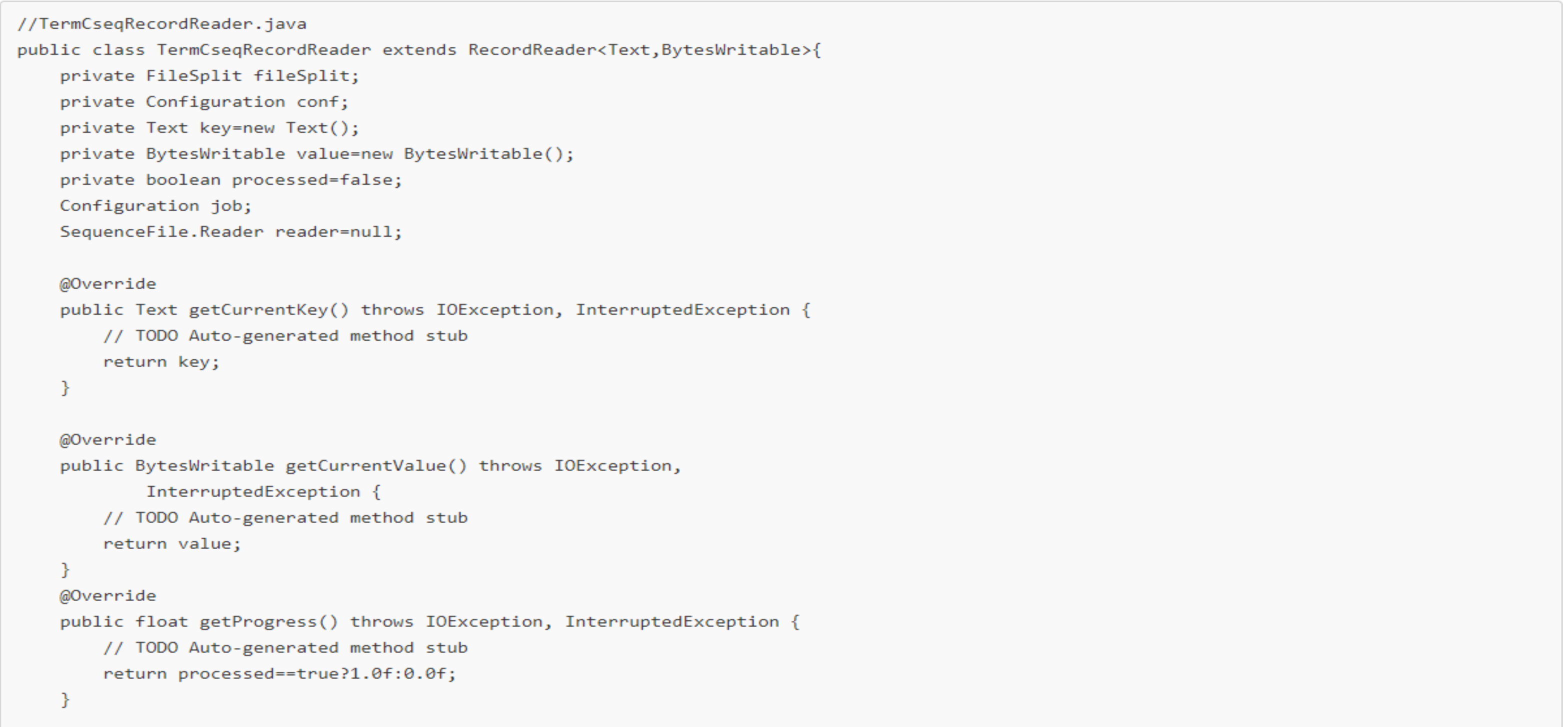
```java
//DocTypeRecordReader.java
public class DocTypeRecordReader extends RecordReader<Text, IntWritable>{
    private IntWritable value=new IntWritable(0);
    private Text key=new Text();
    private FileSplit fileSplit;
    private Configuration conf;
    private boolean processed=false;
    SequenceFile.Reader reader=null;

    @Override
    public Text getCurrentKey() throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        return key;
    }

    @Override
    public IntWritable getCurrentValue() throws IOException,
            InterruptedException {
        // TODO Auto-generated method stub
        return value;
    }

    @Override
    public float getProgress() throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        return processed?1.0f:0.0f;
    }

    @Override
    public void initialize(InputSplit split, TaskAttemptContext context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        this.fileSplit=(FileSplit)split;
        this.conf=context.getConfiguration();
        this.reader = new SequenceFile.Reader(fileSplit.getPath().getFileSystem(conf), fileSplit.getPath(), conf);
    }

    @Override
    public boolean nextKeyValue() throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        if(!processed){
            Writable key_s = (Writable) ReflectionUtils.newInstance(reader.getKeyClass(), conf);
            Writable value_s = (Writable) ReflectionUtils.newInstance(reader.getValueClass(), conf);
            boolean unFinished=reader.next(key_s, value_s);

            if(key_s.toString().trim().equals("")){
                processed=true;
                return true;
            }
            //返回文档父路径即类名
            key.set(new Text((new Path(key_s.toString())).getParent().toString()));
            value=new IntWritable(1);
            if(!unFinished){
                processed=true;
            }
        }
        return !processed;
    }

}
```

2：统计类中单词频率

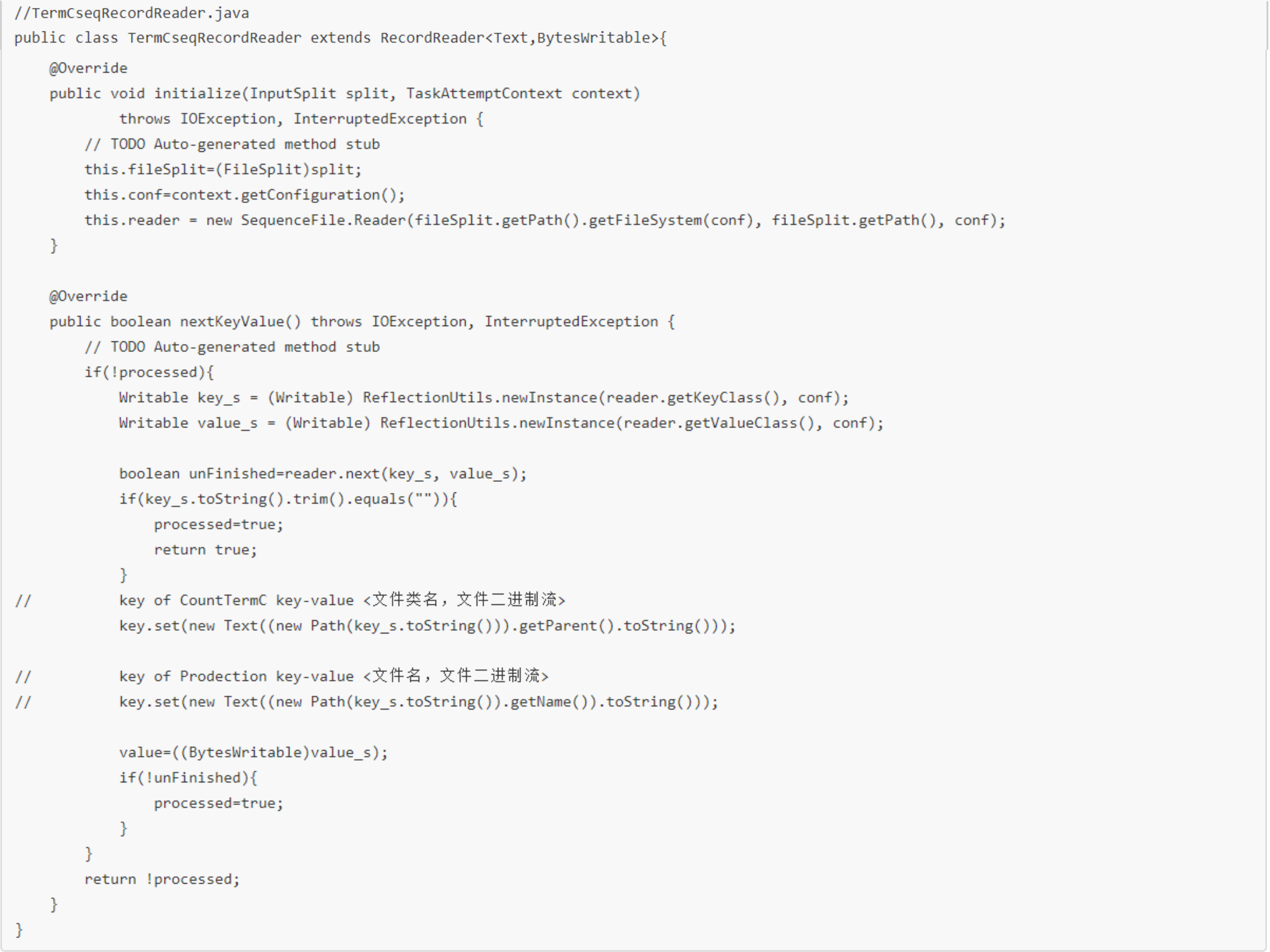
```
//统计类中单词频率
//CountTermC.java
public class CountTermC extends Configured implements Tool{
    private static String outputPath="TermCNumOutput";

    public static void main(String[] args) throws Exception {
        int res=ToolRunner.run(new Configuration(), new CountTermC(),null);
        System.exit(1);
    }

    //统计测试集合内所有单词数
    public static int countTermNumber(){
        String path="hdfs://localhost:9000/user/hadoop/"+outputPath+"/part-r-00000";
        int count=0;
        Configuration conf=new Configuration();
        try {
            FileSystem fs=FileSystem.get(URI.create(path),conf);
            InputStream in=null;
            in=fs.open(new Path(path));
            BufferedReader br=new BufferedReader(new InputStreamReader(in));
            String line=null;
            while((line=br.readLine())!=null){
                String[] splited = line.split("\\s+");
                int termNum=Integer.parseInt(splited[2]);
                count+=termNum;
            }
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        System.out.println("Term num="+count);
        return count;
    }

    public int run(String[] arg0) throws Exception {
        // TODO Auto-generated method stub
        Configuration conf=getConf();
        Job job=Job.getInstance(conf);
        FileSystem  fileSystem=new Path(Global_Env.hdfsPath).getFileSystem(conf);

        try{
            job.setJobName("Count-TermC-Type");
            job.setJarByClass(CountTermC.class);

            FileInputFormat.addInputPath(job, new Path(Global_Env.seqOutputPath));
            new LoadPath(fileSystem, job).addOutPath(outputPath);

            job.setInputFormatClass(TermCNumInputFromat.class);

            job.setMapperClass(CountTermC.CountTermcSeqNumMapper.class);
            job.setCombinerClass(CountTermC.CountTermCNumReduce.class);
            job.setReducerClass(CountTermC.CountTermCNumReduce.class);

            job.setOutputKeyClass(Text.class );
            job.setOutputValueClass(IntWritable.class );

            System.out.println("___start_____");
            boolean res=job.waitForCompletion(true);
            System.out.println("finished!! res="+res);


        }
        finally{

        }
        return 0;
    }
    //Mapper
    //输入<文件父路径名，文件二进制流>
    //输出<类名-单词，IntWritable(1)>
    public static class CountTermcSeqNumMapper extends Mapper<Object,Object,Text,IntWritable>{
        private Text key_l=new Text();
        private BytesWritable value_l=null;

        //InputFormat <className,binary file stream>
        //OutputFormat<<classNmae,word>,count>
        @Override
        public void map(Object line, Object value,
                Mapper<Object,Object,Text,IntWritable>.Context  context)
                throws IOException, InterruptedException {
            // TODO Auto-generated method stub
            key_l=new Text();
            key_l.set(line.toString());
            value_l=(BytesWritable)value;
            //read each line of file write key-value
            writeKVInLine(context);
        }

        private void writeKVInLine(Mapper<Object,Object,Text,IntWritable>.Context  context) throws IOException, InterruptedException {
            // TODO Auto-generated method stub
            byte[] filebytes=value_l.getBytes();
            InputStream in = new ByteArrayInputStream(filebytes);
            BufferedReader brIn=new BufferedReader(new InputStreamReader(in));

            String key_word=null;

            while((key_word=brIn.readLine())!=null){
                if(key_word.toString().trim().equals(""))
                    continue;
                Text key_w=new Text(key_l.toString()+"\t"+new String(key_word));
                context.write(key_w, new IntWritable(1));
            }
        }
    }
//Reduce
//输入<<类名，单词>，{count1，count2，...,countn}>
//输出<<类名，单词>，单词数量>
public static class CountTermCNumReduce extends Reducer<Text,IntWritable,Text,IntWritable>{
        @Override
        public void reduce(Text key, Iterable<IntWritable> values,
                Reducer<Text,IntWritable,Text,IntWritable >.Context  context) throws IOException, InterruptedException {
            // TODO Auto-generated method stub
            int count=0;
            for(IntWritable value:values){
                count+=value.get();
            }
        //  System.out.println("reduce key:"+key+" value:"+count);
            context.write(key, new IntWritable(count));
        }



    }
}
```

```
//TermCNumInputFromat.java
public class TermCNumInputFromat extends FileInputFormat{
    @Override
    public RecordReader createRecordReader(InputSplit split,
            TaskAttemptContext context) throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        //System.out.println("createRecordReader"+split.toString());
        TermCseqRecordReader reader=new TermCseqRecordReader();
        reader.initialize(split, context);
        return reader;
    }
}
```

3：预测文件所对应的最大概率的类



```java
//TermCseqRecordReader.java
public class TermCseqRecordReader extends RecordReader<Text,BytesWritable>{
    private FileSplit fileSplit;
    private Configuration conf;
    private Text key=new Text();
    private BytesWritable value=new BytesWritable();
    private boolean processed=false;
    Configuration job;
    SequenceFile.Reader reader=null;

    @Override
    public Text getCurrentKey() throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        return key;
    }

    @Override
    public BytesWritable getCurrentValue() throws IOException,
            InterruptedException {
        // TODO Auto-generated method stub
        return value;
    }
    @Override
    public float getProgress() throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        return processed==true?1.0f:0.0f;
    }
```



```java
//TermCseqRecordReader.java
public class TermCseqRecordReader extends RecordReader<Text,BytesWritable>{

    @Override
    public void initialize(InputSplit split, TaskAttemptContext context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        this.fileSplit=(FileSplit)split;
        this.conf=context.getConfiguration();
        this.reader = new SequenceFile.Reader(fileSplit.getPath().getFileSystem(conf), fileSplit.getPath(), conf);
    }

    @Override
    public boolean nextKeyValue() throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        if(!processed){
            Writable key_s = (Writable) ReflectionUtils.newInstance(reader.getKeyClass(), conf);
            Writable value_s = (Writable) ReflectionUtils.newInstance(reader.getValueClass(), conf);

            boolean unFinished=reader.next(key_s, value_s);
            if(key_s.toString().trim().equals("")){
                processed=true;
                return true;
            }
//          key of CountTermC key-value <文件类名，文件二进制流>
            key.set(new Text((new Path(key_s.toString())).getParent().toString()));

//          key of Prodection key-value <文件名，文件二进制流>
//          key.set(new Text((new Path(key_s.toString()).getName()).toString()));

            value=((BytesWritable)value_s);
            if(!unFinished){
                processed=true;
            }
        }
        return !processed;
    }
}
```

```
//Prediction.java
public class Prediction extends Configured implements Tool{
    static HashMap<HashMap<String,String>,Double> hashMap=ConditionalProbility.getT_C();
    static HashMap<String,Double> priorProbilityMap=priorProbability.getPriorPrb();
    static List<String> classC=CountDocType.readDocType();


    public static void main(String[] args) {
        try {
            int res=ToolRunner.run(new Configuration(), new Prediction(),null);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        System.exit(1);
    }
    //Parameter 1--文件的二进制文件流
    //parameter 2--测试类名
    //return value--文件属于该类的概率
    static double conditionalProbabilityFroClass(BytesWritable content,String className){
        double conditionalProbability=priorProbilityMap.get(className);
        byte[] filBytes=content.getBytes();
        InputStream in=new ByteArrayInputStream(filBytes);
        BufferedReader br=new BufferedReader(new InputStreamReader(in));
        String term=null;
        try {
            while((term=br.readLine())!=null){
                if(term.toString().trim().equals("")){
                    continue;
                }
                HashMap<String,String> key=new HashMap();
                key.put(className, term);
                if(hashMap.get(key)!=null)
                conditionalProbability+=Math.log(hashMap.get(key));
                else
                    conditionalProbability+=Math.log(1.0/Global_Env.TOATAL_TERM);
            }
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return conditionalProbability;
    }
    public int run(String[] arg0) throws Exception {
        // TODO Auto-generated method stub
        Configuration conf=getConf();
        Job job=Job.getInstance(conf);
        FileSystem fileSystem=new Path(Global_Env.hdfsPath).getFileSystem(conf);

        job.setJobName("Prediction");
        job.setJarByClass(Prediction.class);

        FileInputFormat.addInputPath(job, new Path(Global_Env.seqTestOutputPath));
        new LoadPath(fileSystem, job).addOutPath(Global_Env.testResultPath);

        job.setInputFormatClass(TermCNumInputFromat.class);

        job.setMapperClass(Prediction.PredictionMapper.class );
        job.setReducerClass(Prediction.PreductionReduce.class);

        job.setOutputKeyClass(Text.class );
        job.setOutputValueClass(Text.class );

        System.out.println("___start_____");
        boolean res=job.waitForCompletion(true);
        System.out.println("finished!! res="+res);
        return 0;
    }
    public static class PredictionMapper extends Mapper<Object,Object,Text,Text>{
        private Text key_l=new Text();
        private BytesWritable value_l=null;
        //InputFormat <文件名，文件二进制流>
        //OutputFormat<文件名,<类名，文件属于该类的概率>>
        @Override
        public void map(Object docName, Object docContent,
                Mapper<Object,Object,Text,Text>.Context  context)
                throws IOException, InterruptedException {
            // TODO Auto-generated method stub
            key_l=new Text();
            key_l.set(docName.toString());
            value_l=(BytesWritable)docContent;
            if(key_l.toString().trim()!="")
            //对于文档类集合中的每一个类，计算文档属于该类的概率
            for (int i = 0; i < classC.size(); i++) {
                double probility=conditionalProbabilityFroClass(value_l,classC.get(i));
                Text val=new Text(classC.get(i)+"\t"+probility);
                context.write(key_l, val);
            }
        }
    }
    public static class PreductionReduce extends Reducer<Text,Text,Text,Text>{
            //InputFormat---<文件名,list<类名，文件属于该类的概率>>
            //OutputFormat---<文件名，最大的prob对应的类名>
            @Override
            public void reduce(Text key, Iterable<Text> values,
                    Reducer<Text,Text,Text,Text >.Context  context) throws IOException, InterruptedException {
                // TODO Auto-generated method stub
                double maxP=Double.NEGATIVE_INFINITY;
                String doctype="";
                for(Text value:values){
                    String split[]=value.toString().trim().split("\\s+");
                    if(split.length<2)
                        continue;
                    double currentP=Double.parseDouble(split[1]);
                    if(currentP>maxP){
                        maxP=currentP;
                        doctype=split[0];
                    }
                }
            //  System.out.println("reduce key:"+key+" value:"+count);
                context.write(key,new Text(doctype));
            }
        }
}
```

```
//ConditionalProbility.java
public class ConditionalProbility {
    public static void main(String[] args) {
        getT_C();
    }

    //get the prior Conditional Probility of docC--term
    //HaspMap<<类名，单词>，概率>
    public static HashMap<HashMap<String,String>,Double>  getT_C(){
        int TOATAL_TERM=0;
        HashMap<HashMap<String,String>,Double> hashMap=new HashMap();
        HashMap<String,Integer> total_count=new HashMap();
        String path="hdfs://localhost:9000/user/hadoop/TermCNumOutput/part-r-00000";
        Configuration conf=new Configuration();
        try {
            FileSystem fs=FileSystem.get(URI.create(path),conf);
            InputStream in=null;
            in=fs.open(new Path(path));
            BufferedReader br=new BufferedReader(new InputStreamReader(in));
            String line=null;
            String curDocType=null;
            //totalWordOfC record the word number of the current classC
            int totalWordOfC=0;
            while((line=br.readLine())!=null){
                totalWordOfC++;
                String[] split=line.split("\\s+");
                String docType=split[0];
                String term=split[1];
                Double value=Double.parseDouble((split[2]))+1;
                HashMap<String,String> key=new HashMap();
                key.put(docType, term);
                hashMap.put(key,value);
                if(curDocType==null){
                    curDocType=docType;
                }
                if(!curDocType.equals(docType)){
                    System.out.println("curDocType:"+curDocType+" doctype  "+docType+" "+totalWordOfC);
                    total_count.put(curDocType, totalWordOfC+Global_Env.TOATAL_TERM);
                    TOATAL_TERM+=totalWordOfC;
                    curDocType=docType;
                    totalWordOfC=0;
                }
            }
            total_count.put(curDocType, totalWordOfC+Global_Env.TOATAL_TERM);
            for(Map.Entry<HashMap<String, String>, Double> entry: hashMap.entrySet())
            {
                 Double value=entry.getValue();
                 HashMap<String,String> hashKey=entry.getKey();
                 int dev=0;
                 for(String key:hashKey.keySet())
                {
                 dev=total_count.get(key);
                }
                 entry.setValue(value/dev);
            }
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return hashMap;
    }
}
```

```
//priorProbability.java
public class priorProbability {
    public static void main(String[] args) {
        getPriorPrb();
    }

    //HashMap<类名,该类的先验概率>
    public static HashMap<String,Double> getPriorPrb(){
        int count=0;
        HashMap<String,Double> hashmap=new HashMap();
        String path="hdfs://localhost:9000/user/hadoop/DocTypeNumOutput/part-r-00000";
        Configuration conf=new Configuration();
        try {
            FileSystem fs=FileSystem.get(URI.create(path),conf);
            InputStream in=null;
            in=fs.open(new Path(path));
            BufferedReader br=new BufferedReader(new InputStreamReader(in));
            String line=null;
            while((line=br.readLine())!=null){
                String[] split=line.split("\\s+");
                String key=split[0];
                Double value=Math.log(Double.parseDouble(split[1])/Global_Env.TOTAL_DOC);
                hashmap.put(key, value);
                count++;
            }
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return hashmap;
    }
}
```

```java
//Judge.java
public class Judge {
    static HashMap<String,String>  classOfDoc=new HashMap<String, String>();
    static HashMap<String,String>  resMap=new HashMap<String, String>();
    static HashSet<String> classCSet=new HashSet<String>();

    static  HashMap<String,Double> precisionMap=new HashMap<String,Double>();
    static HashMap<String,Double> recallMap=new HashMap<String,Double>();
    static HashMap<String,Double> f1Map=new HashMap<String,Double>();

    public static void main(String[] args) {
        readClassOfDoc();
        readResult();
        evaluationForC();
        macroPrecision();
        microPrecision();
    }
    //读取文件的真实类型
    static void readClassOfDoc(){
       Configuration conf = new Configuration();
       String path = "docCountOutput/part-r-00000";
       FileSystem fs;
        try {
            fs = FileSystem.get(URI.create(path),conf);
            InputStream in=null;
            in=fs.open(new Path(path));
            BufferedReader br=new BufferedReader(new InputStreamReader(in));
            String line=null;
            while((line=br.readLine())!=null){
                String[] split=line.split("\\s+");
                String classC=split[0];
                String doc=split[1];
                classOfDoc.put(doc,classC);
            }
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    static void evaluationForC(){
        Iterator<String> iterator = classCSet.iterator();
        while(iterator.hasNext()){
             double TP=0;
             double TN=0;
             double FP=0;
             double FN=0;
             String c=iterator.next();
             for(Map.Entry<String, String> testedDoc:resMap.entrySet()){
                 String docName=testedDoc.getKey();
                 String resClass=testedDoc.getValue();
                 String oriClass=classOfDoc.get(docName);

                 if(c.equals(oriClass)&&c.equals(resClass))
                     TP++;
                 else if(c.equals(oriClass)&&!c.equals(resClass))
                     FN++;
                 else if(!c.equals(oriClass)&&c.equals(resClass))
                     FP++;
                 else if(!c.equals(oriClass)&&!c.equals(resClass))
                     TN++;
             }
            double precision=TP/(TP+FP);
            double recall=TP/(TP+FN);
            double f1=2*precision*recall/(precision+recall);
            if(TP!=0)
                f1Map.put(c, f1);
            else
                f1Map.put(c, 0.0);
         }
    }
    static void macroPrecision(){
        double precision=0;
        for(Map.Entry<String, Double> entry:f1Map.entrySet()){
            precision+=entry.getValue();
        }
        precision/=f1Map.size();
        System.out.println("macroPrecision---------  "+precision*100+"%");
    }
```

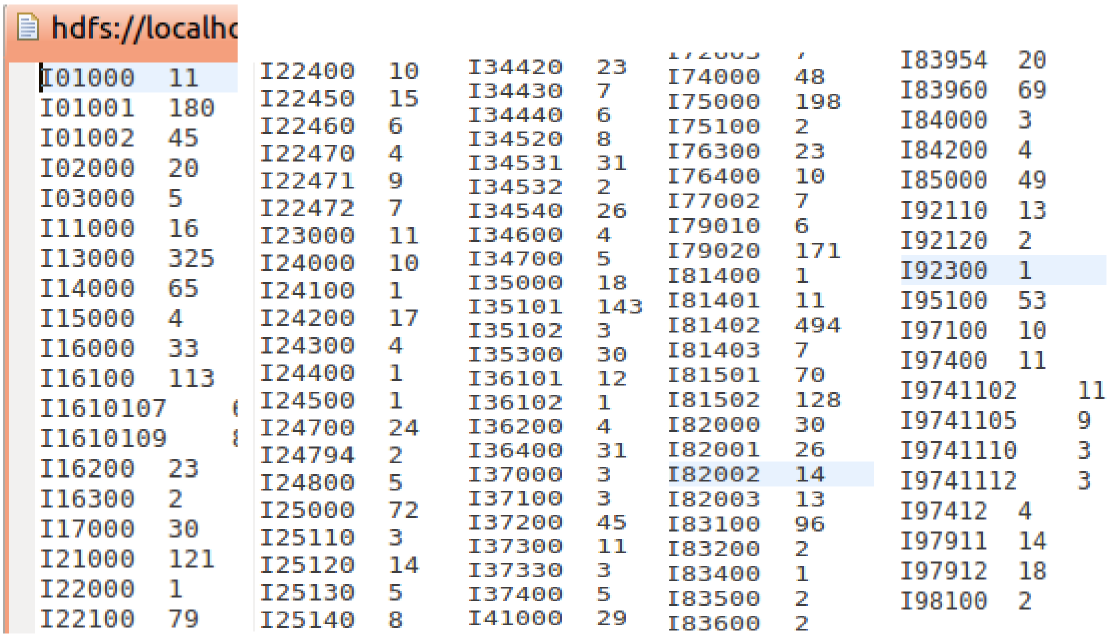
```
    static void microPrecision(){
         double TP=0;
         double TN=0;
         double FP=0;
         double FN=0;
        Iterator<String> iterator = classCSet.iterator();
        while(iterator.hasNext()){
             String c=iterator.next();
             for(Map.Entry<String, String> testedDoc:resMap.entrySet()){
                 String docName=testedDoc.getKey();
                 String resClass=testedDoc.getValue();
                 String oriClass=classOfDoc.get(docName);

                 if(c.equals(oriClass)&&c.equals(resClass))
                     TP++;
                 else if(c.equals(oriClass)&&!c.equals(resClass))
                     FN++;
                 else if(!c.equals(oriClass)&&c.equals(resClass))
                     FP++;
                 else if(!c.equals(oriClass)&&!c.equals(resClass))
                     TN++;
             }
         }
        double precision=TP/(TP+FP);
        double recall=TP/(TP+FN);

        double f1=2*precision*recall/(precision+recall);
        System.out.println("microPrecision---------  "+precision*100+"%");
    }
}
```

五：数据集说明

本项目中使用了 data/Industry 目录下的 4999 个文件。

其中随机选择的测试文件 469 个，训练文件 4533 个。

对训练集使用 CountDocType 执行文件类型的 MapReduce 统计，输出的文件类型与个数如下：



hdfs://localhc

```
I01000   11
I01001   180
I01002   45
I02000   20
I03000   5
I11000   16
I13000   325
I14000   65
I15000   4
I16000   33
I16100   113
I1610107
I1610109
I16200   23
I16300   2
I17000   30
I21000   121
I22000   1
I22100   79
```

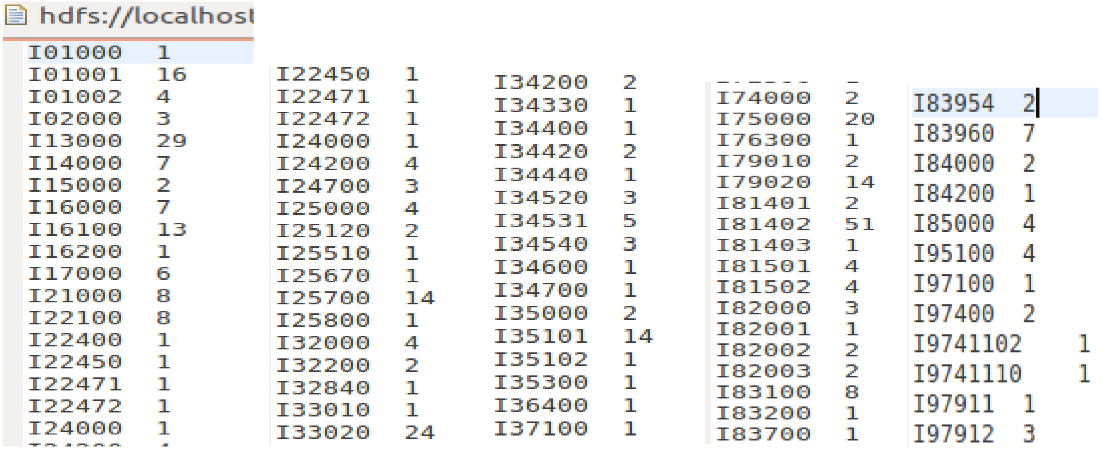
```
I22400   10
I22450   15
I22460   6
I22470   4
I22471   9
I22472   7
I23000   11
I24000   10
I24100   1
I24200   17
I24300   4
I24400   1
I24500   1
I24700   24
I24794   2
I24800   5
I25000   72
I25110   3
I25120   14
I25130   5
I25140   8
```

```
I34420   23
I34430   7
I34440   6
I34520   8
I34531   31
I34532   2
I34540   26
I34600   4
I34700   5
I35000   18
I35101   143
I35102   3
I35300   30
I36101   12
I36102   1
I36200   4
I36400   31
I37000   3
I37100   3
I37200   45
I37300   11
I37330   3
I37400   5
I41000   29
```

```
I74000   48
I75000   198
I75100   2
I76300   23
I76400   10
I77002   7
I79010   6
I79020   171
I81400   1
I81401   11
I81402   494
I81403   7
I81501   70
I81502   128
I82000   30
I82001   26
I82002   14
I82003   13
I83100   96
I83200   2
I83400   1
I83500   2
I83600   2
```

```
I83954   20
I83960   69
I84000   3
I84200   4
I85000   49
I92110   13
I92120   2
I92300   1
I95100   53
I97100   10
I97400   11
I9741102      11
I9741105      9
I9741110      3
I9741112      3
I97412   4
I97911   14
I97912   18
I98100   2
```

对测试集使用 CountDocType 执行文件类型的 MapReduce 统计，输出的文件类型与个数如下：



hdfs://localhost

```
I01000   1
I01001   16
I01002   4
I02000   3
I13000   29
I14000   7
I15000   2
I16000   7
I16100   13
I16200   1
I17000   6
I21000   8
I22100   8
I22400   1
I22450   1
I22471   1
I22472   1
I24000   1
```

```
I22450   1
I22471   1
I22472   1
I24000   1
I24200   4
I24700   3
I25000   4
I25120   2
I25510   1
I25670   1
I25700   14
I25800   1
I32000   4
I32200   2
I32840   1
I33010   1
I33020   24
```

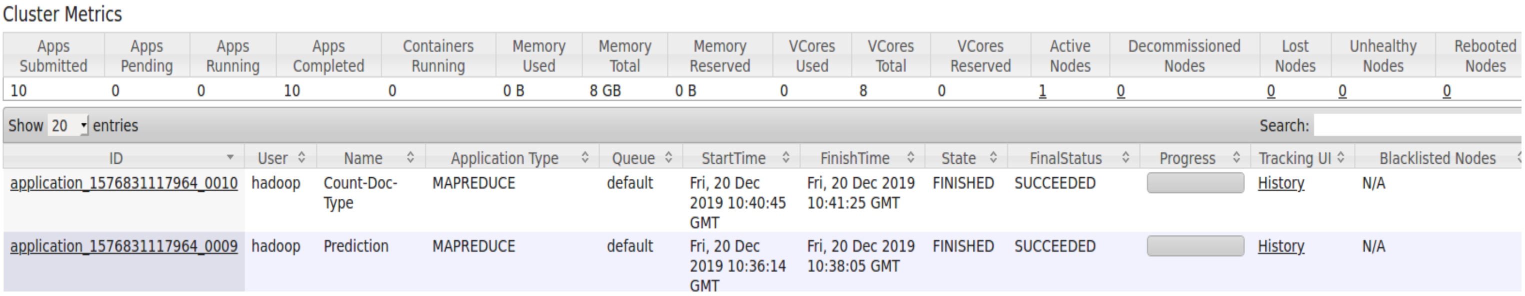
```
I34200   2
I34330   1
I34400   1
I34420   2
I34440   1
I34520   3
I34531   5
I34540   3
I34600   1
I34700   1
I35000   2
I35101   14
I35102   1
I35300   1
I36400   1
I37100   1
```

```
I74000   2
I75000   20
I76300   1
I79010   2
I79020   14
I81401   2
I81402   51
I81403   1
I81501   4
I81502   4
I82000   3
I82001   1
I82002   2
I82003   2
I83100   8
I83200   1
I83700   1
```

```
I83954   2
I83960   7
I84000   2
I84200   1
I85000   4
I95100   4
I97100   1
I97400   2
I9741102      1
I9741110      1
I97911   1
I97912   3
```

在全部类型中随机选择测试集时是按照 10%的比例，可以看出在具体到某一具体类型时，其训练集与测试集的比例也在 10：1 左右。

六：程序运行说明

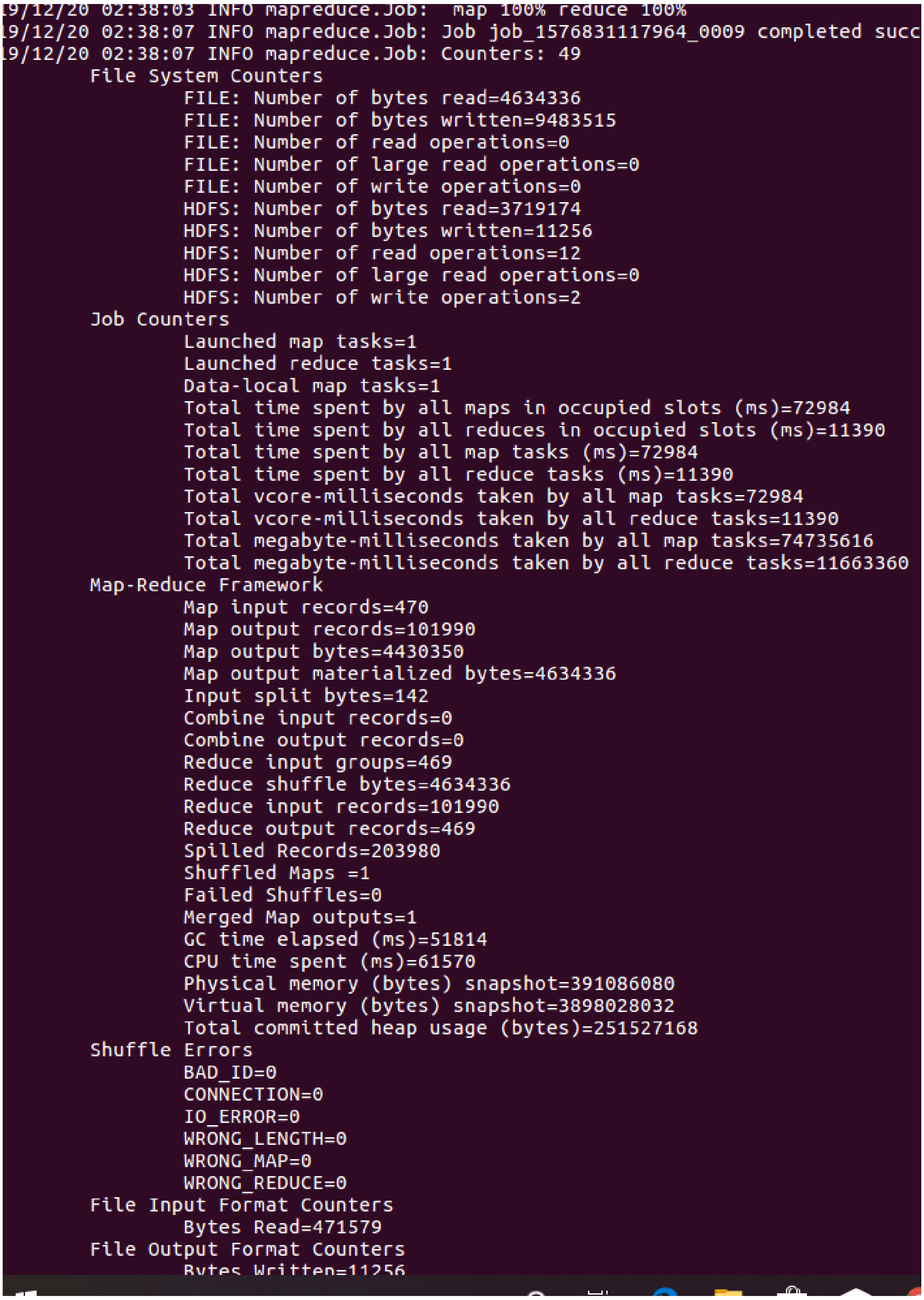


Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
10	0	0	10	0	0 B	8 GB	0 B	0	8	0	1	0	0	0	0

Show 20 entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1576831117964_0010	hadoop	Count-Doc-Type	MAPREDUCE	default	Fri, 20 Dec 2019 10:40:45 GMT	Fri, 20 Dec 2019 10:41:25 GMT	FINISHED	SUCCEEDED		History	N/A
application_1576831117964_0009	hadoop	Prediction	MAPREDUCE	default	Fri, 20 Dec 2019 10:36:14 GMT	Fri, 20 Dec 2019 10:38:05 GMT	FINISHED	SUCCEEDED		History	N/A

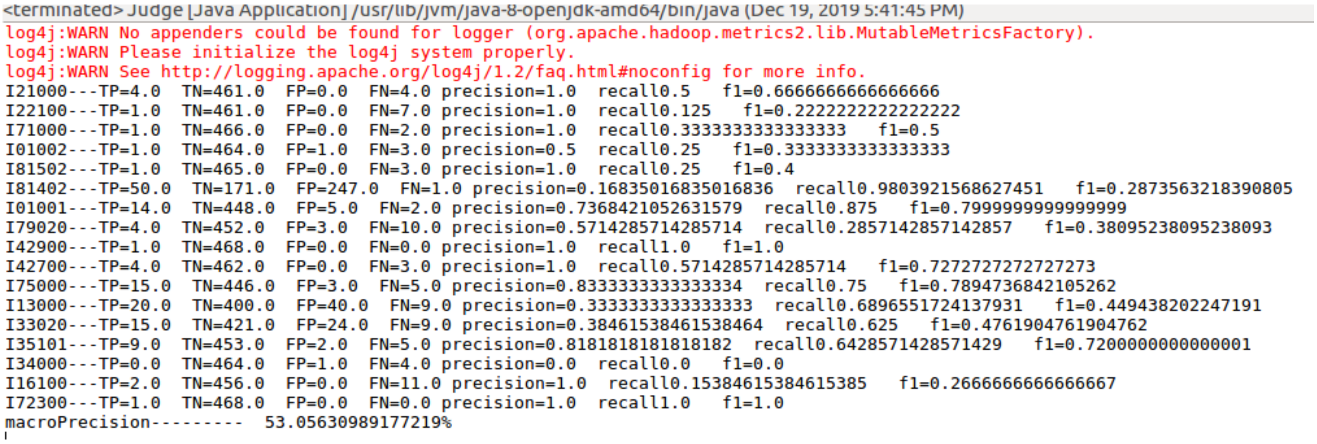


```
19/12/20 02:38:03 INFO mapreduce.Job:  map 100% reduce 100%
19/12/20 02:38:07 INFO mapreduce.Job: Job job_1576831117964_0009 completed succ
19/12/20 02:38:07 INFO mapreduce.Job: Counters: 49
	File System Counters
		FILE: Number of bytes read=4634336
		FILE: Number of bytes written=9483515
		FILE: Number of read operations=0
		FILE: Number of large read operations=0
		FILE: Number of write operations=0
		HDFS: Number of bytes read=3719174
		HDFS: Number of bytes written=11256
		HDFS: Number of read operations=12
		HDFS: Number of large read operations=0
		HDFS: Number of write operations=2
	Job Counters
		Launched map tasks=1
		Launched reduce tasks=1
		Data-local map tasks=1
		Total time spent by all maps in occupied slots (ms)=72984
		Total time spent by all reduces in occupied slots (ms)=11390
		Total time spent by all map tasks (ms)=72984
		Total time spent by all reduce tasks (ms)=11390
		Total vcore-milliseconds taken by all map tasks=72984
		Total vcore-milliseconds taken by all reduce tasks=11390
		Total megabyte-milliseconds taken by all map tasks=74735616
		Total megabyte-milliseconds taken by all reduce tasks=11663360
	Map-Reduce Framework
		Map input records=470
		Map output records=101990
		Map output bytes=4430350
		Map output materialized bytes=4634336
		Input split bytes=142
		Combine input records=0
		Combine output records=0
		Reduce input groups=469
		Reduce shuffle bytes=4634336
		Reduce input records=101990
		Reduce output records=469
		Spilled Records=203980
		Shuffled Maps =1
		Failed Shuffles=0
		Merged Map outputs=1
		GC time elapsed (ms)=51814
		CPU time spent (ms)=61570
		Physical memory (bytes) snapshot=391086080
		Virtual memory (bytes) snapshot=3898028032
		Total committed heap usage (bytes)=251527168
	Shuffle Errors
		BAD_ID=0
		CONNECTION=0
		IO_ERROR=0
		WRONG_LENGTH=0
		WRONG_MAP=0
		WRONG_REDUCE=0
	File Input Format Counters
		Bytes Read=471579
	File Output Format Counters
		Bytes Written=11256
```

可以看到只有一个 Map 和一个 Reduce，这是因为前面使用 SequenceFile 将小文件都合并成了一个文件，该文件的大小要小于默认 split 的大小，故而只有一个 map。

七：实验结果分析

对于 Macoraverage Precision，输出如图：



```
<terminated> Judge [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (Dec 19, 2019 5:41:45 PM)
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.lib.MutableMetricsFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
I21000---TP=4.0  TN=461.0  FP=0.0  FN=4.0 precision=1.0  recall0.5   f1=0.6666666666666666
I22100---TP=1.0  TN=461.0  FP=0.0  FN=7.0 precision=1.0  recall0.125   f1=0.2222222222222222
I71000---TP=1.0  TN=466.0  FP=0.0  FN=2.0 precision=1.0  recall0.3333333333333333   f1=0.5
I01002---TP=1.0  TN=464.0  FP=1.0  FN=3.0 precision=0.5  recall0.25   f1=0.3333333333333333
I81502---TP=1.0  TN=465.0  FP=0.0  FN=3.0 precision=1.0  recall0.25   f1=0.4
I81402---TP=50.0  TN=171.0  FP=247.0  FN=1.0 precision=0.16835016835016836  recall0.9803921568627451   f1=0.2873563218390805
I01001---TP=14.0  TN=448.0  FP=5.0  FN=2.0 precision=0.7368421052631579  recall0.875   f1=0.7999999999999999
I79020---TP=4.0  TN=452.0  FP=3.0  FN=10.0 precision=0.5714285714285714  recall0.2857142857142857   f1=0.38095238095238093
I42900---TP=1.0  TN=468.0  FP=0.0  FN=0.0 precision=1.0  recall1.0   f1=1.0
I42700---TP=4.0  TN=462.0  FP=0.0  FN=3.0 precision=1.0  recall0.5714285714285714   f1=0.7272727272727273
I75000---TP=15.0  TN=446.0  FP=3.0  FN=5.0 precision=0.8333333333333334  recall0.75   f1=0.7894736842105262
I13000---TP=20.0  TN=400.0  FP=40.0  FN=9.0 precision=0.3333333333333333  recall0.6896551724137931   f1=0.449438202247191
I33020---TP=15.0  TN=421.0  FP=24.0  FN=9.0 precision=0.38461538461538464  recall0.625   f1=0.4761904761904762
I35101---TP=9.0  TN=453.0  FP=2.0  FN=5.0 precision=0.8181818181818182  recall0.6428571428571429   f1=0.7200000000000001
I34000---TP=0.0  TN=464.0  FP=1.0  FN=4.0 precision=0.0  recall0.0   f1=0.0
I16100---TP=2.0  TN=456.0  FP=0.0  FN=11.0 precision=1.0  recall0.15384615384615385   f1=0.2666666666666667
I72300---TP=1.0  TN=468.0  FP=0.0  FN=0.0 precision=1.0  recall1.0   f1=1.0
macroPrecision---------  53.05630989177219%
```

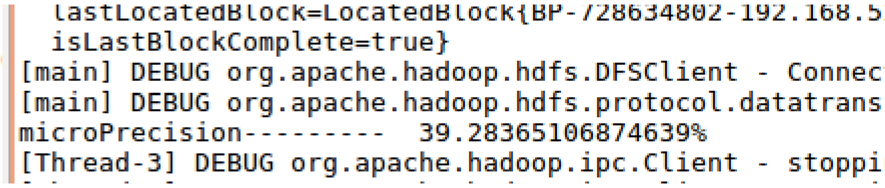
如上图，对于 I21000，其训练集中有 121 个文件，测试集中有 8 个文件，其 f1=66.6%；

对于 I81502，其训练集中有 128 个文件，测试集中有 4 个文件，其 f1=40%；

对于 I13000，其训练集中有 325 个文件，测试集中有 29 个文件，其 f1=44.9%

可以看出由于训练集与测试数据集的数量不同，每一类的 precision、recall 与 f1 的值都有较大差异。

对所有类计算 Micoraverage Precision，结果如图：



```
  lastLocatedBlock=LocatedBlock{BP-728634802-192.168.5
  isLastBlockComplete=true}
[main] DEBUG org.apache.hadoop.hdfs.DFSClient - Connec
[main] DEBUG org.apache.hadoop.hdfs.protocol.datatrans
microPrecision---------  39.28365106874639%
[Thread-3] DEBUG org.apache.hadoop.ipc.Client - stoppi
```