****

**研究生（Hadoop/MapReduce编程模型与应用）课程报告**

**题 目：基于MapReduce框架实现朴素贝叶斯分类**

**学 号： M201977184**

**姓 名： 程诚**

**专 业： 计算机技术**

**指 导 教 师： 辜希武 讲师**

**院（系、所）： 计算机科学与技术学院**

目 录

[1 朴素贝叶斯算法理论介绍 1](#_Toc25938539)

[2 MapReduce算法设计 3](#_Toc25938540)

[2.1 数据预处理 3](#_Toc25938541)

[2.2 MapReduce Job1 4](#_Toc25938542)

[2.3 MapReduce Job2 4](#_Toc25938543)

[2.4 MapReduce Job3 5](#_Toc25938544)

[3 源代码清单 7](#_Toc25938545)

[3.1 Job1.java 7](#_Toc25938546)

[3.2 Job2.java 11](#_Toc25938547)

[3.3 Job3.java 15](#_Toc25938548)

[3.4 JobsInitiator.java 18](#_Toc25938549)

[3.5 其余代码 19](#_Toc25938550)

[4 数据集说明 20](#_Toc25938551)

[5 程序运行说明 20](#_Toc25938552)

[5.1 SmallFilesToSequenceFileConverter 21](#_Toc25938553)

[5.2 MapReduce Job1 22](#_Toc25938554)

[5.3 MapReduce Job2 23](#_Toc25938555)

[5.4 MapReduce Job3 24](#_Toc25938556)

[5.5 测试集分类结果评价 25](#_Toc25938557)

[5.5 Web UI界面监控截图 25](#_Toc25938558)

[6 实验结果分析 26](#_Toc25938559)

[宏平均评价指标： 26](#_Toc25938560)

[微平均评价指标： 26](#_Toc25938561)

1 朴素贝叶斯算法理论介绍

朴素贝叶斯算法（Naive Bayes）是一种属于监督学习的分类模型生成算法，实现简单，没有迭代，基于贝叶斯理论，其中的“朴素”指的是假设每个样本的各个特征之间强相互独立，即每个特征与其他特征都不相关。显然这个假设在实际中不可能为真，但朴素贝叶斯算法在实际运用中依旧有不错的分类效果，是个十分经典的机器学习算法。

朴素贝叶斯分类的内容可以简单表示如下：

1) 设特征集合为，而已知是一个待分类项

2) 已知类别集合

3) 计算条件概率，即已知待分类项*X*，求其属于的概率

4) 设条件概率最大值为，则认为*X*属于类别

朴素贝叶斯分类重点在于计算条件概率，由贝叶斯公式可得：

|  |  |  |
| --- | --- | --- |
|  |  |  |

由于最后是取最大概率，而每个条件概率在式(1.1)中分母相同，故有：

|  |  |  |
| --- | --- | --- |
|  |  |  |

即计算条件概率可以转换成计算概率，又由于假设各个特征之间强相互独立，故有：

|  |  |  |
| --- | --- | --- |
|  |  |  |

即：

|  |  |  |
| --- | --- | --- |
|  |  |  |

其中先验概率为每个类别出现的概率，条件概率为特征在类别中出现的概率。

建立朴素贝叶斯分类器的过程实际上就是利用训练集求每个类别出现的概率和每个特征在各个类别中出现的概率的过程，其中,，对待分类项*X*的分类过程，实际上就是求其在各个类别中出现的概率，取最大概率对应的类别作为其分类类别。

2 MapReduce算法设计

本次实验的主要内容是使用朴素贝叶斯算法针对文档训练集建立文档分类模型，并使用训练后的模型对文档测试集进行分类，最后计算并输出分类结果的各项评价指标。

模型建立分为三个部分，第一个部分统计各类文档中各个单词的数目，第二个部分统计类文档的单词总数和文档总数，用于之后计算各类文档出现的概率，即先验概率，第三部分计算各类文档各个单词出现的概率，即条件概率。

其中整体实验分为数据预处理、建立分类器、分类测试三个部分，其中建立分类器所使用的MapReduce程序分为三个Job，具体内容如下所示：

2.1 数据预处理

为了使得各类文档的先验概率随机，在进行MapReduce程序设计之前，将原始数据集的所有文档名中加上文档类别，使得修改后的文件名为“文档类别—文档ID”的格式，然后将整个数据集按1:1的比例随机分为训练集和测试集。

在Hadoop集群中使用MapReduce程序处理小文件时，由于每个小文件都有占有一个block，都会形成一个split，都需要一个MapTask和ReduceTask来进行处理，因此会产生大量的Task，整个MapReduce程序会在资源的申请和释放上花费大量的时间，同时大量block的原数据会增加NameNode的内存压力。

为了消除训练集中小型文档对Hadoop集群运行MapReduce程序产生的负面影响，提高生成分类模型的效率，在建立分类器模型之前，单机环境下离线运行单个MapReduce Job将训练集聚集生成单个SequenceFile文件，然后上传到HDFS集群上，便于之后的Job读取。在运行MapReduce程序生成SequenceFile文件时，使用自定义的CombineFileInputFormat输入格式整合所有小文件block尽可能形成少量的split，减少MapTask的数量，提高整合效率。

2.2 MapReduce Job1

**Job1：**统计各类文档中各个单词的数量

**输入路径**：训练集SequenceFile文件路径

**输出路径**：自定义Job1输出路径，如：/src/Output/Job1/

**备注**：输入的文件为SequenceFile文件格式，使用SequenceFileInputFormat作为输入格式

**Job1Mapper**：

**输入**：<文档名 , 整个文档记录>，**输出：**<文档类别 单词 , "1">

**备注**：其中“文档类别”通过划分文件名取其前缀获得，“单词”通过对文档记录划分可得，文档类别和单词之间使用制表符“\t”划分

**Job1Combiner**：即Reducer

**Job1Reducer**：

**输入**：<文档类别 单词，{1,1,1,1,1,1……}>，**输出**：<文档类别 单词，单词总数>



2.3 MapReduce Job2

**Job2**：统计出各类文档中单词总数和各类文档总数

**输入路径**：训练集SequenceFile文件路径

**输出路径**：自定义Job2输出路径，如：/src/Output/Job2/

**备注**：输入的文件为SequenceFile文件格式，使用SequenceFileInputFormat作为输入格式

**Job2Mapper**：

**输入**：<文档名 , 整个文档>，**输出**：<文档类别 , 本文档中单词总数>

**Job2Combiner**：No Combiner

**备注**：此种场景不能使用Combiner

**Job2Reducer**：

**输入**：<文档类别，{sum1，sum2……}>，**输出**：<文档类别 本类文档单词总数，本类文档个数>

**备注**：其中“本类文档个数”由sum的个数来计算，每有一个sum则说明有一个对应的文档属于当前类别，因为是每个一个文档作为一个记录进行处理



2.4 MapReduce Job3

**Job3**：计算各个类别文档中各种单词出现的条件概率

**输入路径**：Job1的输出路径，即获取各个类别中各个单词的总数

**输出路径**：自定义Job3输出路径，如：/src/Output/Job3/

**备注**：MapReduce程序会自动忽略以下划线"\_"开头的文件，不用担心\_success文件，本来\_success文件也是空文件

**Job3Mapper**：

**输入**：<记录偏移量LongWritable，<文档类别 单词 单词总数>>

**输出**：<文档类别-单词，条件概率>

**描述**：通过读取Job2的输出文档获得各个类别对应的单词总数，通过读取Job1的输出文档获取各个类别各个单词的总数，由此来计算各个类别中各个单词的出现的条件概率

**备注**：在Job3的Mapper类中重写setup方法，每次启动MapTask时读取Job2的输出文件，在Mapper中读取对应文档类别的单词总数，计算单词出现概率

**Job3Combiner**：No Combiner

**Job3Reducer**：No Reducer



3 源代码清单

3.1 Job1.java

|  |
| --- |
| package com**.**TomAndersen**.**hadoop**.**BayesClassification**;**  **import** com**.**TomAndersen**.**hadoop**.**HDFSTools**.**CombineSmallfileInputFormat**;**  **import** com**.**TomAndersen**.**hadoop**.**HDFSTools**.**WholeFileInputFormat**;**  **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**.**FileSplit**;**  **import** org**.**apache**.**hadoop**.**mapreduce**.**lib**.**input**.**SequenceFileInputFormat**;**  **import** org**.**apache**.**hadoop**.**mapreduce**.**lib**.**output**.**FileOutputFormat**;**  **import** org**.**apache**.**hadoop**.**util**.**Tool**;**  **import** java**.**io**.**IOException**;**  /\*\*  \* @Author TomAndersen  \* @Version  \* @Date 2019/11/4  \* @Description ：Job1统计出各类文档中各种单词的数目  \* Job1:  \* 将一个文档作为一个记录处理  \* 输入路径：训练集  \* 输出路径：自定义Job1输出路径，如：/src/OutPut/Job1/  \* Mapper：  \* 输入：<文档名,整个文档记录>，输出：<文档类别 单词，1>  \* 其中文档类别通过读取文件名前缀获得，单词通过对text划分可得  \* Conbiner：即Reducer  \* Reducer：  \* 输入：<文档类别 单词，{1,1,1,1,1,1……}>，输出：<文档类别 单词，sum>  \*/  public class Job1 **extends** Configured **implements** Tool **{**  @Override  public int run**(**String**[]** args**)** **throws** Exception **{**  // 重写run方法，外部使用ToolRunner启动Job  // 第一个参数为训练集，第二个参数为输出路径  String InputPath **=** args**[**0**];**  String OutputPath **=** args**[**1**];**  Configuration configuration **=** **new** Configuration**();**  Job job **=** Job**.**getInstance**(**configuration**,** **this.**getClass**().**getName**());**  job**.**setInputFormatClass**(**SequenceFileInputFormat**.**class**);**//使用SequenceFile作为输入  job**.**setJarByClass**(**Job1**.**class**);**//设置主类  job**.**setMapperClass**(**Job1Mapper**.**class**);**// 设置Mapper  job**.**setCombinerClass**(**Job1Reducer**.**class**);**// 设置Combiner  job**.**setReducerClass**(**Job1Reducer**.**class**);**// 设置Reducer  //默认使用的OutputFormat是TextOutputFormat，使用时一定要指定Map输出的Key和Value类型  job**.**setMapOutputKeyClass**(**Text**.**class**);**  job**.**setMapOutputValueClass**(**IntWritable**.**class**);**  //设置job输出的key类型  job**.**setOutputKeyClass**(**Text**.**class**);**  //设置job输出的value类型  job**.**setOutputValueClass**(**IntWritable**.**class**);**  FileInputFormat**.**addInputPath**(**job**,** **new** Path**(**InputPath**));**// 添加输入路径  FileOutputFormat**.**setOutputPath**(**job**,** **new** Path**(**OutputPath**));**// 设置输出路径  **return** job**.**waitForCompletion**(true)** **?** 0 **:** 1**;**  **}**  //Mapper  public static class Job1Mapper **extends** Mapper**<**Text**,** BytesWritable**,** Text**,** IntWritable**>** **{**  private final static Text KEYOUT **=** **new** Text**();**//预定义KeyOut  private final static IntWritable VALUEOUT **=** **new** IntWritable**(**1**);**//预定义ValueOut  @Override  public void map**(**Text KeyIn**,** BytesWritable ValueIn**,** Context context**)**  **throws** IOException**,** InterruptedException **{**//KeyIn是文档名，ValueIn是文档内容  // Text, BytesWritable, Text, IntWritable输入输出键值对类型  // 将文档内容按照回车分割，即分割成一个个单词  String**[]** contents **=** **new** String**(**ValueIn**.**copyBytes**()).**split**(**"\r\n|\n|\r"**);**  // 以各种系统的换行形式作为分隔符进行切分  // 获取文档类别，文档名中已有类别戳  String fileClass **=** KeyIn**.**toString**().**split**(**"-"**)[**0**];**  **for** **(**String word **:** contents**)** **{**  // 对每个单词加上类别名  KEYOUT**.**set**(**fileClass **+** "\t" **+** word**);**//设置KeyOut，使用制表符来间隔文档类别和单词  //输出格式为<类别名 单词，“1”>  context**.**write**(**KEYOUT**,** VALUEOUT**);**  **}**  **}**  **}**  //Reducer  public static class Job1Reducer **extends** Reducer**<**Text**,** IntWritable**,** Text**,** IntWritable**>** **{**  private final static IntWritable VALUEOUT **=** **new** IntWritable**();**//预定义ValueOut  @Override  public void reduce**(**Text KeyIn**,** Iterable**<**IntWritable**>** ValuesIn**,** Context context**)**  **throws** IOException**,** InterruptedException **{**  int sum **=** 0**;**  **for** **(**IntWritable Value **:** ValuesIn**)** **{**  sum **+=** Value**.**get**();**// 要使用combiner就不能只是单纯的计数+1，而应该是取值相加  **}**  // 输出<类别名 单词，单词总数>  VALUEOUT**.**set**(**sum**);**  context**.**write**(**KeyIn**,** VALUEOUT**);**  **}**  **}**  **}** |

3.2 Job2.java

|  |
| --- |
| package com**.**TomAndersen**.**hadoop**.**BayesClassification**;**  **import** com**.**TomAndersen**.**hadoop**.**HDFSTools**.**CombineSmallfileInputFormat**;**  **import** com**.**TomAndersen**.**hadoop**.**HDFSTools**.**WholeFileInputFormat**;**  **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**.**util**.**Tool**;**  **import** java**.**io**.**IOException**;**  /\*\*  \* @Author TomAndersen  \* @Version  \* @Date 2019/11/4  \* Job2：  \* 输入路径：训练集  \* 输出路径：自定义Job2输出路径，如：/src/OutPut/Job2/  \* Mapper：  \* 输入：<文档名,整个文档>，输出：<文档类别，本文档中单词总数sum>  \* Reducer：  \* 输入：<文档类别，{sum1，sum2……}>，输出：<文档类别 本类文档单词总数，本类文档个数>  \* 其中“本类文档个数”由sum的个数来计算，每有一个sum则说明有一个对应的文档属于当前类别，因为是一个文档作为一个记录进行处理  \*/  public class Job2 **extends** Configured **implements** Tool **{**  @Override  public int run**(**String**[]** args**)** **throws** Exception **{**  // 重写run方法，外部使用ToolRunner启动Job  // 第一个参数为训练集，第二个参数为输出路径  String InputPath **=** args**[**0**];**  String OutputPath **=** args**[**1**];**  Configuration configuration **=** **new** Configuration**();**  Job job **=** Job**.**getInstance**(**configuration**,** **this.**getClass**().**getName**());**  // 设置输入格式  //job.setInputFormatClass(WholeFileInputFormat.class);//使用整个文件内容作为记录输入  //job.setInputFormatClass(CombineSmallfileInputFormat.class);//使用整合小文件block的方式输入  job**.**setInputFormatClass**(**SequenceFileInputFormat**.**class**);**//使用SequenceFile作为输入  job**.**setJarByClass**(**Job2**.**class**);**//设置主类  job**.**setMapperClass**(**Job2Mapper**.**class**);**// 设置Mapper  // 不能设置combiner  job**.**setReducerClass**(**Job2Reducer**.**class**);**// 设置Reducer  job**.**setNumReduceTasks**(**1**);**// 设置单个ReduceTask，确保输出文件只有一个，便于后面的Job进行读取  // 默认使用的OutputFormat是TextOutputFormat，使用时一定要指定Map和Reduce输出的Key-Value类型  // 设置Map输出Key Value类型  job**.**setMapOutputKeyClass**(**Text**.**class**);**  job**.**setMapOutputValueClass**(**IntWritable**.**class**);**  //设置job输出的key类型  job**.**setOutputKeyClass**(**Text**.**class**);**  //设置job输出的value类型  job**.**setOutputValueClass**(**IntWritable**.**class**);**  FileInputFormat**.**addInputPath**(**job**,** **new** Path**(**InputPath**));**// 添加输入路径  FileOutputFormat**.**setOutputPath**(**job**,** **new** Path**(**OutputPath**));**// 设置输出路径  **return** job**.**waitForCompletion**(true)** **?** 0 **:** 1**;**  **}**  public static class Job2Mapper **extends** Mapper**<**Text**,** BytesWritable**,** Text**,** IntWritable**>** **{**  private final static Text KEYOUT **=** **new** Text**();**//预定义KeyOut，避免多次New  private final static IntWritable VALUEOUT **=** **new** IntWritable**();**//预定义ValueOut，避免多次New  @Override  public void map**(**Text KeyIn**,** BytesWritable ValueIn**,** Context context**)**  **throws** IOException**,** InterruptedException **{**  // 对整个文档内容按回车、换行、空字符进行分割，分割产生的碎片数量即为本文档中单词总数  String**[]** contents **=** **new** String**(**ValueIn**.**copyBytes**()).**split**(**"\r\n|\n|\r"**);**  // 本文档单词总数作为ValueOut  VALUEOUT**.**set**(**contents**.**length**);**  // 文件名中包含类别信息，需要分离出来  // 获取文档类别，文档名中已有类别戳  String fileClass **=** KeyIn**.**toString**().**split**(**"-"**)[**0**];**  // 文档类别作为KeyOut  KEYOUT**.**set**(**fileClass**);**  context**.**write**(**KEYOUT**,** VALUEOUT**);**  **}**  **}**  public static class Job2Reducer **extends** Reducer**<**Text**,** IntWritable**,** Text**,** IntWritable**>** **{**  private final static Text KEYOUT **=** **new** Text**();**// 预定义KeyOut  private final static IntWritable VALUEOUT **=** **new** IntWritable**();**// 预定义ValueOut  //不能使用combiner  public void reduce**(**Text KeyIn**,** Iterable**<**IntWritable**>** ValuesIn**,** Context context**)**  **throws** IOException**,** InterruptedException **{**  int sumOfWords **=** 0**;**// 本类别文档单词总数  int sumOfFiles **=** 0**;**// 本类别文档总数  **for** **(**IntWritable value **:** ValuesIn**)** **{**  //每有一个value则有一个此类别文档  sumOfWords **+=** value**.**get**();**  sumOfFiles **+=** 1**;**  **}**  // KeyOut为<文档类别 本类单词总数>,KeyIn为文档类别  String fileClass **=** KeyIn**.**toString**();**  // 以制表符 \t 作为分隔符  KEYOUT**.**set**(**fileClass **+** "\t" **+** sumOfWords**);**  // ValueOut为本类文档个数  VALUEOUT**.**set**(**sumOfFiles**);**  // 写入  context**.**write**(**KEYOUT**,** VALUEOUT**);**  **}**  **}**  **}** |

3.3 Job3.java

|  |
| --- |
| package com**.**TomAndersen**.**hadoop**.**BayesClassification**;**  **import** com**.**TomAndersen**.**hadoop**.**HDFSTools**.**BayesTools**;**  **import** org**.**apache**.**hadoop**.**conf**.**Configuration**;**  **import** org**.**apache**.**hadoop**.**conf**.**Configured**;**  **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**.**lib**.**input**.**FileInputFormat**;**  **import** org**.**apache**.**hadoop**.**mapreduce**.**lib**.**input**.**TextInputFormat**;**  **import** org**.**apache**.**hadoop**.**mapreduce**.**lib**.**output**.**FileOutputFormat**;**  **import** org**.**apache**.**hadoop**.**util**.**Tool**;**  **import** java**.**io**.**IOException**;**  **import** java**.**util**.**HashMap**;**  /\*\*  \* @Author TomAndersen  \* @Version  \* @Date 2019/11/4  \* Job3：  \* 输入路径：Job1的输出路径（MapReduce程序会自动忽略以下划线"\_"开头的文件，不用担心\_success文件）  \* 输出路径：自定义Job3输出路径，如：/src/OutPut/Job3/  \* Mapper：  \* 输入：< LongWritable，<文档类别 单词 sum>>  \* 输出：<文档类别 单词，条件概率p>  \* 在Mapper中读取对应文档类别的单词总数，计算单词出现概率  \* No Reducer  \*/  public class Job3 **extends** Configured **implements** Tool **{**  // public static HashMap fileClassToSumOfWords = null;// 存放文档类别到单词总数的映射，键值对类型为<String,String>  // public static HashMap fileClassToSumOfFiles = null;// 存放文档类别到文档总数的映射  @Override  public int run**(**String**[]** args**)** **throws** Exception **{**  // 获取配置信息  Configuration configuration **=** **new** Configuration**();**  //下面是对Job3的配置  String InputPath **=** args**[**0**];** // 输入路径为Job1的输出路径  String OutputPath **=** args**[**1**];**// 输出路径  // 获取Job实例  Job job **=** Job**.**getInstance**(**configuration**,** **this.**getClass**().**getName**());**  // 设置输入格式  job**.**setInputFormatClass**(**TextInputFormat**.**class**);**  // 设置Job3的主类  job**.**setJarByClass**(**Job3**.**class**);**  // 设置Mapper  job**.**setMapperClass**(**Job3Mapper**.**class**);**  // No Combiner,no Reducer，使用默认的Reducer  job**.**setNumReduceTasks**(**1**);**// 设置单个ReduceTask，确保输出文件只有一个，便于后面的Job进行读取  // 设置Mapper的输出Key-value类型  job**.**setMapOutputKeyClass**(**Text**.**class**);**  job**.**setMapOutputValueClass**(**DoubleWritable**.**class**);**  // 添加输入路径  FileInputFormat**.**addInputPath**(**job**,** **new** Path**(**InputPath**));**  // 设置输出路径  FileOutputFormat**.**setOutputPath**(**job**,** **new** Path**(**OutputPath**));**  // 返回执行状态  **return** job**.**waitForCompletion**(true)** **?** 0 **:** 1**;**  **}**  public static class Job3Mapper **extends** Mapper**<**LongWritable**,** Text**,** Text**,** DoubleWritable**>** **{**  private final static Text KEYOUT **=** **new** Text**();**//预定义KeyOut，避免多次New  private final static DoubleWritable VALUEOUT **=** **new** DoubleWritable**();**//预定义ValueOut，避免多次New  private static HashMap fileClassToSumOfWords **=** **null;**// 存放文档类别到单词总数的映射，键值对类型为<String,String>  // 重载setup方法，在setup方法中初始化fileClassToSumOfWords，setup方法每次启动Task时由run方法调用，只执行一次  @Override  public void setup**(**Context context**)** **throws** IOException**,** InterruptedException **{**  **super.**setup**(**context**);**  Configuration configuration **=** context**.**getConfiguration**();**  HashMap**[]** myMaps **=** BayesTools**.**getKeyValuesByReadFile**(**JobsInitiator**.**Job2\_OutputPath **+** "part-r-00000"**,**  configuration**,** "\t"**);**  fileClassToSumOfWords **=** myMaps**[**0**];**  **}**  // 输入：<NullWritable，<文档类别 单词 单词总数>>  // 输出：<文档类别 单词 条件概率>  @Override  public void map**(**LongWritable KeyIn**,** Text ValueIn**,** Context context**)**  **throws** IOException**,** InterruptedException **{**  // 对输入的一行文本数据按制表符 \t 进行分割  // ValueIn格式为<文档类别 单词 单词总数>，KeyIn为记录偏移量  String**[]** ValuesIn **=** ValueIn**.**toString**().**split**(**"\t"**);**  String fileClass **=** ValuesIn**[**0**];**// 获取文档类别  String word **=** ValuesIn**[**1**];**// 获取当前单词  Double sumOfWords **=** Double**.**valueOf**(**ValuesIn**[**2**]);**// 获取当前单词总数  // 获取对应类别所有单词的总数  Double sumOfClassWord **=** Double**.**valueOf**((**String**)**  Job3Mapper**.**fileClassToSumOfWords**.**get**(**fileClass**));**  // 计算单词的条件概率  // 设置输出的Key值，为<文档类别-单词>，虽然实现了TextPair，为了便于后续读取还是直接用“-”分割  KEYOUT**.**set**(**fileClass **+** "-" **+** word**);**  // 设置输出的Value值，为单词条件概率  VALUEOUT**.**set**(**sumOfWords **/** sumOfClassWord**);**  context**.**write**(**KEYOUT**,** VALUEOUT**);**  **}**  **}**  **}** |

3.4 JobsInitiator.java

|  |
| --- |
| package com**.**TomAndersen**.**hadoop**.**BayesClassification**;**  **import** com**.**TomAndersen**.**hadoop**.**HDFSTools**.**BayesTools**;**  **import** org**.**apache**.**hadoop**.**util**.**ToolRunner**;**  /\*\*  \* @Author  \* @Version  \* @Date 2019/11/5  \* 用于启动所有的Job，本类中只进行作业调度  \*/  public class JobsInitiator **{**  // 单机测试下的本地文件系统相对路径，千万注意此路径集群中不能使用，集群中路径前面需要加/  public static final String Job1\_OutputPath **=** "src/Output/Job1/"**;**  public static final String Job2\_OutputPath **=** "src/Output/Job2/"**;**  public static final String Job3\_OutputPath **=** "src/Output/Job3/"**;**  /\*// 集群测试下的HDFS文件路径  public static final String Job1\_OutputPath = "/src/Output/Job1/";  public static final String Job2\_OutputPath = "/src/Output/Job2/";  public static final String Job3\_OutputPath = "/src/Output/Job3/";  \*/  public static void main**(**String**[]** args**)** **throws** Exception **{**  **if** **(**args**.**length **<** 1**)** **{**  System**.**out**.**println**(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*At least one parameters are required!\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**);**  System**.**exit**(**1**);**  **}** **else** **if** **(**args**.**length **>** 1**)** **{**  System**.**out**.**println**(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Only one parameters are required!\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**);**  System**.**exit**(**1**);**  **}**  // 第一个参数为训练集文档路径  final String TrainDataSetPath **=** args**[**0**];**  int JobexitCode **=** 0**;**  BayesTools**.**CheckOutputPath**(**Job1\_OutputPath**);**//检查Job1输出路径是否为空  JobexitCode **+=** ToolRunner**.**run**(new** Job1**(),** **new** String**[]{**TrainDataSetPath**,** Job1\_OutputPath**});**    BayesTools**.**CheckOutputPath**(**Job2\_OutputPath**);**//检查Job2输出路径是否为空  JobexitCode **+=** ToolRunner**.**run**(new** Job2**(),** **new** String**[]{**TrainDataSetPath**,** Job2\_OutputPath**});**  BayesTools**.**CheckOutputPath**(**Job3\_OutputPath**);**//检查Job2输出路径是否为空  JobexitCode **+=** ToolRunner**.**run**(new** Job3**(),** **new** String**[]{**Job1\_OutputPath**,** Job3\_OutputPath**});**  System**.**exit**(**JobexitCode**);**// 当参数为0时表示正常终止JVM，为非0时表示异常终止  // 使用训练好的模型进行分类  // BayesTools.BayesClassifier(TestDataSetPath);  // 将分类器建模和分类器测试分为两个程序比较好，就不放在这里运行了  **}**  **}** |

3.5 其余代码

本报告只介绍了朴素贝叶斯分类器构建的MapReduce代码，关于测试集分类其余详细工程源代码请见：

<https://github.com/TomAndersen-cc/MyFirstHadoopProject>

4 数据集说明

本实验所用具体数据集如表4.1所示。其中训练集、测试集文档按照比例1:1从中随机选取。

表4.1 实验所用数据集

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **BELG** | **BRAZ** | **CANA** | **总计** |
| **文档数量** | 154 | 200 | 263 | 617 |

5 程序运行说明

在本次实验中，每个MapReduce Job运行时都只有一个MapTask和ReduceTask。首先本次实验中为了提高对小文件的处理效率，采用了生成SequenceFile文件的方式整合了数据集，由此大大减小了集群在Task资源调度时和频繁地将中间结果写入磁盘时所浪费的时间。其次由于文档数量有限，数据量太小，生成的SequenceFile文件大小依旧无法填满一个block，所以最后实际执行时，每个Job最多只有一个MapTask和ReduceTask。

其中关于程序运行时Web UI页面作业监控截图及程序运行命令行截图具体如下所示：

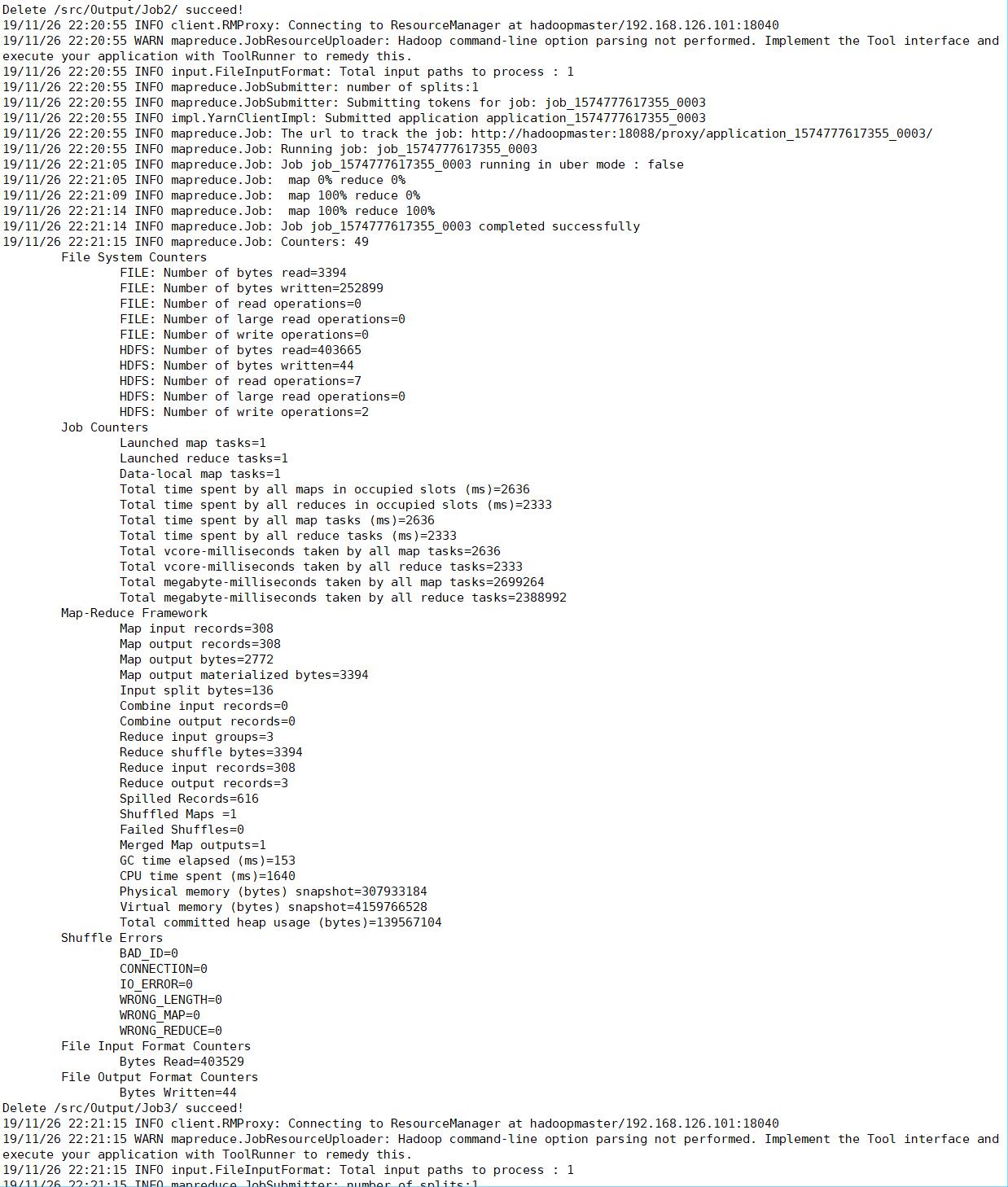
5.1 SmallFilesToSequenceFileConverter



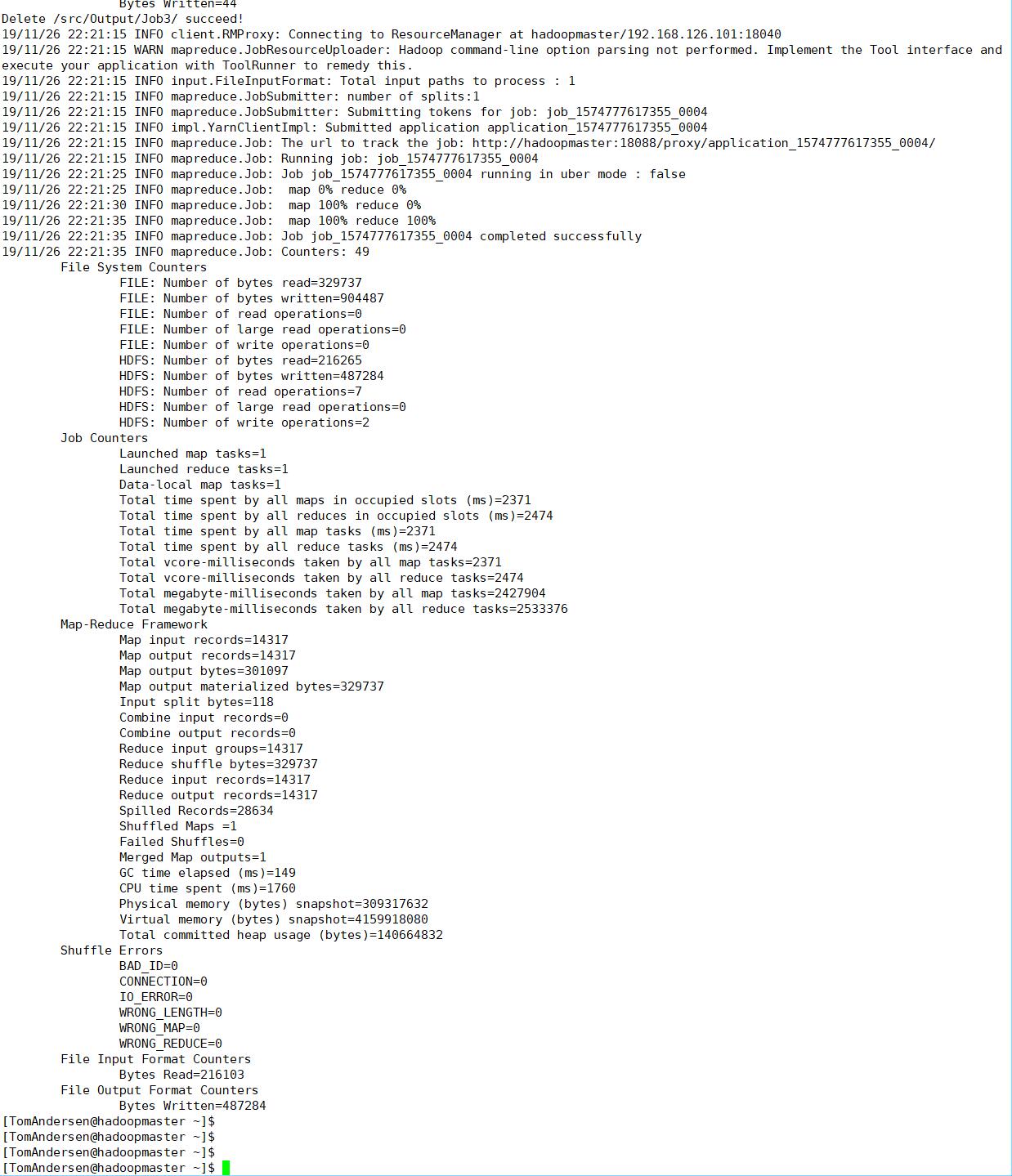
5.2 MapReduce Job1



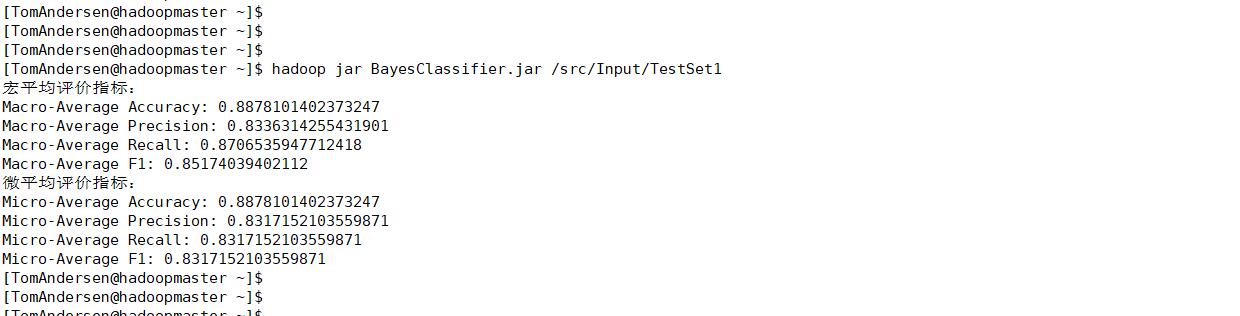
5.3 MapReduce Job2



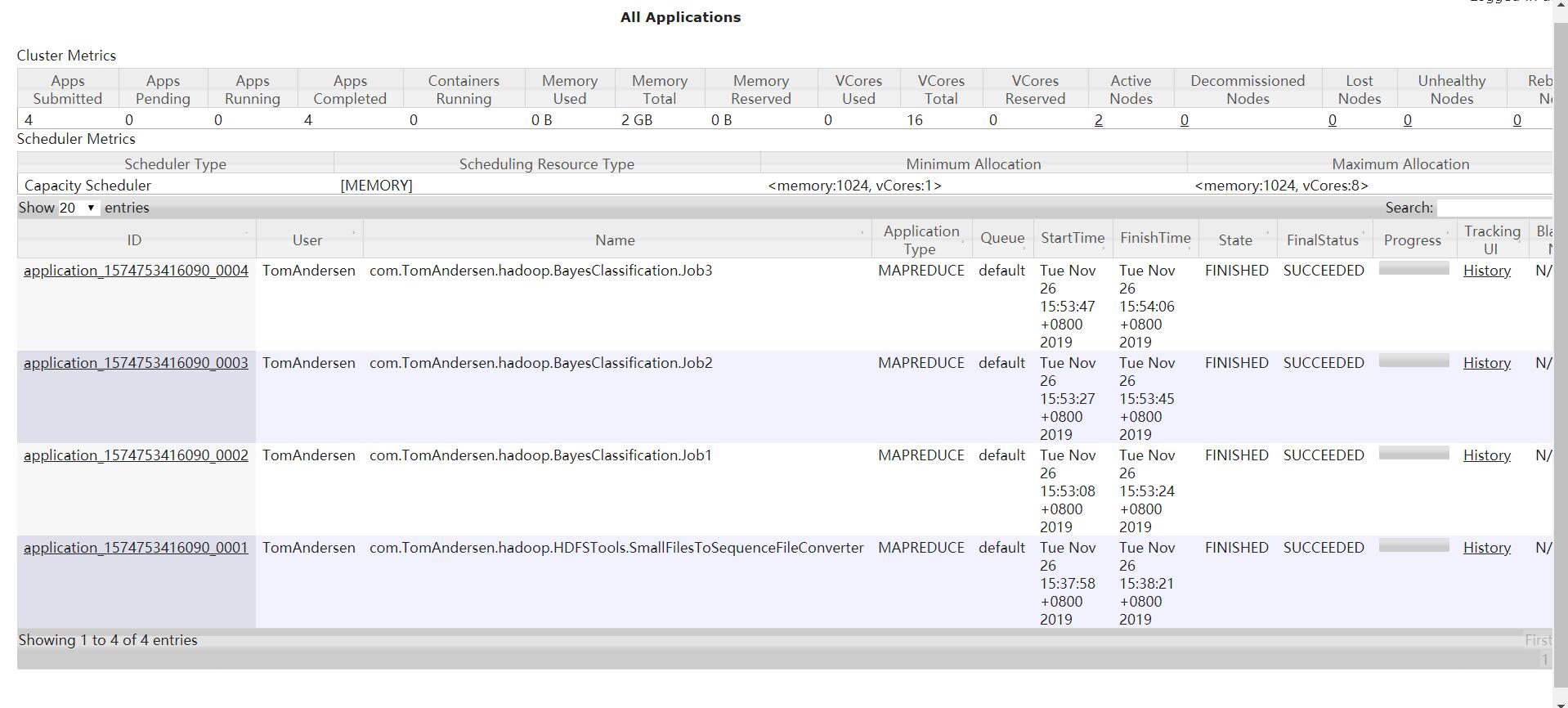
5.4 MapReduce Job3



5.5 测试集分类结果评价



5.6 Web UI界面监控截图



6 实验结果分析

本次实验中使用BELG、GRAZ、CANA三类共计617个文档，其中测试集和训练集比例为1:1，最终使用训练集构建的朴素贝叶斯分类器对测试集分类结果的各项评价指标计算如下所示。

从分类结果来看，实验所构建的朴素贝叶斯分类器效果显著。

宏平均评价指标：

Macro-Average Accuracy: 0.8878101402373247

Macro-Average Precision: 0.8336314255431901

Macro-Average Recall: 0.8706535947712418

Macro-Average F1: 0.85174039402112

微平均评价指标：

Micro-Average Accuracy: 0.8878101402373247

Micro-Average Precision: 0.8317152103559871

Micro-Average Recall: 0.8317152103559871

Micro-Average F1: 0.8317152103559871