**目录**

[1.贝叶斯分类器理论介绍 1](#_Toc58184330)

[2.贝叶斯分类器训练的MapReduce算法设计 2](#_Toc58184331)

[2.1实验环境搭建 2](#_Toc58184332)

[2.2 算法执行过程 2](#_Toc58184333)

[2.3.1 配置文件 3](#_Toc58184334)

[2.3.2 训练集与测试集划分 3](#_Toc58184335)

[2.3.3 模型训练：CalculateWordCount 4](#_Toc58184336)

[2.3.4 分类：CalculateResult 5](#_Toc58184337)

[2.3.5 评价：CalculateEvaluation 6](#_Toc58184338)

[2.3 Data Flow 7](#_Toc58184339)

[3.源代码清单 7](#_Toc58184340)

[3.1 CalculateWordCount 7](#_Toc58184341)

[3.2 CalculateResult 7](#_Toc58184342)

[3.2 CalculateEvaluation 8](#_Toc58184343)

[4.数据集说明 8](#_Toc58184344)

[5.程序运行说明 8](#_Toc58184345)

[6.实验结果分析 10](#_Toc58184346)

# 1.贝叶斯分类器理论介绍

贝叶斯方法是以贝叶斯原理为基础，使用概率统计的知识对样本数据集进行分类。由于其有着坚实的数学基础，贝叶斯分类算法的误判率是很低的。贝叶斯方法的特点是结合先验概率和后验概率，即避免了只使用先验概率的主观偏见，也避免了单独使用样本信息的过拟合现象。贝叶斯分类算法在数据集较大的情况下表现出较高的准确率，同时算法本身也比较简单。

朴素贝叶斯方法是在贝叶斯算法的基础上进行了相应的简化，即假定给定目标值时属性之间相互条件独立。也就是说没有哪个属性变量对于决策结果来说占有着较大的比重，也没有哪个属性变量对于决策结果占有着较小的比重。虽然这个简化方式在一定程度上降低了贝叶斯分类算法的分类效果，但是在实际的应用场景中，极大地简化了贝叶斯方法的复杂性。

总之贝叶斯分类器主要是使用条件概率对目标进行分类的一个分类器。基本思想是给定类别标签集合，需对d进行分类。分别计算，使条件概率最大的作为d的类别。



其中p (ci|d)为后验概率或条件概率，p (ci)为先验概率，p (d|ci)为似然概率，p (d)为证据。当先验概率

对于文本分类问题。p (ci)为ci类别占训练集文档总数的比例（概率）。



对于p (d|ci )不太容易计算，因此需要先进行Term（单词）独立性假设，即文档中每个单词的出现是彼此独立的。假设文档d包含nd个单词tnd，则p (d|ci )可转换为p (tk|ci )的乘积。



其中p (tk|ci )为：



# 2.贝叶斯分类器训练的MapReduce算法设计

## 2.1实验环境搭建

实验使用docker构建一个Ubuntu容器并搭建好hadoop3.2.1的单机伪分布式集群。启动DFS后集群包括NameNode、DataNode、SecondaryNameNode，NodeManager各一个。此环境可以直接运行MapReduce，而为了便于查看之后的MapReduce详细运行情况，需要再分别开启Yarn和JobHistoryServer，之后可在web中查看相关信息。

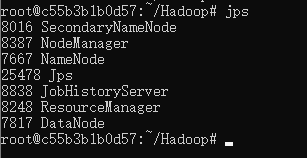


图2.1 输入JPS后查看集群详情

启动环境之后，通过HDFS命令将实验数据全部放到FS上，后续MapReduce程序可以直接访问文件。

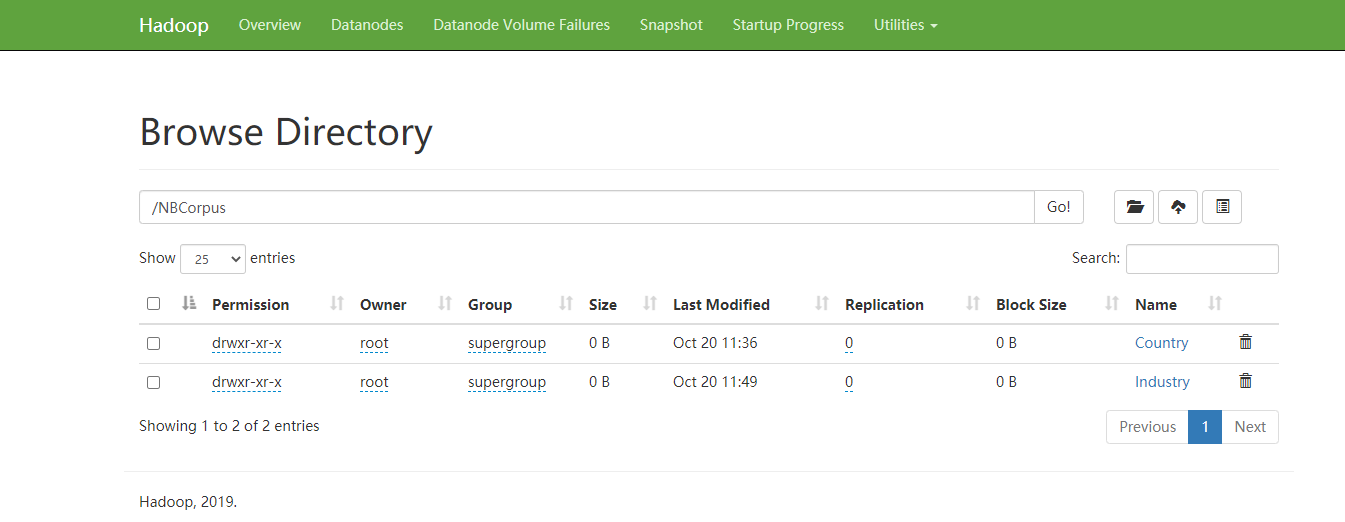


图2.2 通过Web查看HDFS上的数据集文件

## 2.2 算法执行过程

本文本分类器主要分为三大部分：配置类（Config）、主类（Classification），以及三个MapReduce类（CalculateWordCount，CalculateResult，CalculateEvaluation）。

为了便于修改路径、生成文件的名称，以及简化命令行等操作，直接将部分可定制化信息以json的形式放入一个文件中，当执行jar包时会先读取并解析配置文件。之后会对配置文件中给定路径下的文件划分为训练集与测试集。最后依次执行三个MapReduce Job：对训练集文件统计单词个数，对测试集文件统计单词个数并使用贝叶斯公式计算概率，以最大概率的类别作为分类依据，之后再对分类结果进行评价。

其中，所有Job的输入输出文件以及中间输出的临时文件全都位于HDFS上。并且所有的执行过程（数据集划分，MapReduce）全都在主类（Classification）的Main中被依次执行。在执行前会先在HDFS上以配置文件的DirPath作为名称创建一个文件夹，后续生成的所有文件都会放到文件夹中，便于整体下载与删除。如果在创建前已存在对应文件，则会先整体删除该文件再去创建。

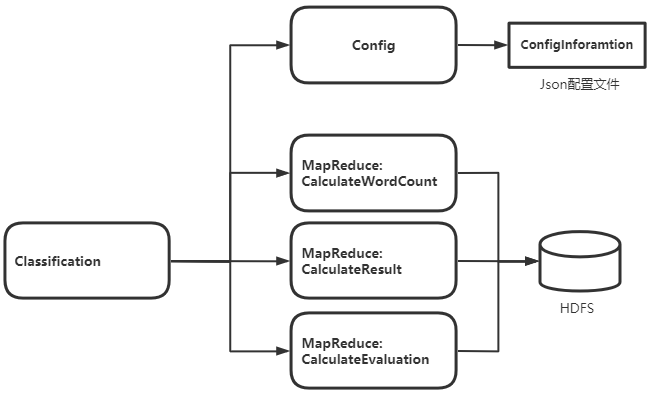


图2.3 程序主要执行过程

### 2.3.1 配置文件

配置文件以Json形式表示如下，包括HDFS的地址，要分类的类别名称个数，对应的数据集的二级子目录路径，以及各种输出路径名称等配置信息，这些信息可按需求动态修改。当Java程序打成jar包后，需要将配置文件放入相同的目录才可运行。

在程序一开始运行时会先使用Config类读取配置文件，并使用FastJson库（已合并打入Jar包）解析文件。在后续运行MapReduce Job时会读取对应字段的配置信息。

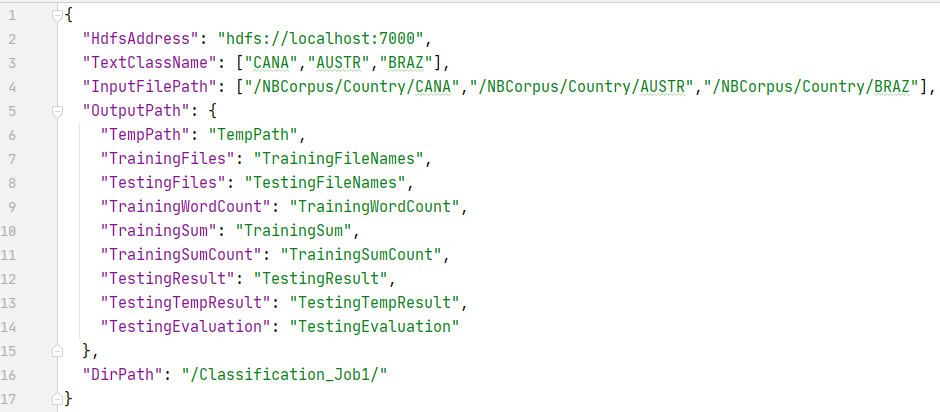


图2.4 Json配置文件示例

### 2.3.2 训练集与测试集划分

给定多个数据集二级子目录，使用FS.listStatus读取每个目录下的FileStatus。之后随机Shuffle，再以8:2的比例按打散后的顺序，将FileStatus的路径（既文本文件的路径）以文本的格式分为训练集与测试集，为了保证公平性，需注意是对每类目录文件按此比例划分再合并，而不是整体合并再划分，同时为防止目录下文件过少无法整除等情况，需要保证必须含有测试集，所以保证先取20%的测试集并向上取整，其余的作为训练集。

最后合并所有类别的训练集与测试集的文件路径，以String的格式，按每行一个Path的形式分别写入到两个文件中（TrainingFileNames，TestingFileNames）。

### 2.3.3 模型训练：CalculateWordCount

本Job是对模型进行训练，因为根据贝叶斯分类模型可知，文本分类是需要计算训练集中每类的每个单词的概率，因此首先需要对训练集文件进行WordCount。所以需要读取上一步划分得到的训练集路径文件（TrainingFileNames），逐行设置为Job的输入文件，同时这里需要按类别统计单词次数，由于只有知道总单词个数才能计算概率，因此这里考虑是将概率计算的部分后移，不输出单词概率文件，而是分别输出每个单词的每类别的次数以及每个类别的总单词个数。要获取单词的概率时需要先读取这两个文件，实时计算出概率后才能得到。这样可减少一次计算概率的MapReduce Job。

（1）Map

输入训练集文件的每一行。根据Context的路径名得知此文本属于哪个类别。然后根据配置文件的类别顺序，计算出当前类别的索引（int类型，从0开始）。输出时，以单词名作为K2，以索引序号作为V2。

（2）Reduce

根据K2/V2获取当前单词名称与各自对应的类别，因此先按类别加和计算出各自次数。再以单词名作为K3，将这些整形数字（即次数）以逗号作为间隔符转为Text作为V3，存入到TrainingWordCount/Part文件中，其中各类别次数按配置文件给出的顺序结合写入。

此外，还要分别对每个类别的总次数加上当前单词每个类别的次数。而为了输出多个文件，需要使用MultipleOutPut.Write方式，在所有Reduce执行后的CleanUp阶段，输出各类别单词总次数的信息。以固定文本”Sum”作为K3，将整形数字（次数）以逗号作为间隔符转为Text作为V3写入TrainingSum/Part文件。

（3）中间输出文件

每个单词的各类别总次数文件：TrainingWordCount/Part-r-00000

所有类别的单词总数文件：TrainingSum/Part-r-00000

（4）MapReduce的Key与Value

表2.1 CalculateWordCount的Key与Value

|  |  |  |
| --- | --- | --- |
|  | 类型 | 含义 |
| Key1 | Object | 默认 |
| Value1 | Text | 训练集文件（TrainingFileNames中的文件）的每行文本 |
| Key2 | Text | 单词名 |
| Value2 | IntWritable | 类别索引 |
| Key3-1 | Text | 单词名 |
| Value3-1 | Text | 当前单词各类别的总数，按配置的顺序给出，以逗号分隔 |
| Key3-2 | Text | 默认值:Sum |
| Value3-2 | Text | 各类别所有单词的总数，按配置的顺序给出，以逗号分隔 |

### 2.3.4 分类：CalculateResult

本Job是依据训练集中单词概率对测试集文本进行分类，最终输出一个分类结果文件。

（1）Map

首先在Map开始前的Setup阶段读取上一步训练集生成的WordCount与Sum两个文件信息，对于训练集的TrainingWordCount文件，以原始的K3和V3 解析作为HashMap的K-V值，方便后续快速查找与计算单词的概率。同时还需要预先计算出类别j的概率，也就是训练集中属于j类别的文件占全部文件的比例。

由于是对每个单独的文件计算概率并分类的，为了方便计算，考虑将整体文件作为Map的输入Value。由于默认是以输入文件的行信息进行划分传入Map的，目前是需要直接将多个测试集文件按单个完整文件分发给Map，因此这里需要定义一个WholeFileRecordReader类来实现此效果（该类继承自RecordReader）。并且在后续启动Job前设置输入类型（setInputFormatClass）。因此Map输入的Value为BytesWritable类型。

之后首先对输入的整个测试集文件进行WordCount，然后对统计后的每个单词计算概率并得到整体的乘积。若i为测试集文件，j为当前类别，k为当前测试集文件i的其中一个不重复的单词，为i中k的WordCount单词次数，为当前单词的在训练集中的单词次数，为训练集的j类别的单词总数，C为一个常数。那么可知，测试集文件i的对应j类别的概率公式如下。

因为原始概率公式为乘测试集的每个单词在训练集的概率乘积和。这里为了优化，先对训练集进行WordCount计算，再以单词次数作为幂求积。而由于训练集内单词数量较大，最终求出的每个单词的概率都比较小，因此最终乘积和会非常小，在后续实验中会出现几乎为0的情况，而且当对分母取log后，数据仍过小，因此考虑分子乘上一个系数，让最终的概率不会过小或过大，使得计算更加精确同时最终不会出现非法结果。通过多次试验，这里考虑是C为4乘上。另外，会出现训练集没有测试集出现的单词或者最终单词个数为0的情况，那么为了保证概率仍可计算，将的值设为1。

最终以当前文件的路径作为K2，将计算出的概率（原始为double类型）合并转为String并以逗号为间隔作为V2输出。

（2）Reduce

解析Map的V2，获取最大概率对应的类别，并以该类别的索引获取类别名称，并作为V3，K3仍为训练集文件路径。最终输出一个分类结果文件。每行为文件路径与分类类别名。

（3）输出文件

文件： TestingResult/Part-r-00000

（4）MapReduce的Key与Value

表2.2 CalculateResult的Key与Value

|  |  |  |
| --- | --- | --- |
|  | 类型 | 含义 |
| Key1 | Object | 默认 |
| Value1 | ByteWritable | 单个测试集的整体文件（TestingFileNames中的文件） |
| Key2 | Text | 测试文件的路径名 |
| Value2 | Text | 文件的各类别的概率，按配置的顺序给出，以逗号分隔 |
| Key3-1 | Text | 测试文件的路径名 |
| Value3-1 | Text | 文件的各类别的概率，按配置的顺序给出，以逗号分隔 |
| Key3-2 | Text | 测试文件的路径名 |
| Value3-2 | Text | 类别名 |

### 2.3.5 评价：CalculateEvaluation

本Job是对文本分类情况进行评价。主要根据TestingResult分类结果求出每个类别的Precision，Recall，F1Score。同时考虑到多分类的情况，使用微平均与宏平均额外计算给出。。

（1）Map

解析TestingResult分类结果文件。V2为文件中的每行数据。可根据测试集文件路径名获取原始文件类型。而为了方便后续计算，K2为原始的类别索引，V2为被分类后的类别索引。

（2）Reduce

根据K2，统计出每类数据的TP、FP、FN三个值（即求出混淆矩阵）。a原始类别索引，b为被分类后的类别索引。当a==b时，a的TP加1。当a!=b时，a的FN加1，并且b的FP加1。

在所有Reduce执行完成后的CleanUp阶段，根据TP、FP、FN分别计算出每类的Precision，Recall，F1Score三个值并输出。并且再使用微平均与宏平均两种方式计算与输出。因此K3与V3均为Text类型，内容为Precision，Recall，F1Score等信息。

宏平均是对所有类的Precision，Recall，F1Score计算平均数分别作为三个值。微平均是对所有类别的TP、FP、FN三个值求和（对混淆矩阵相加）后再计算。

（3）输出文件

文件： TestingEvaluation/Part-r-00000

（4）MapReduce的Key与Value

表2.3 CalculateResult的Key与Value

|  |  |  |
| --- | --- | --- |
|  | 类型 | 含义 |
| Key1 | Object | 默认 |
| Value1 | Text | CalculateResult分类结果文件的每行数据 |
| Key2 | IntWritable | 原始的类别索引 |
| Value2 | IntWritable | 被分类后的类别索引 |
| Key3 | Text | 以字符串给定的评价信息（Precision，Recall，F1Score等） |
| Value3 | Text | 以字符串给定的评价信息（Precision，Recall，F1Score等） |

## 2.3 Data Flow

主要流程为2.2算法执行过程所述。其中每个MapReduce需要由Classification主类提供解析后的配置信息，由HDFS提供输入文件。其中每个Job都会生成相应的中间文件，也会放入到HDFS上，后一步的Job会根据配置信息的路径读取前一Job的文件。

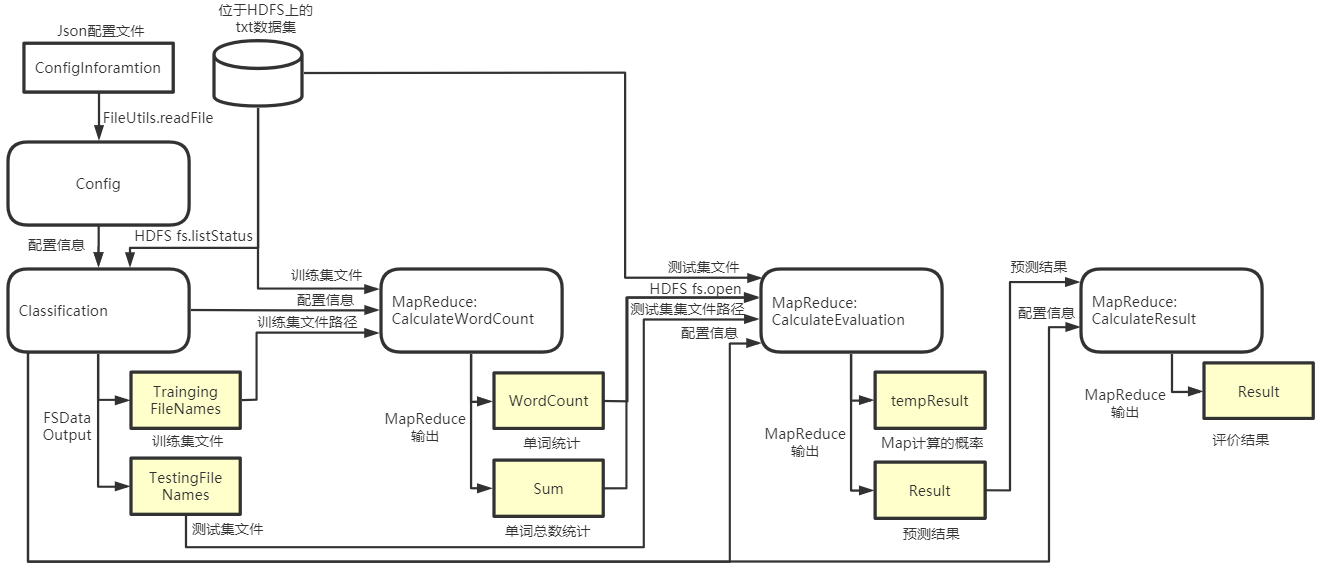


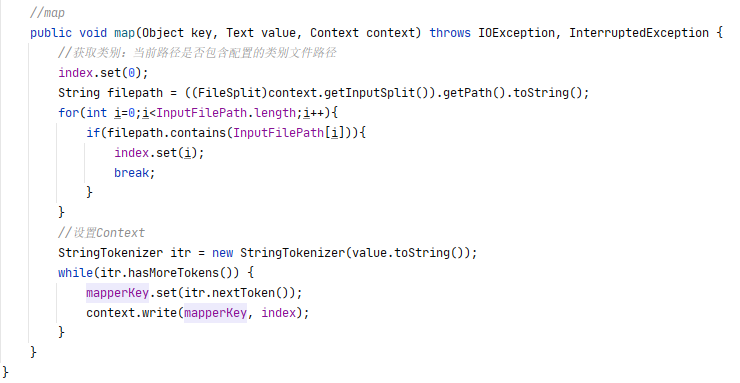
图2.5 数据流图

# 3.源代码清单

## 3.1 CalculateWordCount

（1）Map

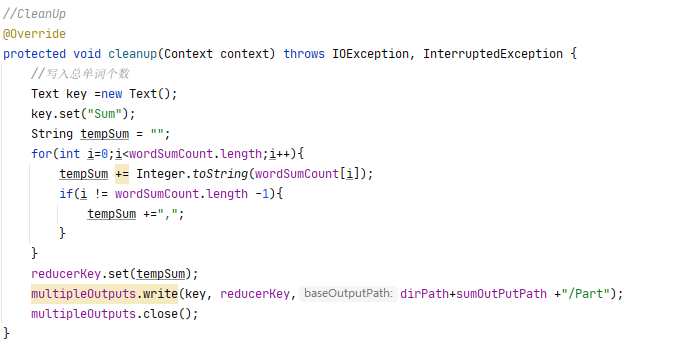




（2）Reduce

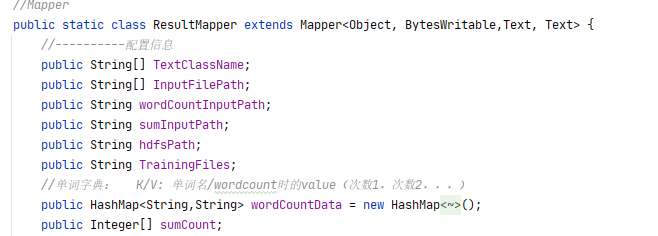


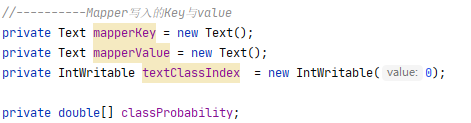




## 3.2 CalculateResult

（1）Map



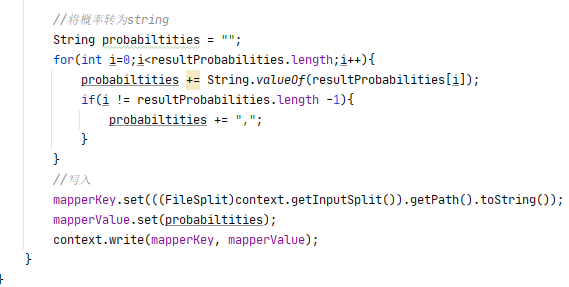






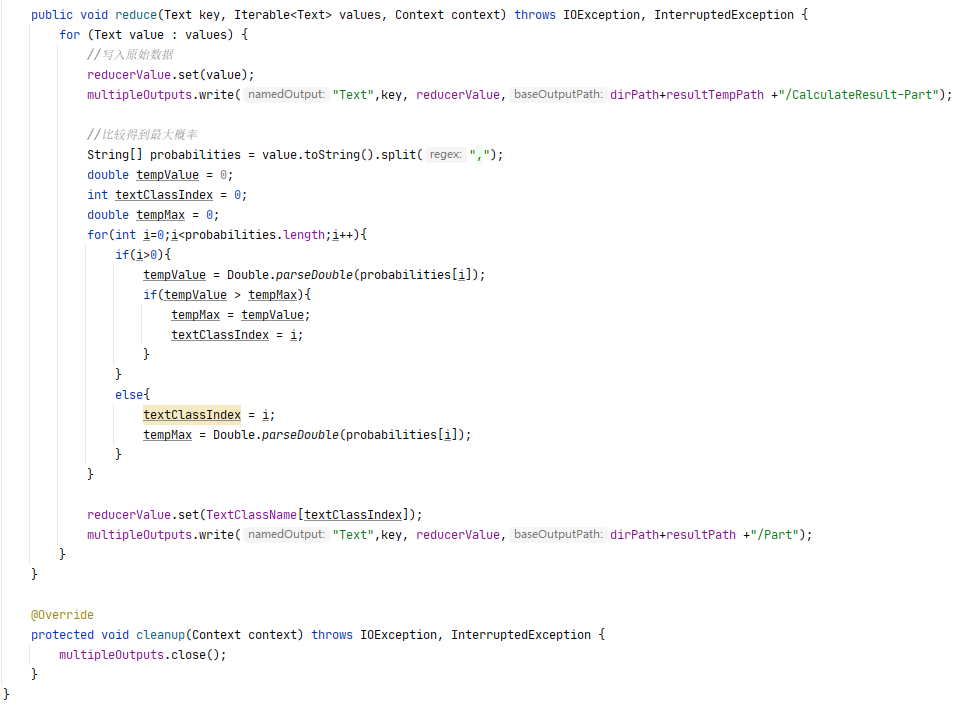






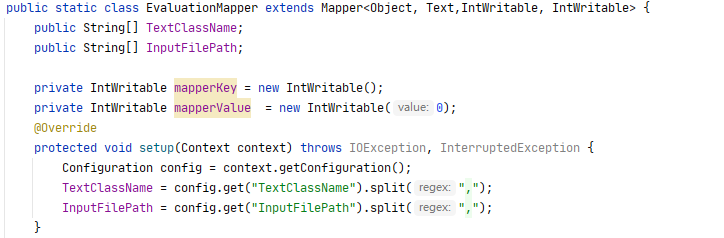
（2）Reduce

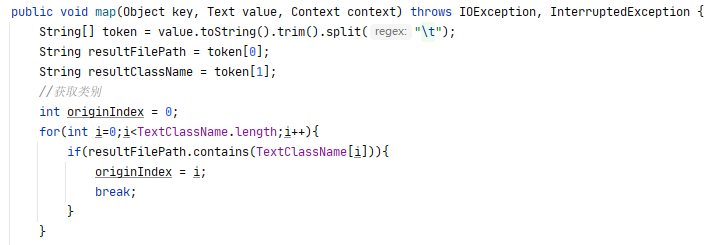


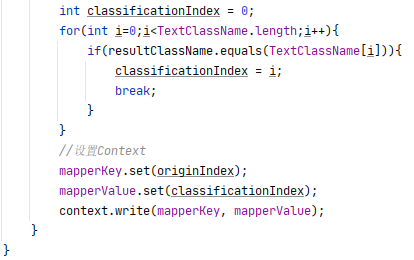


## 3.2 CalculateEvaluation

（1）Map

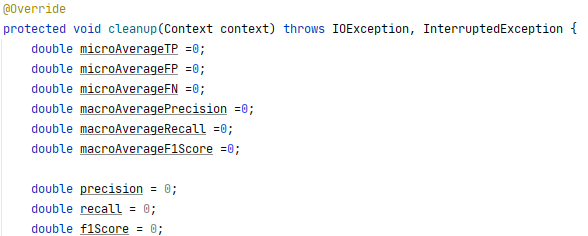


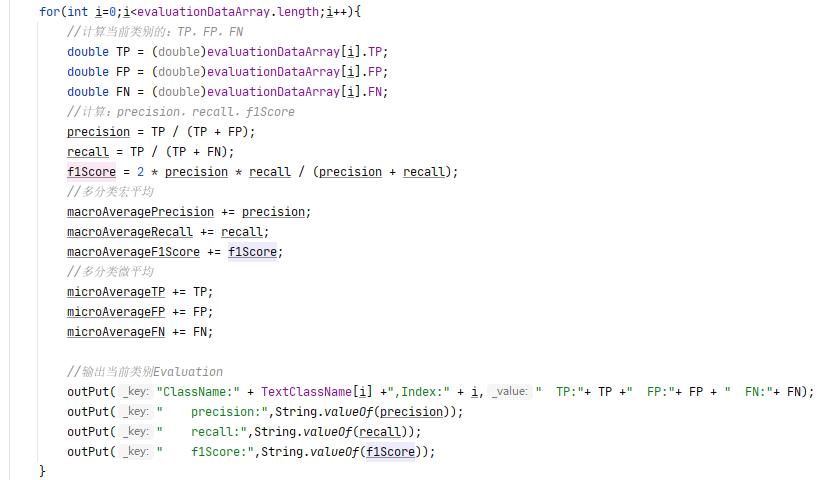




（2）Reduce









# 4.数据集说明

使用Country文件夹下的CANA，AUSTR，BRAZ三个类别进行测试。共3个类别，分别对应263，305，200个文档，总共使用了768个文档。其中按8：2的比例随机分为训练集与测试集，最终训练集为614个，测试集为154个。

表4.1 数据集说明

|  |  |  |  |
| --- | --- | --- | --- |
|  | 训练集 | 测试集 | 总计 |
| CANA | 210 | 53 | 263 |
| AUSTR | 244 | 61 | 305 |
| BRAZ | 160 | 40 | 200 |
| 总计 | 614 | 154 | 768 |

# 5.程序运行说明

当配置好环境，将数据集放到HDFS后，可运行程序，运行时直接给定Jar包名称与主类名称，不需要给定其他的命令行参数。需要注意的是必须先将配置文件与jar包放到统一目录层级中（如有特定需求可修改Json配置文件）。

命令形如：hadoop jar HadoopTest\_2.jar Classification

（1）MapReduce Job

在执行MapReduce前启动Yarn和HistoryServer，当运行完成后可查看到Job的详情。可看出三个Job分别有614个，154个，1个Map，各有1个reduce。分析认为可能是数据集小文件过多，而程序未对训练集与测试集文件进行合并或者合并成SequenceFile，因此导致Map数量较大。

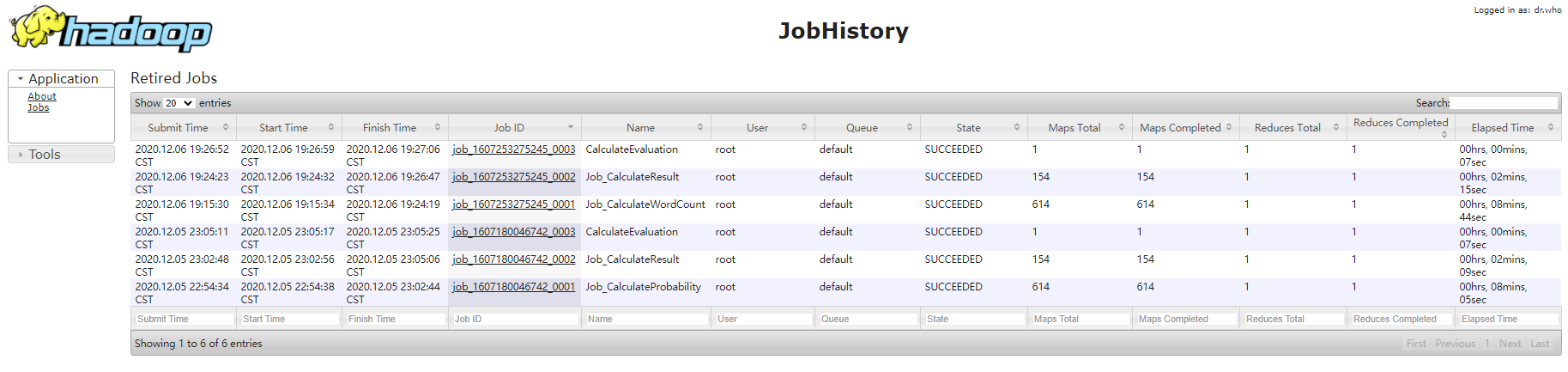


图5.1查看JobHistory

（2）作业监控截图

可通过8088端口（yarn.resourcemanager.webapp.address）查看正在运行的Job。

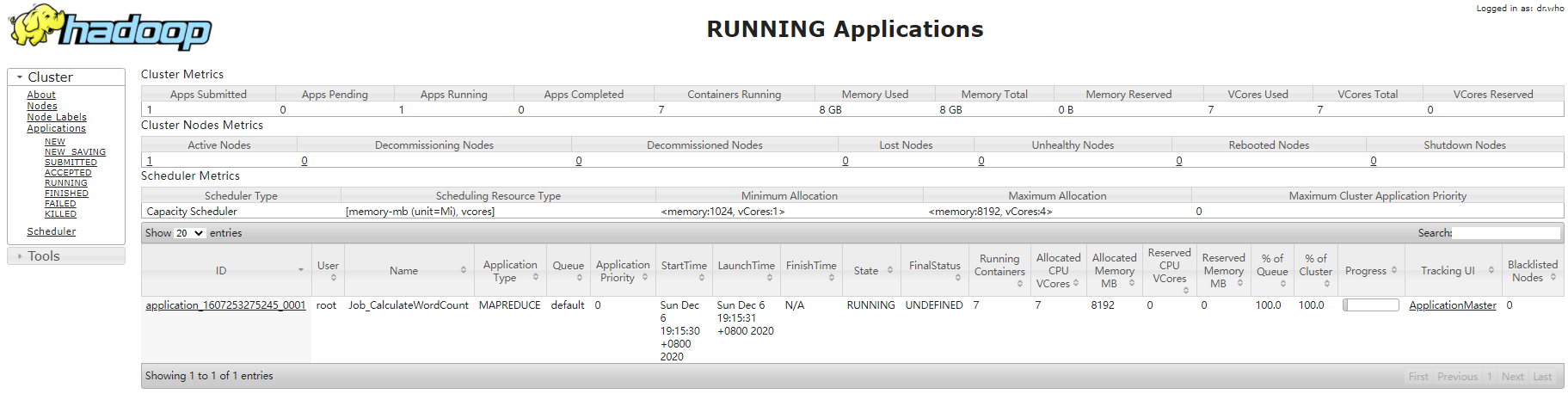


图5.2查看正在运行的Job

（3）程序运行监控截图

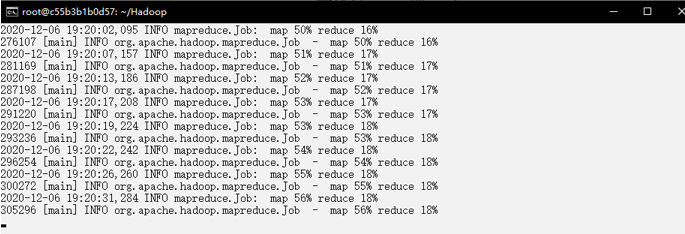


图5.3程序运行时的状态

当运行完成后可在web中看到Job执行结果。

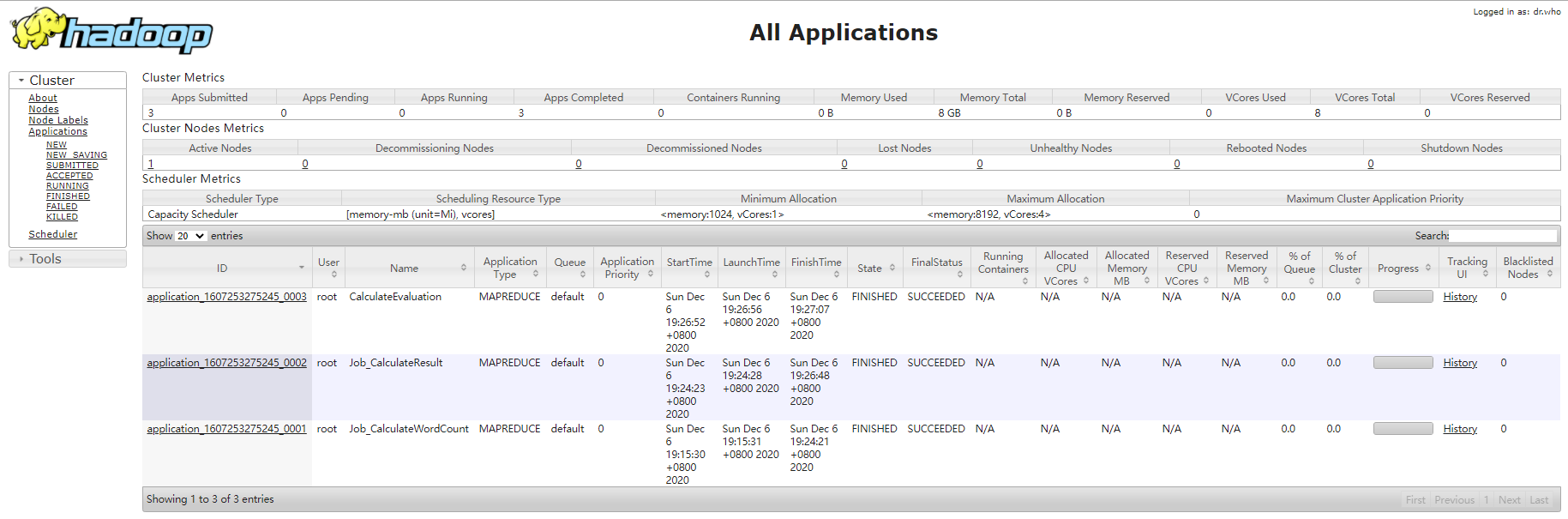


图5.4程序运行完成后Job的状态

# 6.实验结果分析

（1）统计分类结果的TP，FP，FN。

表6.1 分类后的统计信息

|  |  |  |  |
| --- | --- | --- | --- |
| 类别 | TP | FP | FN |
| CANA | 43 | 5 | 10 |
| AUSTR | 38 | 1 | 23 |
| BRAZ | 40 | 27 | 0 |
| 多分类-宏平均 | - | - | - |
| 多分类-微平均 | 121 | 33 | 33 |

（2）由于是多分类，因此分别使用了宏平均和微平均两种计算方式。可看出三个字段的值处于0.80左右。

表6.2 分类评价表

|  |  |  |  |
| --- | --- | --- | --- |
| 类别 | precision | recall | f1Score |
| CANA | 0.8958 | 0.8113 | 0.8515 |
| AUSTR | 0.9744 | 0.6230 | 0.7600 |
| BRAZ | 0.5970 | 1.000 | 0.7477 |
| 多分类-宏平均 | 0.8224 | 0.8114 | 0.7864 |
| 多分类-微平均 | 0.7857 | 0.7857 | 0.7857 |