# Project介绍

项目的大致流程如下：

1）用MapReduce算法实现贝叶斯分类器的训练过程，并输出训练模型；

2） 用输出的模型对测试集文档进行分类测试。测试过程可基于单机Java程序，也可以是MapReduce程序。输出每个测试文档的分类结果；

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

# 贝叶斯分类器理论介绍

## 2.1朴素贝叶斯分类的原理与流程

朴素贝叶斯分类是一个有监督的学习方法，也叫做**多项式朴素贝叶斯(multinomial Naïve Bayes)**或者**多项式NB(multimomial NB)**模型，它是一种基于概率的学习方法。其基本思想：对于给出的待分类项，求解在此项出现的条件下各个类别出现的概率，哪个最大，就认为此待分类项属于哪个类别。

朴素贝叶斯分类的正式定义如下：

1. 设

C://Users/zhang/AppData/Local/YNote/data/code1995@126.com/7ea39f0f568b4e108721b4f031943b87/gif.gif

为一个待分类项，其中是待分类项x的一个特征属性。

1. 有类别集合

C://Users/zhang/AppData/Local/YNote/data/code1995@126.com/656cf31e99174e4a8ac723cf47aebc28/gif.gif

,,…,是已知的类别集合

1. 计算

C://Users/zhang/AppData/Local/YNote/data/code1995@126.com/da5369d351af44eaa8be88f2e7c23e62/gif.gif

是待分类项x属于类别的概率。

1. 如果

C://Users/zhang/AppData/Local/YNote/data/code1995@126.com/df99e42a606b464e98d575bfdb9bcfeb/gif.gif

最大，则待分类项x属于类别。

现在关键的就是如何计算第三步中的各个条件概率。我们可以这样做：

1. 找到一个已知分类的待分类项集合，这个集合叫做训练样本集
2. 统计得到各类别下各个特征属性的条件概率估计，即求出：

,,…, 各项的值。

1. 假设各个特征属性是条件独立的，则根据贝叶斯定理有如下推导：

C://Users/zhang/AppData/Local/YNote/data/code1995@126.com/a5991331fdf9460183c196506ced08bb/gif.gif

因为分母对于所有类别为常数，因此只需要将分子最大化即可。又因为各特征属性是条件独立的，所以有：

C://Users/zhang/AppData/Local/YNote/data/code1995@126.com/31c1f86b69094ddbb86fa5251a108559/gif.gif

根据上述分析，朴素贝叶斯分类的流程可以由下图表示：

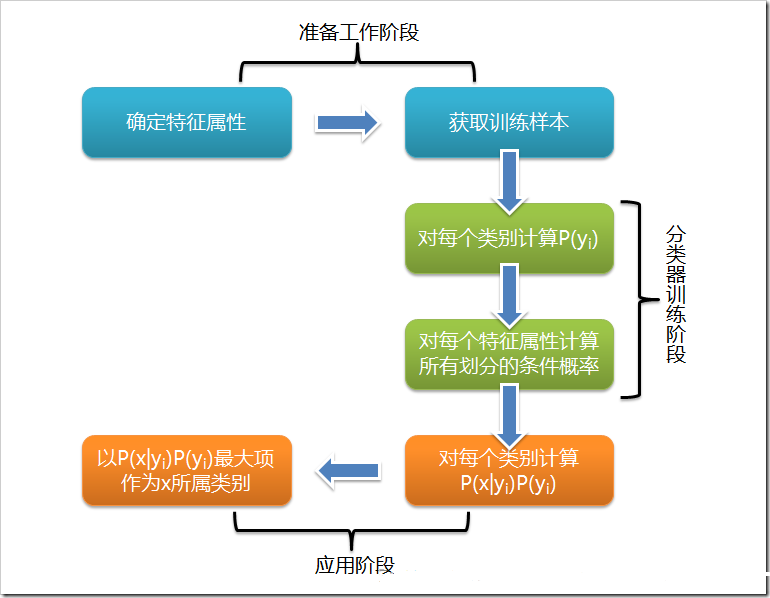


图1-1 bayes分类流程

可以看到，整个朴素贝叶斯分类分为三个阶段：

  第一阶段——准备工作阶段，这个阶段的任务是为朴素贝叶斯分类做必要的准备，主要工作是根据具体情况确定特征属性，并对每个特征属性进行适当划分，然后由人工对一部分待分类项进行分类，形成训练样本集合。这一阶段的输入是所有待分类数据，输出是特征属性和训练样本。

  第二阶段——分类器训练阶段，这个阶段的任务就是生成分类器，主要工作是计算每个类别在训练样本中的出现频率及每个特征属性划分对每个类别的条件概率估计，并将结果记录。其输入是特征属性和训练样本，输出是分类器。

  第三阶段——应用阶段。这个阶段的任务是使用分类器对待分类项进行分类，其输入是分类器和待分类项，输出是待分类项与类别的映射关系。

## 2.2朴素贝叶斯文本分类

在1.1小节中已经讲解了朴素NB的原理以及流程，接下来用朴素NB对文本进行分类。文档d属于类别c的概率的计算方法如下：

其中，是出现在类c文档中的条件概率，是文档出现在类c中的先验概率，<>是d中的词条，是d中所有词条的数目。

在文本分类中，我们的目标是找出文档最可能属于的类别。对于NB分类来说，最可能的类是具有MAP(最大后验概率)估计值的结果:

由于我们不知道参数的确切值，所以上述公式采用了从训练集中估计的参数值来代替P。对于所有的，计算其对应的条件概率的乘积，这可能会导致浮点数下界溢出。更好的方法是引入对数，从而转变成计算多个概率的对数和。因此，大多数NB在实现时所求的最大值实际是：

## 2.3 估算NB模型的参数

在1.2节中我们知道通过得到文档所属于的类别，那么如何估计参数和？我们使用最大似然估计(MLE)，它实际最后算出的是相对频率值，这些值能使训练数据的出现概率最大。

MLE估计下的类别先验概率为

其中，是训练集中c类文档所包含的文档数目，而N是训练集的文档总数。

条件概率的估计值为t在c类文档中出现的相对频率：

其中，是单词t在训练集合c类文档出现的词频，V是所有单词的集合。如果直接用上式，会发现一个问题：对没有在训练集中出现的<词频，类别>来说，其MLE估计值为0，即。这样最后导致条件概率相乘最后得到的为0，只要一旦某个词频出现概率为0，其他词频概率再高也没有意义。出现零概率的主要原因来自数据的稀疏性，即训练集合永远不可能大到所有罕见事件都能出现。

为了去掉零概率，一个简单的方法就是**加一平滑**，即在每个数字上加1：

其中，B=是词汇表中所有词项的数目。加一平滑可以认为是采用均匀分布作为先验分布(每个词项在每个类中出现一次)，然后根据训练数据进行更新得到的结果。

# Bayes分类器的MapReduce实现

## 3.1 Bayes分类器的训练流程

由第一章可知，Bayes分类器的训练过程：

-假设类别集合C=

-需要计算两种概率：N个先验概率，每个term在每类中出现的条件概率

先验概率计算公式：

其中，是训练集中c类文档所包含的文档数目，而N是训练集的文档总数。

条件概率计算公式：

其中，是单词t在训练集合c类文档出现的词频，是类别c中的单词总数，是词汇表，B=是词汇表中所有词项的数目。

## 3.2 数据的预处理

由于测试的数据集NBCorpus的文件格式是：包含两个子目录country和industry，country和industry下每个子目录就是一个类别，每个类别的子目录下一些文档很多，一些文档很少，且都是小文件。因此将那么多小文件存放在HDFS是不合理的，而且如果直接用MR程序处理这些小文件，会对每个小文件启动一个map，这就会导致很多数量的map启动，导致运行速度特别慢。

因此，需要对NBCorpus的数据进行预处理。代码中的SequenceFile.java文件作用就是序列化小文件，将数据处理成格式<<类名:文档ID>,文档内容>，文档内容的单词是以空格为分隔符的，最终将数据集处理成一个顺序文件。

## 3.3 条件概率的MR实现

计算每个单词在一个类下的条件概率，我们有三个未知参数，分别是单词t在c类文档下的词频，类别c中的全部单词总数和训练集中所有词项的数目(不重复单词的个数)B。要求这三个参数，设计了三个MR程序。

### 3.3.1 第一个MR的设计

第一个MR用于处理数据集NBCorpus序列化之后的文件，最终得到<<类名:单词>,单词出现次数>,即<<Class:word>,TotalCounts>。

输入：序列化的训练集，key为<类名:文档名>,value为文档中对应的单词，形式为<<ClassName:Doc>,word1 word2...>

输出：key为<类名:单词>,value为单词出现次数,即<<Class:word>,TotalCounts>

DataFlow示意图如下：

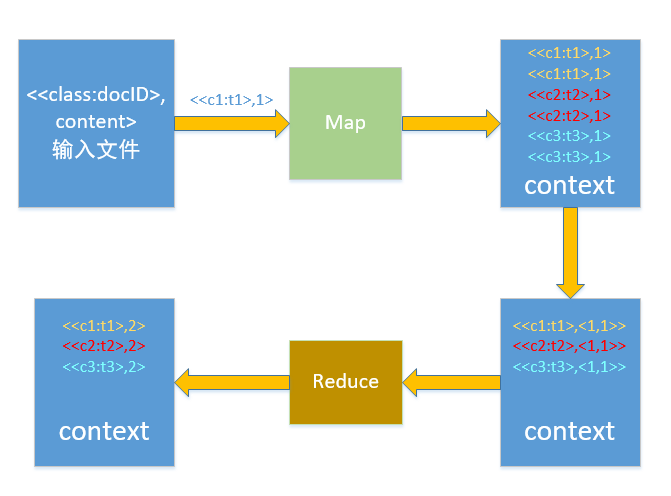


图3-1 第一个MR的DataFlow图

Map阶段：每读取一行数据，设置key是<ClassName:word>，表示的是每个类中的每个单词，value是1。

Reduce阶段：输入是来自Map阶段的输出<<ClassName:word>,<1,1,..>>

key设计为<ClassName:word>，value则是这些1的个数。最终得到序列化文件格式为：<<类名：单词>，出现次数>

### 3.3.2 第二个MR的设计

第二个MR是在第一个MR计算的基础上进一步得到每个类的单词总数<class,TotalWords>。

输入：来自第一个MR的结果，格式为<<class,word>, TotalCounts>

输出：输出的key为类名，value为单词总数，格式<class，TotalWords>

DataFlow示意图如下：

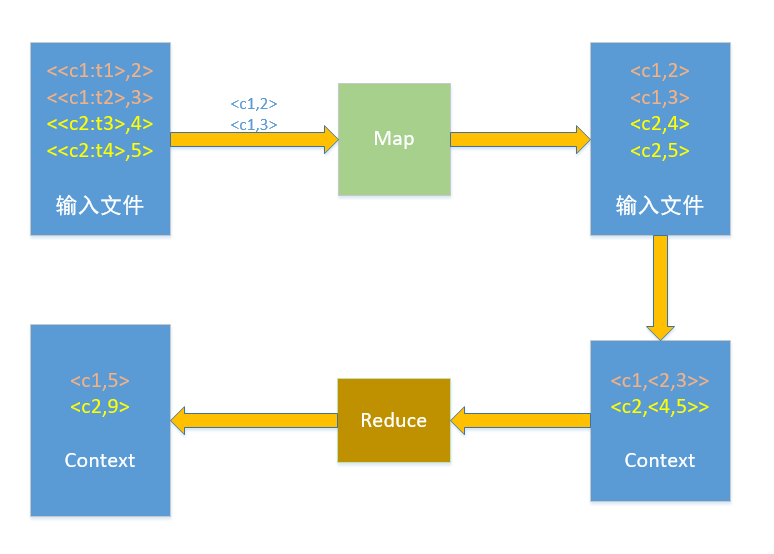


图3-2 第二个MR的DataFlow图

Map阶段：输入是<<class,word>, TotalCounts>，key是类名class，value是TotalCounts，是单词t在类c中出现的次数。

Reduce阶段：输入是来自map阶段的输出<class,<counts1,counts2,..>>

key是类名，value是将这些counts相加的结果，表示的是该类的单词总数。

### 3.3.3 第三个MR的设计

第三个MR是是在第一个MR的计算基础上得到训练集中不重复单词<word,1>，即训练集的词项表。

输入：第一个MR的结果，输入格式为为<<class:word>,counts>

输出：输出key为不重复单词，value为1，格式为<word,one>

DataFlow示意图如下：

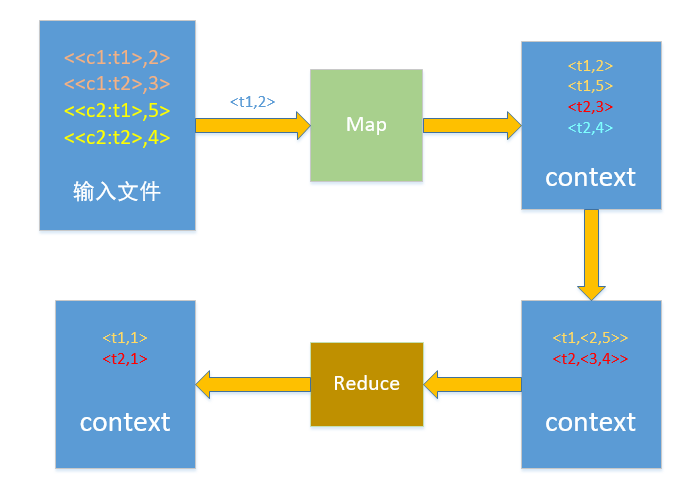


图3-3 第三个MR的DataFlow图

Map阶段：输入为<<class:word>,counts>，key为word，value为counts

Reduce阶段：输入为Map阶段的<word,<counts1,counts2,…>>，key设置为word，是训练集的词项，value设为1。

## 3.4先验概率的MR实现

第四个mapreduce主要是处理训练集序列化文件，得到每个类下的文挡数量，格式为<class,docnum>，只需要一个MR就可以算出。

输入：序列化的训练集,key为<类名:文档名>,value为文档中对应的单词.形式为<<ClassName:Doc>,word1 word2...>

输出：key为类名,value为类对应的文档数目,即<class,docnum>

DataFlow示意图如下：

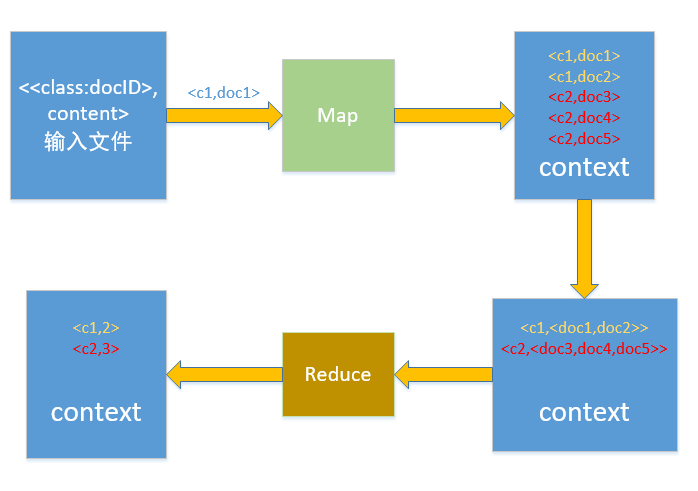


图3-4 第四个MR的DataFlow图

Map阶段：输入为训练集的序列化文件<<ClassName:Doc>,word1 word2...>，key为ClassName，value为文档名Doc

Reduce阶段：输入为来自map阶段的<ClassName,<Doc1,Doc2,…>>，key设为ClassName，value为Doc的个数。

## 3.5 预测文档的MR实现

第五个mapreduce主要是为了将序列化的测试集进行分类，输出格式为：<文档名，文档所分的类>，即<doc，class>。

在进行mapreduce之前，需要将先验概率和后验概率算出来。通过第一个、第二个、第三个MR程序得到的结果，算出每个单词在每个类下的条件概率，存放在一个HashMap<String,Double> conditionProbably，即<<类名：单词>,条件概率>。条件概率=(类c下单词tk在各个文档中出现过的次数之和+1)/（类c下单词总数+训练样本中不重复特征词总数）。需要注意的是，还需要在每个类别中加上一个没有出现过的单词概率，其格式为<类名，概率>,这个概率为1/(类c的词条总数+训练集词项总数)。

通过第四个MR程序的结果<class,docnum>，将每个类的先验概率存放于一个HashMap<String，Double> priorProbably,即<类名，先验概率>。先验概率=类c下的文件总数/训练集下的文件总数。得到先验概率和条件概率之后，就可以开始执行MR程序了。

输入：测试集的序列化文档,格式为<<ClassName:Doc>,word1,word2,…>

先验概率HashMap<String,Double> priorProbably

后验概率HashMap<String,Double> conditionProbably

输出：测试集的分类结果，格式为<文档名，文档所分的类>。

DataFlow示意图如下：

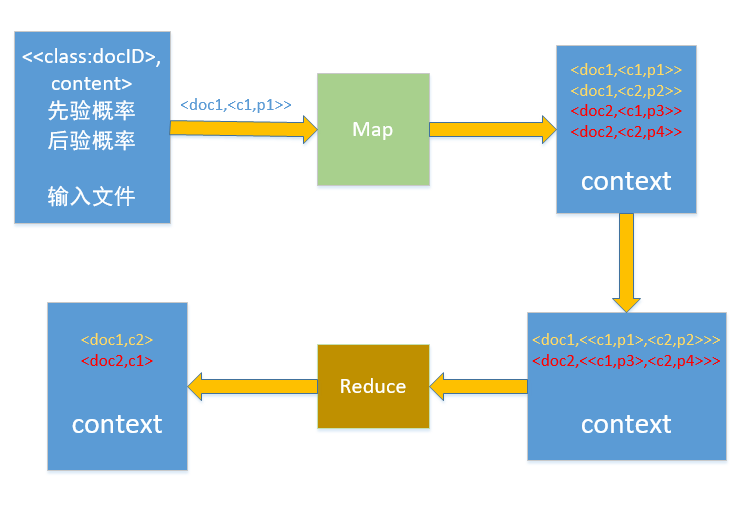


图3-5 第五个MR的DataFlow图

Map阶段：key为文档名docID，value为<class:probably>,即<类：属于该类的概率>，这个map阶段主要算出每篇文档属于每一类的概率。

Reduce阶段：输入为来自map阶段的输出<docID,<class1,probably1> < class2,probably2> …>，key为docID，value为class，即经bayes分类这篇文档被划分的类，找出使docID概率最大的那个类，作为value的值。

## 结果评测的MR程序实现

主要对分类结果进行P、R值及F值进行计算。在这里设计了两个mr程序，第一个mr是读取测试集的集合，输出真实的文档分类，输出格式为<ClassName,Doc>。第二个mr程序读取经过贝叶斯分类器分类得到的文档分类结果，然后输出bayes的文档分类<ClassName,Doc>。最后再写一个函数，分别对这两个输出进行评估。

### 第一个MR程序

这个MR程序主要是获得测试集真实的文档分类，得到<类，在该类下的文档集合>。

输入：原本的测试集数据，输入格式为<<ClassName:Doc>,word1 word2...>

输出：真实的文档分类，格式为<ClassName,Doc1;Doc2;…>，即<类名，此类下的文档>，文档之间用;隔开。

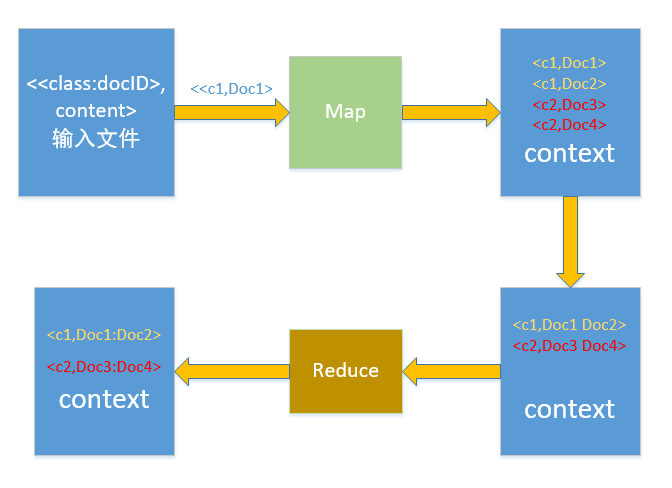


图3-6 第一个MR

Map阶段：读的每一行格式为<<class:docID>,content>,key设置为类名class，value设置为文档名docID 。

Reduce阶段：key设置位类名class，value设置为一个类下所有文档名的集合，格式为<Doc1;Doc2;Doc3>。

### 第二个MR程序

这个MR程序主要是获得经过bayes分类之后的文档分类，得到<类，在该类下的文档集合>。

输入：贝叶斯分类之后的结果分类<Doc,ClassName>

输出：将其转换为格式<ClassName,Doc1;Doc2;…>

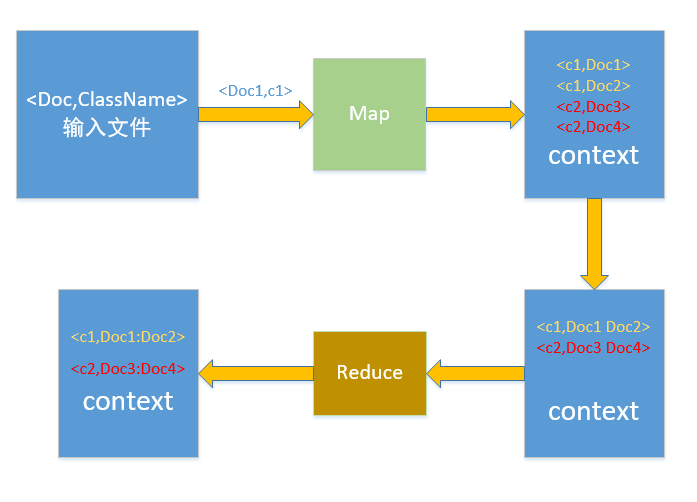


图3-7 第二个MR

Map阶段：读的每一行格式为<docID,Classname>,key设置为类名class，value设置为文档名docID 。

Reduce阶段：key设置位类名class，value设置为一个类下所有文档名的集合，格式为<Doc1;Doc2;Doc3>。

# 4 源代码清单

源代码的文件如下图所示：



图4-1 源代码清单

第一个MR程序的map、reduce代码：

/\*

\* 第一个MapReduce用于处理训练集序列化后的文件，得到<<类名:单词>,单词出现次数>,即<<Class:word>,TotalCounts>

\* 输入:args[0],序列化的训练集,key为<类名:文档名>,value为文档中对应的单词.形式为<<ClassName:Doc>,word1 word2...>

\* 输出:args[1],key为<类名:单词>,value为单词出现次数,即<<Class:word>,TotalCounts>

\*/

**public** **static** **class** ClassWordCountsMap **extends** Mapper<Text, Text, Text, IntWritable>{

**private** Text mykey = **new** Text();

**private** **final** IntWritable one = **new** IntWritable(1);

@Override

**protected** **void** map(Text key, Text value, Context context)**throws** IOException, InterruptedException {

String[] line = key.toString().split(":");//line[0]:类名 line[1]:文档名

String Class = line[0].toString();

String[] words = value.toString().split(" ");//单词之间是以空格隔开的

**for**(String word:words) {

mykey.set(Class+":"+word);

context.write(mykey,one);

}

}

}

**public** **static** **class** ClassWordCountsReduce **extends** Reducer<Text, IntWritable, Text, IntWritable>{

**private** IntWritable result = **new** IntWritable();

@Override

**protected** **void** reduce(Text key, Iterable<IntWritable> values,

Context context) **throws** IOException, InterruptedException {

**int** sum = 0;

**for**(IntWritable value:values) {

sum += value.get();

}

result.set(sum);

context.write(key, result);

}

}

第二个MR程序的map、reduce代码：

/\*\*

\* 第二个MapReduce程序处理第一个MapReduce得到的序列化文件，算出每个类的词条总数<class,TotalWords>

\* 输入：args[0],输入格式为<<class:word>,counts>

\* 输出：args[1],输出格式为<class,TotalWords>

\* **@author** zhang

\*

\*/

**public** **static** **class** ClassTotalWordsMap **extends** Mapper<Text, IntWritable, Text, IntWritable>{

**private** Text myKey = **new** Text();

@Override

**protected** **void** map(Text key, IntWritable value,Context context)

**throws** IOException, InterruptedException {

String[] line = key.toString().split(":");

myKey.set(line[0]);

context.write(myKey, value);

}

}

**public** **static** **class** ClassTotalWordsReduce **extends** Reducer<Text, IntWritable,Text, IntWritable>{

**private** IntWritable result = **new** IntWritable();

@Override

**protected** **void** reduce(Text key, Iterable<IntWritable> values,

Context context) **throws** IOException, InterruptedException {

**int** sum = 0;

**for**(IntWritable value:values) {

sum += value.get();

}

result.set(sum);

context.write(key, result);

}

}

第三个MR程序的map、reduce代码：

/\*\*

\* 第三个mapreduce处理第一个mr程序的结果，得到训练集中的每个单词<word,one>

\* 输入:args[0],输入格式为<<class:word>,counts>

\* 输出:args[1],输出key为不重复单词,value为1.格式为<word,one>

\* **@author** zhang

\*

\*/

**public** **static** **class** WordOneMap **extends** Mapper<Text, IntWritable, Text, IntWritable>{

**private** Text myKey = **new** Text();

@Override

**protected** **void** map(Text key, IntWritable value, Context context)

**throws** IOException, InterruptedException {

String[] line = key.toString().split(":");

myKey.set(line[1]);

context.write(myKey, value);

}

}

**public** **static** **class** WordOneReduce **extends** Reducer<Text, IntWritable, Text, IntWritable>{

**private** **final** IntWritable one = **new** IntWritable(1);

@Override

**protected** **void** reduce(Text key, Iterable<IntWritable> values,

Context context) **throws** IOException, InterruptedException {

context.write(key, one);

}

}

第四个MR程序的map、reduce代码：

/\*\*

\* 第四个mapreduce处理训练集序列化文件，得到<class,docnum>,即<类,该类下的文档总数>

\* 这个mr程序的主要目的是求文档的先验概率

\* 输入:args[0],序列化的训练集,key为<类名:文档名>,value为文档中对应的单词.形式为<<ClassName:Doc>,word1 word2...>

\* 输出:args[1],key为类名,value为类对应的文档数目,即<class,docnum>

\* **@author** zhang

\*

\*/

**public** **static** **class** ClassDocnumMap **extends** Mapper<Text, Text, Text, IntWritable>{

**private** Text mykey = **new** Text();

**private** **final** IntWritable one = **new** IntWritable(1);

@Override

**protected** **void** map(Text key, Text value, Mapper<Text, Text, Text, IntWritable>.Context context)

**throws** IOException, InterruptedException {

String[] line = key.toString().split(":");

mykey.set(line[0]);

context.write(mykey, one);

}

}

**public** **static** **class** ClassDocnumReduce **extends** Reducer<Text, IntWritable, Text, IntWritable>{

**private** IntWritable result = **new** IntWritable();

@Override

**protected** **void** reduce(Text key, Iterable<IntWritable> values,

Reducer<Text, IntWritable, Text, IntWritable>.Context context) **throws** IOException, InterruptedException {

**int** sum = 0;

**for**(IntWritable value:values) {

sum += value.get();

}

result.set(sum);

context.write(key, result);

}

}

第五个MR程序的map代码：

**public** **static** **class** PredictMap **extends** Mapper<Text, Text, Text, Text>{

@Override

**protected** **void** setup(Mapper<Text, Text, Text, Text>.Context context) **throws** IOException, InterruptedException {

*GetPriorProbably*();

*GetConditionProbably*();

}

**private** Text myKey = **new** Text();

**private** Text myValue = **new** Text();

@Override

**protected** **void** map(Text key, Text value, Context context)

**throws** IOException, InterruptedException {

String[] line = key.toString().split(":");

myKey.set(line[1]);//key为文档名<docID>

//遍历所有类，算出该文档在每一个类中的概率，map的输出格式为<docID,<类,概率>>

**for**(Map.Entry<String, Double> entry:*priorProbably*.entrySet()) {

String className = entry.getKey();//得到类名

**double** tmpValue = Math.*log*(entry.getValue());//降维，防止相乘浮点数溢出

String[] words = value.toString().split(" ");

**for**(String word:words) {//内层遍历，对每个单词的属于当前类的概率进行log相加

String tmpKey = className + ":" + word;

//使用conditionProbably查找条件概率，log相加

**if**(*conditionProbably*.containsKey(tmpKey)) {

tmpValue += Math.*log*(*conditionProbably*.get(tmpKey));

}**else** {

//若测试集的单词在训练集未出现，则加上之前在条件概率中算的概率

tmpValue += Math.*log*(*conditionProbably*.get(className));

}

}

myValue.set(className + ":" + tmpValue);

context.write(myKey, myValue);

}

}

}

第五个MR程序的reduce代码：

**public** **static** **class** PredictReduce **extends** Reducer<Text, Text, Text, Text>{

**private** Text myValue = **new** Text();

//reduce阶段收到的格式为：<<docID>,<类1:概率1> <类2:概率2>...>

@Override

**protected** **void** reduce(Text key, Iterable<Text> values,Context context)

**throws** IOException, InterruptedException {

**double** maxProbably = 0.0;

**boolean** flag = **true**;

String tmpClass = **null**;

**for**(Text value:values) {

String[] line = value.toString().split(":");

**if**(flag) {//表明是第一次进入循环

tmpClass = line[0];

maxProbably = Double.*parseDouble*(line[1]);

flag = **false**;

}**else** {

**if**(maxProbably < Double.*parseDouble*(line[1])) {

//若概率变大，则更新maxProbably

tmpClass = line[0];

maxProbably = Double.*parseDouble*(line[1]);

}

}

}

myValue.set(tmpClass);

context.write(key, myValue);

}

}

最后两个MR程序的map、reduce代码：

/\*\*

\* 得到原本的文档分类

\* 输入:初始数据集合,格式为<<ClassName:Doc>,word1 word2...>

\* 输出:原本的文档分类，即<ClassName,Doc>

\*/

**public** **static** **class** OriginalDocOfClassMap **extends** Mapper<Text, Text, Text, Text>{

**private** Text newKey = **new** Text();

**private** Text newValue = **new** Text();

**public** **void** map(Text key, Text value, Context context) **throws** IOException, InterruptedException{

String[] line = key.toString().split(":");

newKey.set(line[0]);

newValue.set(line[1]);

context.write(newKey, newValue);

}

}

/\*\*

\* 得到经贝叶斯分分类器分类后的文档分类

\* 读取经贝叶斯分类器分类后的结果文档<Doc,ClassName>,并将其转化为<ClassName,Doc>形式

\*/

**public** **static** **class** ClassifiedDocOfClassMap **extends** Mapper<Text, Text, Text, Text>{

**public** **void** map(Text key, Text value, Context context) **throws** IOException, InterruptedException{

context.write(value, key);

}

}

**public** **static** **class** Reduce **extends** Reducer<Text, Text, Text, Text>{

**private** Text result = **new** Text();

**public** **void** reduce(Text key, Iterable<Text> values, Context context) **throws** IOException, InterruptedException{

//生成文档列表

String fileList = **new** String();

**for**(Text value:values){

fileList += value.toString() + ";";

}

result.set(fileList);

context.write(key, result);

}

}

# 5 数据集说明

数据集主要用了NBCorpus\Country\目录下的ALB、ARG、AUSTER、BELG、BRAZ、CANA六个类进行训练与测试。训练集与测试集大小如下表所示：

|  |  |  |
| --- | --- | --- |
| 文件名 | 训练集 | 测试集 |
| ALB（81） | 60 | 21 |
| ARG(108) | 80 | 28 |
| AUSTER(305) | 220 | 80 |
| BELG(154) | 110 | 44 |
| BRAZ(200) | 150 | 50 |
| CANA(263) | 200 | 63 |

# 6 程序运行说明

程序运行时基于Web页面的作业监控截图如下图所示：

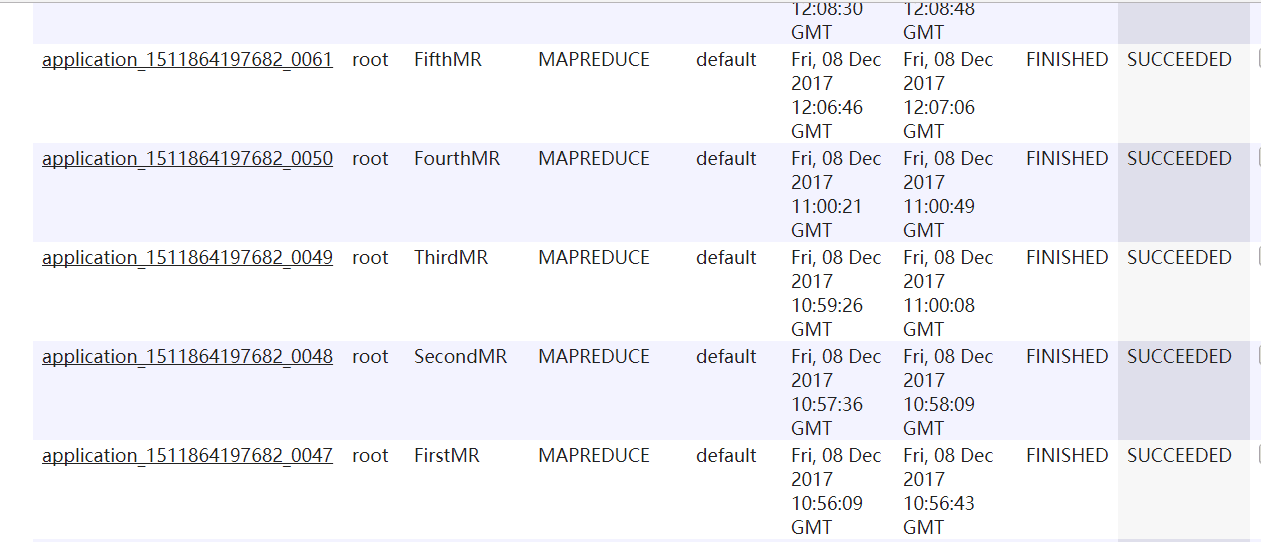


图6-1 分类作业监控

进行分类结果评测的作业监控截图如下所示：

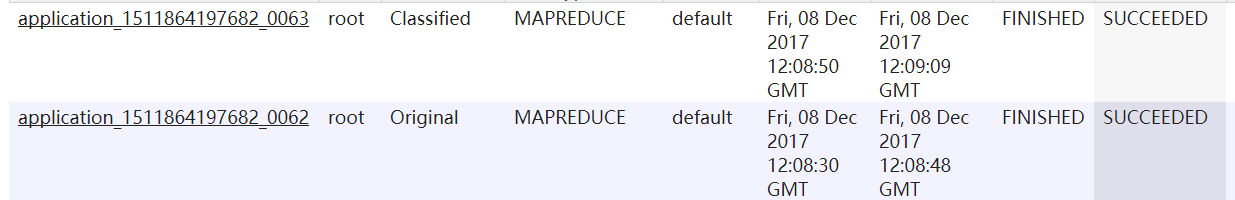


图6-2 结果评测的作业监控

一共运行了7个mapreduce任务，前三个mr程序是为了算出条件概率，第四个是为了算出文档的先验概率，第五个是将前四个得到的结果对测试集进行预测分类。而最后两个mapreduce程序是将评估bayes分类的效果，计算P、R值及F值，MacroAverage和MicroAverage。

# 7 实验结果分析

最后评估各个类的P、R以及F值的运行结果如下图所示：

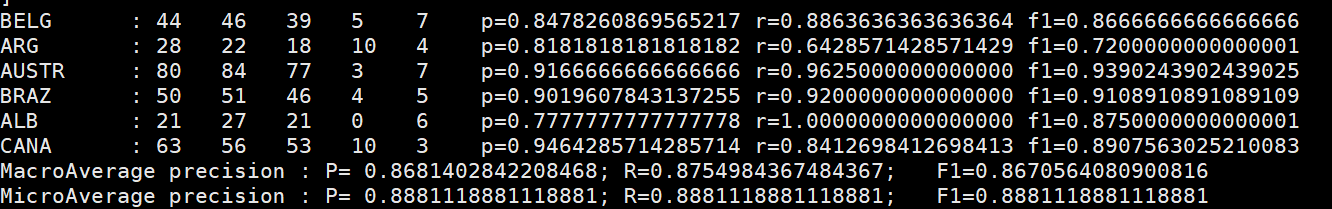


图 7-1 运行结果

对应表格如下(保留2位小数点)：

|  |  |  |  |
| --- | --- | --- | --- |
| BELG | Precision | Recall | F值 |
| ARG | 0.85 | 0.89 | 0.87 |
| AUSTR | 0.82 | 0.64 | 0.72 |
| BRAZ | 0.92 | 0.96 | 0.94 |
| ALB | 0.78 | 1.0 | 0.88 |
| CANA | 0.95 | 0.84 | 0.89 |
| MacroAverage | 0.87 | 0.88 | 0.87 |
| MicroAverage | 0.89 | 0.89 | 0.89 |