

# 基于 MapReduce 的软件 Bug 分类

张铭伟 19214895、杨金瑞 19214876、任嘉昕 19214836、陈俊霖 19214945

(中山大学数据科学与计算机学院 广州 510006)

**摘要** 在 Github 代码仓库中, 存在大量已分类 (即加上标签) 的软件 bug。但是现在分类标签大都是基于人工添加的, 效率比较低。为了提高分类效率, 节省人力资源, 使得在海量数据的情况下, 依然能够快速给软件 bug 执行自动分类, 我们利用 MapReduce 分布式编程模型, 实现了朴素贝叶斯分类算法, 自动地给 Bug 加上标签。实验结果表明, 我们的算法能够有效地执行分类功能。我们的代码会在 [Github](#) 上公开。

**关键词** 朴素贝叶斯分类算法、MapReduce、分布式、软件 bug

## Software Bug Classification Based on MapReduce

ZHANG Jia-Wei 19214895、YANG Jin-Rui 19214876、REN Jia-Xin 19214836、CHEN Jun-Lin 19214945

(School of Data and Computer Science, Sun Yat-Sen University, Guangzhou 510006)

**Abstract** There are a large number of classified (i.e., tagged) software bugs in the Github code repository. But now most of the classification labels are manually added, and the efficiency is relatively low. In order to improve the classification efficiency and save human resources, so that software bugs can still be automatically classified automatically in the case of massive data, we use the MapReduce distributed programming model to implement the Naive Bayes classification algorithm and automatically add bugs to the bugs. label. Experimental results show that our algorithm can effectively perform classification functions. Our code will be released on [Github](#).

**Key words** Naive Bayes classification algorithm; MapReduce; distributed; software bug

## 1 问题描述

在 Github 代码仓库中, 有一个 issues 栏目, 这个栏目的目的是收集用户对该开源代码的反馈, 即用户会提出软件 bug。目前的 Github issues 中存在大量已分类 (即加上标签) 的软件 bug (如图 1 中红框所示)。

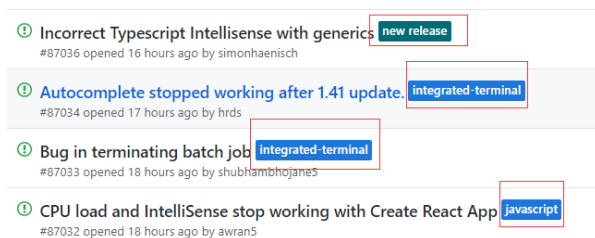


图 1. Github 上的软件 bug 示例

然而, 这些分类标签都是人为手工添加的, 当 issues 的数量很大的时候, 人工添加标签的效率就显得极为低下。

为了解决这个问题, 我们设计了基于 MapReduce 并行计算模型的朴素贝叶斯分类方法, 使得计算机可以通过分布式计算根据 issue 的标题快速地给软件 bug 进行分类。

## 2 实验原理

### 2.1 朴素贝叶斯分类算法

本次实验的问题, 其实是一个文本分类的问

题, 我们将问题简化成对每个 issue 的标题进行分类(加上标签)。解决这样的问题可以使用朴素贝叶斯算法[1]。

用朴素贝叶斯去解决分类和回归问题首先要知道贝叶斯定理[2], 贝叶斯定理主要是提供了一个算法解决这样一个问题: 已知某条件概率, 如何求事件交换后的条件概率, 即已知  $P(A|B)$ , 求  $P(B|A)$ 。定理如下:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \quad (1)$$

其中:

$$P(A|B) = \frac{P(AB)}{P(B)} \quad (2)$$

上式中  $P(A|B)$  表示事件 B 已经发生的前提下事件 A 发生的概率,  $P(AB)$  表示 A 和 B 同时发生的概率。而当 A 和 B 两个事件是互斥的, 完全独立的没有交集的两件事, 此时  $P(AB) = P(A)P(B)$ , 这时就是朴素的。

朴素贝叶斯求解分类问题就是对于给出的待分类项, 算出它属于每个标签的概率, 哪个概率最大, 就是哪个标签。首先我们要求出每个样本的特征向量  $x$ , 对给定的样本特征  $x$ , 样本属于类别  $ei$  的概率是

$$P(ei|x) = \frac{P(x|ei)P(ei)}{x} \quad (3)$$

假设  $x$  的维数是  $K$  (该文本有  $K$  个单词), 由于是朴素的, 特征条件互斥, 根据全概率公式有:

$$P(ei|x) = \frac{P(ei) \prod_{j=1}^K P(x_j|ei)}{P(x)} \quad (4)$$

由于对于同一个文本来讲,  $P(x)$  是一样的, 所以预测它属于哪个分类, 只要算上式的分子部分即可。对于一个给定的测试样本, 分子部分常用的有两种模型:

#### 1. 伯努利模型(Bernoulli Model):

$$P(x_j|ei) = \frac{N_{ei}(x_j)}{N_{ei}} \quad (5)$$

$$P(ei) = \frac{N_{ei}}{N} \quad (6)$$

其中  $N_{ei}$  表示有多少个训练集文本的标签是  $ei$ ,  $N_{ei}(x_j)$  表示特征  $x_j$  出现在了多少个标签为  $ei$  的样本中。  $N$  表示训练集总样本数。在这个模型中特征向量  $x$  是一个  $K$  维 01 向量, 每一维表示某个单词是否出现在当前文本里。

#### 2. 多项式模型(Multinomial Model):

$$P(x_j|ei) = \frac{NW_{ei}(x_j)}{N_{ei}} \quad (7)$$

$$P(ei) = \frac{N_{ei}}{N} \quad (8)$$

其中  $NW_{ei}(x_j)$  表示在训练集中, 标签为  $ei$  的文本中, 单词  $x_j$  出现了多少次 (一篇文章中出现多次按多次计算),  $NW_{ei}$  表示训练集中标签为  $ei$  的文本中包含多少个单词(不去重)。

从二者的公式我们可以看出, 伯努利模型是以文本为粒度, 多项式模型是以单词为粒度, 伯努利模型相比多项式模型会丢失词频信息, 这会造成分类能力下降, 所以我们这次实验采用的是多项式模型。

当测试文本中存在一个单词  $x$ ,  $x$  是训练集中出现过的单词, 但是  $x$  没有在标签为  $ei$  的训练集文本中出现过, 那么此时  $P(x_j|ei) = 0$ , 从而导致  $P(ei) \prod_{j=1}^K P(x_j|ei) = 0$ 。此时, 我们可以通过平滑解决这个问题。增加了平滑后的多项式模型公式如下:

$$P(x_j|ei) = \frac{NW_{ei}(x_j) + \alpha}{NW_{ei} + \alpha * \text{totalwords}} \quad (9)$$

在上式中,  $0 < \alpha \leq 1$ , 当  $\alpha = 1$  时, 称为拉普拉斯平滑; 当  $0 < \alpha < 1$  时, 称为 Lid stone 平滑。  $\text{totalwords}$  表示训练集中有多少种不同的单词数。在本次实验中, 我们采用拉普拉斯平滑, 通过该平滑公式, 我们保证了每一项  $P(x_j|ei)$  不为 0。

另外, 当  $K$  很大的时候,  $\prod_{j=1}^K P(x_j|ei)$  趋近于 0, 在计算机里可能会造成浮点数溢出导致结果为 0, 这个时候我们可以对每一项  $P(x_j|ei)$  求对数, 变乘为加:

$$P(ei|x) = \log(P(ei)) + \sum_{j=1}^K \log(P(x_j|ei)) \quad (10)$$

## 2.2 Hadoop简介

Hadoop 是 Apache 软件基金会所开发的并行计算框架与分布式文件系统, 最核心的部分就是 HDFS 和 MapReduce。

### 2.2.1 HDFS

HDFS 是 Hadoop 分布式文件系统 (Hadoop Distributed File System) 的缩写, 为分布式计算存储提供了底层支持。采用 JAVA 语言开发, 可以部署在多种普通的廉价机器上, 以集群处理数量积达到大型主机处理性能。

HDFS 采用 master/slave 架构。一个 HDFS 集

## HDFS Architecture

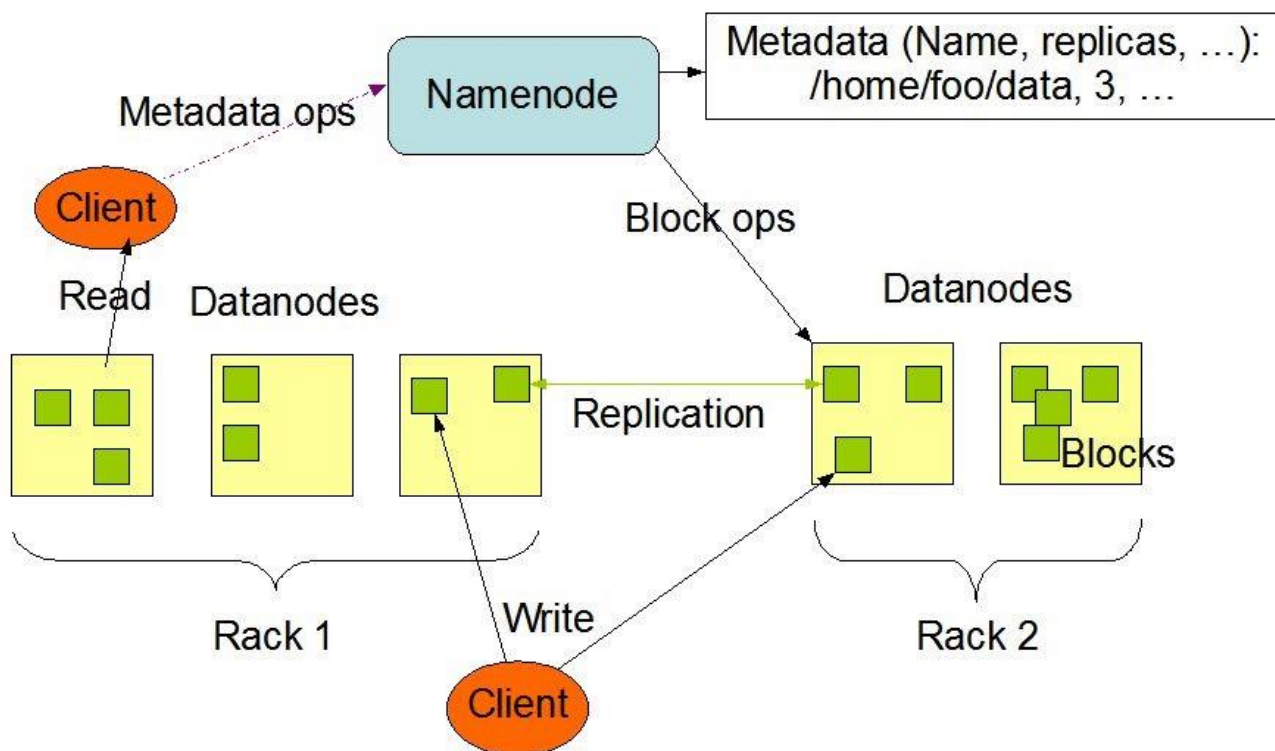


图 2. HDFS 文件系统架构图<sup>[3]</sup>

群包含一个单独的 NameNode 和多个 DataNode。NameNode 作为 master 服务，它负责管理文件系统的命名空间和客户端对文件的访问。NameNode 会保存文件系统的具体信息，包括文件信息、文件被分割成具体 block 块的信息、以及每一个 block 块归属的 DataNode 的信息。对于整个集群来说，HDFS 通过 NameNode 对用户提供了一个单一的命名空间。

DataNode 作为 slave 服务，在集群中可以存在多个。通常每一个 DataNode 都对应于一个物理节点。DataNode 负责管理节点上它们拥有的存储，它将存储划分为多个 block 块，管理 block 块信息，同时周期性的将其所有的 block 块信息发送给 NameNode。

HDFS 文件系统架构图如图 2 所示。文件写入时，Client 向 NameNode 发起文件写入的请求，NameNode 根据文件大小和文件块配置情况，返回给 Client 它所管理部分 DataNode 的信息，Client 将文件划分为多个 block 块，并根据 DataNode 的地址信息，按顺序写入到每一个 DataNode 块中。

文件读取时，Client 向 NameNode 发起文件读取的请求。NameNode 返回文件存储的 block 块信息、及其 block 块所在 DataNode 的信息。Client 读取文件信息。

### 2.2.2 MapReduce 编程模型

MapReduce 是一种用于大规模数据集（大于 1TB）的并行计算编程模型，它的宗旨是让不懂并行编程的人也能写出并行代码。它的主要思想就是“Map（映射）”和“Reduce（规约）”，是借鉴了函数式编程语言和矢量编程语言的特性。

MapReduce 编程模型的数据流如图 3 所示。我们不深究 MapReduce 的具体底层实现，简单来说它就分为 3 个步骤：

1. Map(映射)
2. Shuffle (排序)
3. Reduce(规约)

为了让 MapReduce 程序运行，我们首先要将所有输入数据组织成键值对（key, value）的形式并存放在 hdfs 的文件系统里。

# Map/Reduce Dataflow

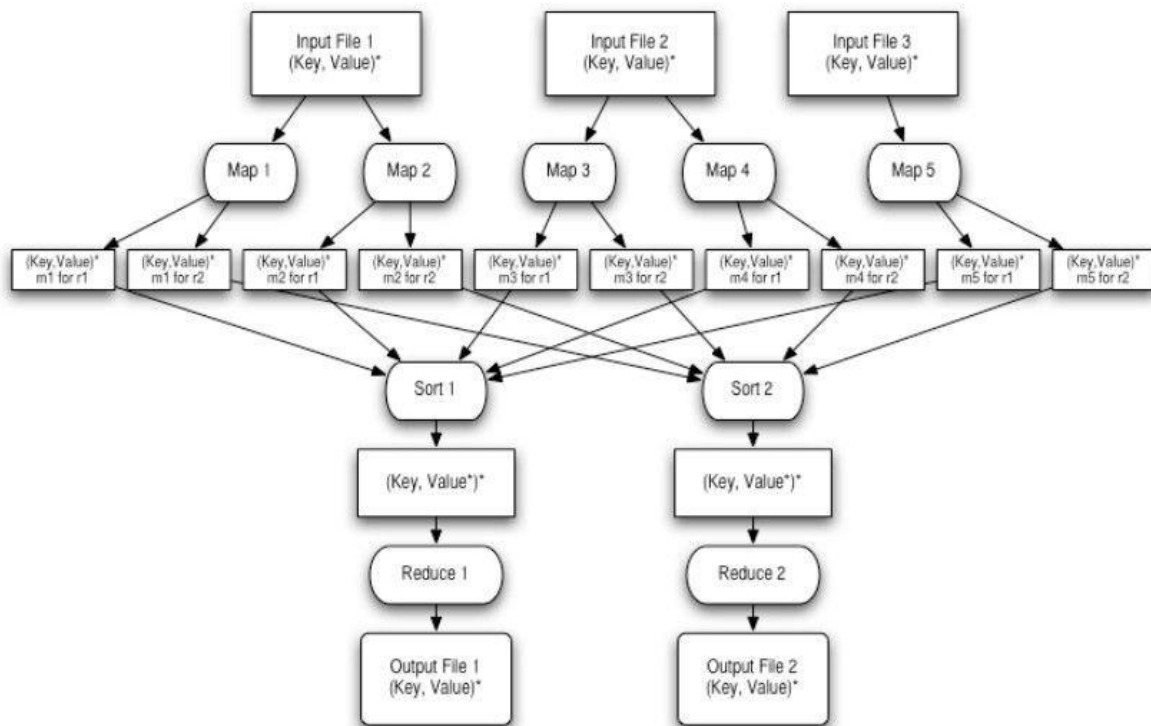


图 3. MapReduce 模型数据流<sup>[3]</sup>

一个常用的方法是将行作为一个 **key**，将行的内容作为 **value**，那么每一次，输入会按行切开成一个一个 **split**，每一个输入 **split** 会创建一个计算任务来调用我们定义好的 **Map** 函数进行相应的处理，最后将结果以新的(**key**, **value**)形式输出。具体如何定义 **key** 和 **value** 要取决于用户希望下一步作什么处理。所以 **Map** 这一步就是将输入映射成新的键值对输出。

在进入 **Reduce** 步骤之前，会先进行一次 **Shuffle** 操作。**Shuffle** 不需要用户去实现，它的主要功能就是对 **Map** 的输出进行一个排序，将相同键值的键值对合并到一起传给 **Reduce**。

**Reduce** 接收 **shuffle** 后的结果作为输入，在这里 **Reduce** 的输入实际上是(**key**, **List<value>**)，即将相同 **key** 的所有 **value** 合并成一个数组，这个数组整体作为一个 **value**，然后传入到 **Reduce** 中，那么在 **Reduce** 中，我们就可以一次性对同一个 **key** 的所有 **value** 进行处理，最后将结果还是以

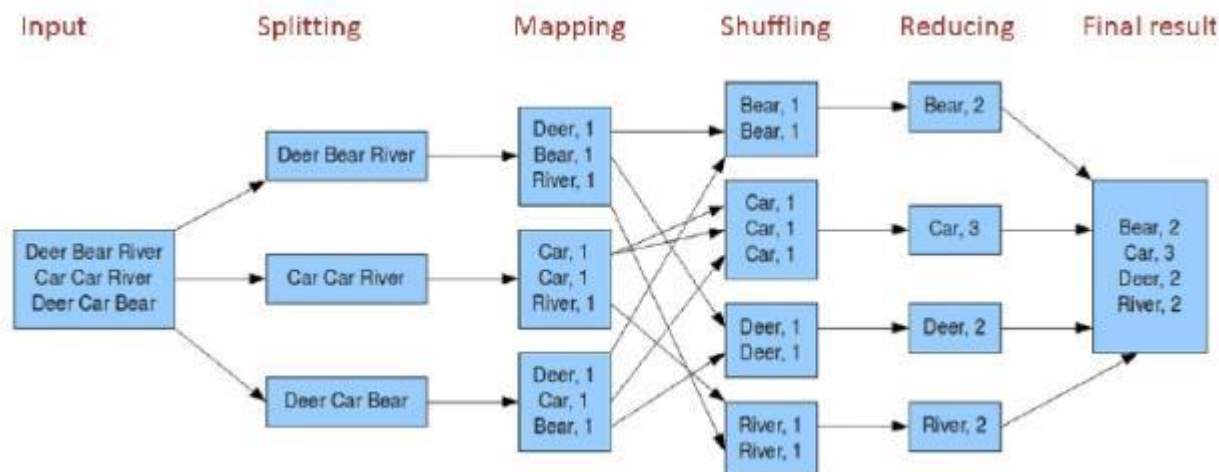
(**key**, **value**) 的形式输出。

为了更好地理解 **MapReduce** 模型，一个实例如图 4 所示。这个实例是统计每个单词出现了多少次，在 **Map** 阶段对每行每遇到一个单词就输出一个(单词, 1)的键值对，在 **reduce** 阶段将同一个单词为 **key** 的 **key-value** 对的 **value** 值相加（因为这些 **value** 值都是 1），然后就可以得到 **key** 这个单词出现了多少次，最后输出。

## 2.3 使用python开发MapReduce程序

传统的 **MapReduce** 程序应该是用 **JAVA** 语言写的，但我们对 **JAVA** 不是很熟练，反而对 **python** 比较熟悉。经过查找资料，我们找到了使用 **python** 开发 **MapReduce** 程序的方法[5]。

**Hadoop Streaming** 提供了一个便于进行 **MapReduce** 编程的工具包，使用它可以基于一些脚本语言或者其他编程语言来实现 **Map** 和 **Reduce** 过程，从而充分利用 **Hadoop** 并行计算框架的优势

图 4. 利用 MapReduce 统计单词数<sup>[4]</sup>

来处理大数据。

Streaming 方式是基于 Unix 系统的标准输入输出来进行 MapReduce 任务的运行，任何支持标准输入输出特性的编程语言都可以使用 Streaming 方式来实现 MapReduce 程序。

在利用 Hadoop Streaming 实现的 MapReduce 程序中，我们姑且将实现 Map 功能和 Reduce 功能的程序成为 mapper 和 reducer。mapper 和 reducer 会从标准输入中读取用户数据，一行一行处理后发送给标准输出。mapper 任务运行时，它把输入切分成行并把每一行提供给可执行文件进程的标准输入。同时，mapper 收集可执行文件进程标准输出的内容，并把收到的每一行内容转化成(key, value)对，作为 mapper 的输出。默认情况下，一行中第一个制表符(\t)之前的部分作为 key，之后的(不包括\t)作为 value。如果没有制表符，整行作为 key 值，value 值为 null。

因此我们使用 python 开发 MapReduce 代码的思路就清晰了。我们只需要用 python 实现 mapper 和 reducer 即可。在每一个代码中，使用 python 的 sys.stdin 来读入标准输入的数据，使用 sys.stdout 来输出数据就可以了，当然对于输入和输出数据都要处理成“key\t value”这样的形式。最后我们直接调用 Hadoop Streaming 工具包来运行我们的 python 程序即可。

### 3 模型设计

#### 3.1 朴素贝叶斯分类算法变体

根据公式 8 和公式 9，我们整理得到使用朴素贝叶斯分类算法进行软件 bug 分类需要计算的中间变量表：

表 1. 朴素贝叶斯分类需计算的中间变量表

变量名	意义
N	训练集文章总数
$N_{Wei[label][x_j]}$	训练集中标签为 label 的标题中单词 $x_j$ 的出现次数
$NW[label]$	训练集中标签为 label 的 title 中包含了几个单（不去重）
$Nei[label]$	训练集中标签为 label 的标题总数
totalwords	训练集中一共有多少个不同的单词

对于一个标签为 label 的标题中的所有单词可以分成三类：

- A. 在训练集中标签为 label 的标题里出现过
- B. 在训练集中标签为 label 的标题里没出现过
- C. 在训练集所有标题里都没有出现过。

对于 C 类的单词我们按传统的做法直接舍弃，当它没有出现过。对于 A 类和 B 类单词我们按照公式 9 计算相应概率。



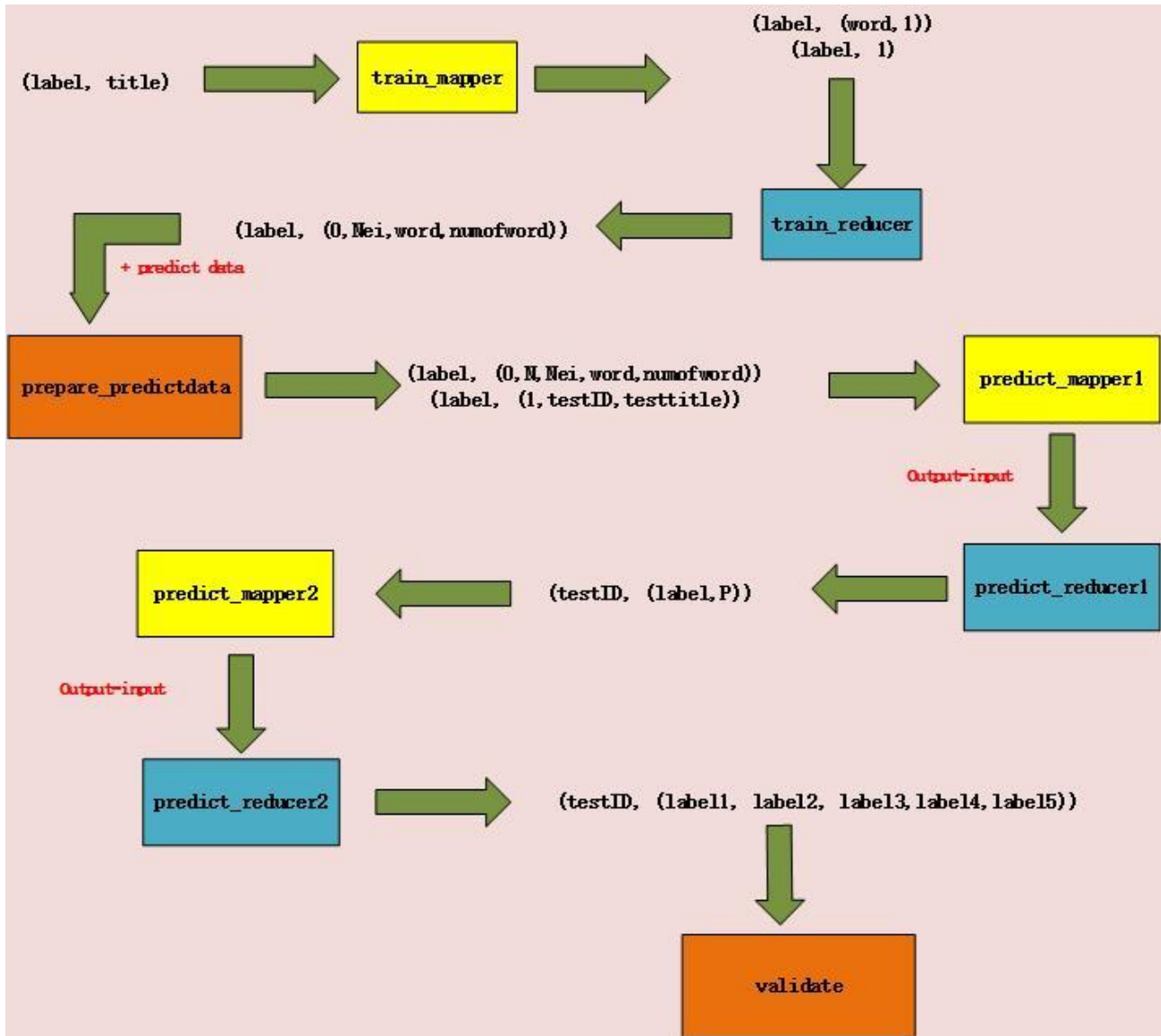


图 5. 基于 MapReduce 的朴素贝叶斯分类算法总体框图

现在问题是平滑公式中的  $\text{totalwords}$  是比较难以统计的，就算我们能够统计，这个数字也是比较大，那么计算出来的概率就会很小，连乘之后容易溢出，我们可以将其改为  $N$ ，一方面  $N$  远没有  $\text{totalwords}$  大，另一方面  $N$  也比较容易知道方便编程。同时这样更改不会影响预测概率之间的相对大小，也就不会影响正确性，这条式子我们称为朴素贝叶斯分类算法变体：

$$P(x_j|e_i) = \frac{NW_{ei}(x_j) + \alpha}{NW_{ei} + \alpha * N} \quad (11)$$

### 3.2 基于 MapReduce 的朴素贝叶斯分类算法

由于 Github 上一条 issue 可以有多个标签，由于我们目前无法做到多标签分类，于是我们将

问题简化，将每个 issue 的标签设为其第一个标签。基于此我们设计的基于 MapReduce 的朴素贝叶斯分类算法总体框图如图 5 所示。

我们经过分析，认为无法只使用一对 MapReduce 程序完成训练和分类，因此我们将训练和预测过程分成不同的 MapReduce 程序去执行，如图 5 所示，我们的代码总有 `train_mapper`、`train_reducer`、`prepare_predictdata`、`predict_mapper1`、`predict_reducer1`、`predict_mapper2`、`predict_reducer2`、`validate` 共 8 个部分，下面我们每一个部分逐一介绍。

#### 3.2.1 训练过程----`train_mapper` 和 `train_reducer`

我们将训练数据按  $(\text{label}, \text{title})$  的形式预先

整理到一个文本中，训练过程主要是统计每种 label 有多少个训练样本以及每个单词在某个 label 的样本中出现了多少次。

在 `train_mapper` 程序中，遍历每一个文本，对每一文本中的每个单词，输出键值对(`label`, (`word`, 1))，意思是标签为 `label` 的 `title` 中，单词 `word` 出现了 1 次，最后遍历完一个文本的所有单词之后，输出键值对(`label`, 1)，意思是标签为 `label` 的 `title` 出现了 1 次，代码如下：

```
import sys
"""
输入:
label title
输出:
<label, <word, 1>>
<label, 1>
Nei[label]: 标签为label的title数
"""
for row in sys.stdin:
    label, title = row.split('\t\t') # 我这里数据以两个制表符分割key和value
    title = title.split()
    flag = False
    for word in title:
        key_value = "{}\t\t{}".format(label, word + ' ' + str(1))
        print(key_value)
    print("{}\t\t{}".format(label, str(1)))
```

在 `train_reducer` 程序中，对 `train_mapper` 提供的相同 label 的键值对统一处理。这里输入会有两种情况，分别为(`label`, (`word`, 1))和(`label`, 1)，我们可以通过判断 `value` 包含多少个元素来区分这两种输入，对于第一种输入，我们可以用来计算标签为 `label` 的 `title` 中单词 `word` 出现了多少次，对于第二种输入，可以用来统计标签为 `label` 的 `title` 数。

```
import sys
"""
输入:
<label, <word, 1>>
<label, 1>
输出:
<<label>, <0, Nei, word, numofword>>
统计标签为label1的title中 单词word的出现次数
"""
wordHash = dict()
# 键值为 (label, word) 的字典，统计标签为label的title中单词word出现的次数
titleHash = dict()
# 键值为label的字典，统计标签为label的title个数
for row in sys.stdin:
    key, value = row.split('\t\t')
    value = value.split()
    if len(value) == 1:
        if key not in titleHash:
            titleHash[key] = 0
        titleHash[key] += int(value[0])
    elif len(value) == 2:
        newkey = key + ' ' + value[0]
        if newkey not in wordHash:
            wordHash[newkey] = 0
        wordHash[newkey] += int(value[1])

for k, v in wordHash.items(): # 得到输出
    k = k.split(' ')
    key = k[0]
    value = str(0) + ' ' + str(titleHash[key]) + ' ' + k[1] + ' ' + str(v)
    print(key + "\t\t" + value)
```

上面 `reducer` 的代码输出键值对为(`label`, (`0`, `Nei`, `word`, `numofword`))，它的意思是，标签为 `label` 的 `title` 一共有 `Nei` 个，在这里面单词 `word` 一共出现了 `numofword` 次。这里 `value` 中还有一个 `0`，这里这个 `0` 是用来表示这个键值对是训练数据的键值对。后面会使用 `1` 来表示需要预测的数据的键值对。

### 3.2.2 准备需要预测的数据----prepare\_predictdata

我们的算法是按照训练预测分离，其中训练的模型可以复用到不同测试数据上的方式设计的。这一步 `prepare_predictdata` 是一个本地程序，的作用是整合训练数据和需要预测的数据到一个文本中作为预测 `MapReduce` 程序的输入，这里预测的数据你可以随意替换成不同的，比较灵活。这一步会得到如下输出：

- A. (`label`, (`0`, `Nei`, `word`, `numofword`))
- B. (`label`, (`1`, `testID`, `testtitle`))

其中前者如 3.2.1 中所述，后者的意义是第 `testID` 条需预测数据的内容是 `testtitle`，`1` 表示它是测试数据，请注意，这里这个 `label` 不是它真实的标签，而是希望程序计算它属于这个 `label` 的概率，也就是说假设我们一个有 `n` 种标签，那么对于每一条测试数据，我们会得到 `n` 个键值对：

(`labeli`, (`1`, `testID`, `testtitle`))     `i`=0,1,2...`n`

这一步不是 `MapReduce` 过程，代码不在此展示，可见附件中的 `MapReduce_code` 文件夹下的 `prepare_predictdata.py`。

### 3.2.3 预测过程 1----predict\_mapper1 和 predict\_reducer1

这一步是求出每条需预测数据属于每种 `label` 的概率。这一步中 `mapper` 的输出=输入，我们的目的是希望利用 `mapper` 之后的 `shuffle` 过程对数据先按照键值作一个排序，将相同 `label` 的训练数据和测试数据归到一起之后再作处理：

```
import sys
"""
输入:
<<label>, <0, N, Nei, word, numofword>>
<<label>, <1, testID, testtitle>>
输出 = 输入，目的是理由shuffle先排个序
"""
for row in sys.stdin:
    row = row.strip()
    key, value = row.split('\t\t')
    key_value = "{}\t\t{}".format(key, value)
    print(key_value)
```





编程的方法来实现，因此我们采用了将这个多标签多分类问题转化成了单一标签多分类问题，将每条 issue 的第一个软件 bug 标签作为它的真实标签。我们将重点放在学习 MapReduce 的编程模式上。

## 4.2 评价指标

前面提到了，每个 issue 可能有多个标签，但是我们只取了一个，而人们在手工打标签的时候不会按照最相关的顺序打标签，所以有可能我们预测出来的 label 中，概率最高的 label 不是我们之前保留的那个 label，因此，我们参照了行人识别领域的评价指标，采用 rank-1 正确率和 rank-5 正确率来评价我们的模型。

rank-1 正确率就是取概率最高的预测 label 来跟我们保留的真实 label 比较从而计算正确率。对于 rank-5 正确率，对每一条待预测数据，假设我们预测出的概率最高的 5 个 label 中存在一个 label 与我们保留的真实 label 相同，我们就认为这个数据预测正确。

## 4.3 实验环境

由于机器条件限制，我们的实验没有使用真实的分布式集群，我们使用 VMware 虚拟机在一台电脑上安装了三个 Ubuntu16.04 系统，利用虚拟机的 NAT 网络设置将三个虚拟机以及我们的本机置于同一个网段下，这样我们可以将这三个系统部署成一个分布式集群，我们称之为虚拟分布式集群，然后在该虚拟集群上面安装 Hadoop 来跑 MapReduce 并行代码。并且由于和本机在一个网段下，我们可以在本机上通过 ssh 工具登陆它们，而避免了虚拟机和本机之间麻烦的键鼠切换。具体设置方法可以查看我们 Environment Setting 文件夹里的设置说明。

## 4.4 实验结果

为了验证我们的 MapReduce 代码是否编写正确，我们首先实现了一个单机版的程序在本地进行实验，以便后续作对比实验。单机版程序保存在 local\_code 文件夹下的 classifier\_local.py 中。实验结果如图 6 所示：

```

C:\Users\JW\Desktop\Naive_Bayesian_Classifier_MapReduce> local_code> .\classifier_local.py
12 NWei, NW, Nei, words, N = dict(), dict(), dict(), set(), 0
13 f1 = open(inputfile, "r", encoding="utf-8")
14 lines = f1.readlines()
15
16 for row in lines:
17     ei, text = row.split('\t\t')
18     text = text.split()
19     N += 1 # 文章计数加1
20     if ei not in Nei:
21         Nei[ei] = 0
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
```

较高，而 label 的总类是达到了 32 种的，能在前 5 个就找到正确的 label，说明我们的分类器的性能是很好的。

#### 4.5 实验中遇到的问题

最开始我们认为在将数据输送到 Map 过程的时候就会将所有相同的键值的所有数据归在一起计算，所以我们一开始在 Map 过程中进行了统计类计算，这导致最后结果是不正确的。原因是 Map 的输入是将输入数据分成一个一个 split，然后对每一个 split 创建一个计算任务来进行分布式计算，这些 split 是相同键值的数据，但是不一定是该键值的所有数据，有可能两个 split 的数据加起来才是该键值的所有数据。因此我们在编写 MapReduce 程序的时候一定不能在 Map 过程中进行像最终统计这样的计算，否则会导致结果错误。只有在 Reduce 过程中，才会对相同键值的所有数据一起处理。

另外，在设计 MapReduce 版的朴素贝叶斯分类程序的时候，由于该算法设计到的需要计算的变量较多，只用一次 MapReduce 完成该任务基本上是不可能的。一开始我们没有想到要拆成几次 MapReduce 的做法，在这一步卡了很久。后面想到了这个做法，再将所需变量一个个写出来，才想出每一步 MapReduce 应该执行什么功能。所以我们以后如果还要编写 MapReduce 程序，不需要执着于一个 MapReduce 完成所有任务，可以拆分过程，分而治之。

最后，当我们代码运行出错的时候，Hadoop 的屏幕输出不会告诉我们在哪一步错了，所以一开始我们每次遇到问题，只能肉眼反反复复地排查代码，但后面发现错误信息可以在 Hadoop 安装目录下的 logs 文件夹中的 userlogs 文件里找到。

## 5 总结

本次大作业我们针对目前 Github 仓库上的 issues 栏目的软件 bug 标签需要人为添加而导致效率低下的问题，设计了基于 MapReduce 的朴素贝叶斯分类算法，并且在虚拟分布式小集群上运行了我们的代码。实验表明，我们的分布式代码的结果与本地单机版代码的结果是一样，说明我们的代码编写正确。此外，分类性能也在我们的

预期之内，rank-5 准确率达到了一个比较好的数值。

通过这次大作业，我们学习了 MapReduce 编程模型，对它的程序编写方法有了比较清晰的了解，同时我们也感受了什么是分布式的计算，学到了原来没有接触过的知识。

本次大作业的遗憾之处在于我们没有找到一个良好的可以改写成 MapReduce 形式的支持多标签多分类的算法，希望以后有机会能够改进。

#### 参考文献

- [1] 《朴素贝叶斯》，百度百科，<https://baike.baidu.com> 2019,11,01
- [2] 《贝叶斯定理》，百度百科，<https://baike.baidu.com> 2019,09,26
- [3] 《hdfs》，百度百科，<https://baike.baidu.com> 2019,11,23
- [4] 《MapReduce 工作流程最详细解释》，简书，<https://www.jianshu.com/p/461f86936972> 2018,10,18
- [5] 《Python 实现 Hadoop MapReduce 程序》，cnblog，<https://www.cnblogs.com/chushiyaoyue/p/5713177.html> 2016,07,28