

基于 MapReduce 的软件 Bug 分类

19335015 陈恩婷

1. 题目要求

题目

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，现在的分类标签大都是基于人工添加的，效率比较低。本项目通过爬取大量具有分类标签的 Bug，利用 MapReduce 分布式编程模型，实现分类算法，自动给 Bug 加上标签。

要求：

- 爬取至少 1000 个具有分类标签的 bug；
- 采用 MapReduce 实现分类算法；
- 测试验证算法的准确度；
- 分析结果并得出结论；
- 提交源码和报告，压缩后命名方式为：学号_姓名_班级

2. 实验环境的搭建

本次实验我安装的是 Hadoop 3.2.2 版本，一开始尝试了在 Windows 10 家庭版环境下配置，但是没有成功，于是换到了 Ubuntu 21.10 下进行安装，主要参考的是[这篇文章](#)，就可以安装成功了。

3. 实验过程与结果

(1) 获取数据并进行预处理

由于本项目要求要爬取至少 1000 个具有分类标签的 bug，所以需要选择一个带标签的 issue 比较多的 repository。这里我选用了 tensorflow 的 repository，并且选了以下比较热门的标签来运行分类任务：

```
'comp:apis','comp:autograph','comp:cloud','comp:core',  
'comp:signal','comp:tensorboard','comp:tf.function','comp:tfdgb',  
'comp:tpus','comp:xla'
```

程序首先将 github 上所需的 issue 的标题爬所在的页面爬取下来，再将所需的标题文字进行过滤，最后将所有的标题按照标签进行分类，最后把原始数据存储在 issue_titles.txt 中供后面的步骤使用。

代码放置在 1_GetAndPreprocess 目录下。

(2) 单词的计数

爬取好所需数据后，接下来就是用 Hadoop 运行经典的 WordCount 程序了。首先用命令将准备好的原始数据 issue_titles.txt 上传到 HDFS 上：

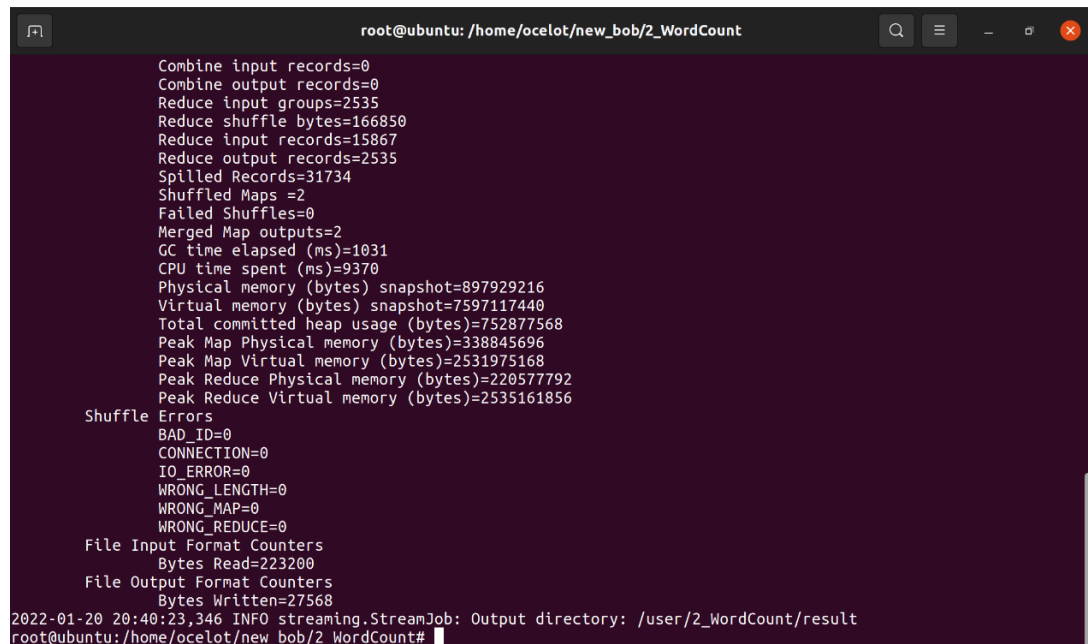
```
hdfs dfs -mkdir -p /user/2_WordCount
```

```
hdfs dfs -put issue_titles.txt /user/2_WordCount
```

再用命令启动 Hadoop 任务：

```
$ hadoop jar /home/ocelot/hadoop-3.2.2/share/hadoop/tools/lib/hadoop-streaming-3.2.2.jar -D stream.non.zero.exit.is.failure=false -input "/user/2_WordCount" -output "/user/2_WordCount/result" -mapper "python3 ./mapper.py" -reducer "python3 ./reducer.py" -file ./*.py
```

运行结果如下：

A terminal window titled 'root@ubuntu: /home/ocelot/new_bob/2_WordCount' displays the output of a Hadoop WordCount job. The output lists various metrics such as 'Combine input records=0', 'Reduce shuffle bytes=166850', and 'Spilled Records=31734'. It also shows memory usage statistics like 'Physical memory (bytes) snapshot=897929216' and 'Virtual memory (bytes) snapshot=7597117440'. At the bottom, it indicates the output directory as '/user/2_WordCount/result' and shows the command prompt 'root@ubuntu: /home/ocelot/new_bob/2_WordCount#'.

```
Combine input records=0
Combine output records=0
Reduce input groups=2535
Reduce shuffle bytes=166850
Reduce input records=15867
Reduce output records=2535
Spilled Records=31734
Shuffled Maps =2
Failed Shuffles=0
Merged Map outputs=2
GC time elapsed (ms)=1031
CPU time spent (ms)=9370
Physical memory (bytes) snapshot=897929216
Virtual memory (bytes) snapshot=7597117440
Total committed heap usage (bytes)=752877568
Peak Map Physical memory (bytes)=338845696
Peak Map Virtual memory (bytes)=2531975168
Peak Reduce Physical memory (bytes)=220577792
Peak Reduce Virtual memory (bytes)=2535161856
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=223200
File Output Format Counters
Bytes Written=27568
2022-01-20 20:40:23,346 INFO streaming.StreamJob: Output directory: /user/2_WordCount/result
root@ubuntu: /home/ocelot/new_bob/2_WordCount#
```

运行时终端的输出保存在了 terminal_output.txt 中。

(3) 生成 TF-IDF 向量，并划分训练集和测试集

TF-IDF 是自然语言处理领域中，实现文本向量化的其中一种方法，主要的计算公式如下：

□ **tf-idf**: tf-idf value for word t in document d :

$$w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$$

Words like "the" or "it" have very low idf

$$\text{tf}_{t,d} = \text{count}(t,d)$$

Instead of using raw count, we squash a bit:

$$\text{tf}_{t,d} = \log_{10}(\text{count}(t,d)+1)$$

□ df_t is the number of documents t occurs in.

(note: not collection frequency: total count across all documents)

$$\text{idf}_t = \log_{10} \left(\frac{N}{\text{df}_t} \right)$$

对于所有的文本标题，计算出该标题中的所有单词的 TF-IDF 值，并拼接成一个向量，就得到了该标题的 TF-IDF 向量。再按 8:2 的比例划分训练集和测试集，就得到了训练和测试的数据。具体代码放置在 3_TF-IDF 目录下。

(4) 训练模型

训练模型的过程借助 Hadoop 运行，主要的任务就是对于每一个标签，计算出上一步中每个 issue 标题的 TF-IDF 向量的平均值。这个平均值就是这类标签下的 issue 的特征向量。在测试阶段，将每一类标签的特征向量和测试样例进行求余弦距离的计算，结果最优的一类即为预测出的标签类别。

代码的实现放置在 4_Train 下。将上一步得到的训练集数据上传到 HDFS 上后，启动 Hadoop 任务：

```
root@ubuntu:/home/ocelot/new_bob/4_Train# hadoop jar
/home/ocelot/hadoop-3.2.2/share/hadoop/tools/lib/hadoop-streaming-3.2.2.jar
-D stream.non.zero.exit.is.failure=false -input "/user/4_Train" -output
"/user/4_Train/result" -mapper "python3 ./mapper.py" -reducer
"python3 ./reducer.py" -file ./*.py
```

```
root@ubuntu: /home/ocelot/new_bob/4_Train
Combine input records=0
Combine output records=0
Reduce input groups=10
Reduce shuffle bytes=43194036
Reduce input records=2358
Reduce output records=10
Spilled Records=4716
Shuffled Maps =2
Failed Shuffles=0
Merged Map outputs=2
GC time elapsed (ms)=1416
CPU time spent (ms)=72240
Physical memory (bytes) snapshot=980774912
Virtual memory (bytes) snapshot=7597371392
Total committed heap usage (bytes)=849346560
Peak Map Physical memory (bytes)=350015488
Peak Map Virtual memory (bytes)=2558902272
Peak Reduce Physical memory (bytes)=314695680
Peak Reduce Virtual memory (bytes)=2566852608
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=43188068
File Output Format Counters
  Bytes Written=279612
2022-01-20 22:50:49,738 INFO streaming.StreamJob: Output directory: /user/4_Train/result
root@ubuntu: /home/ocelot/new_bob/4_Train#
```

运行结果放置在 HDFS 的/user/4_Train/result 中，终端的输出保存在 terminal_output.txt 中。

(5) 测试

训练好模型后就可以开始测试了。测试的原理主要是通过计算测试文本的向量和每个向量之间的正弦值，找出正弦值最大的标签类别，即为模型预测出的类别，再根据真实的分类类别计算出每一类的正确率，具体公式如下：

$$\text{cosine}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| |\vec{w}|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}}$$

Based on the definition of the dot product between two vectors a and b

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \theta \quad \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|} = \cos \theta$$

运行结果如下：

```
comp:apis | Total: 184 | Correct: 159 | Accuracy: 0.8641304347826086
comp:autograph | Total: 52 | Correct: 46 | Accuracy: 0.8846153846153846
comp:cloud | Total: 4 | Correct: 4 | Accuracy: 1.0
comp:core | Total: 145 | Correct: 29 | Accuracy: 0.2
comp:signal | Total: 2 | Correct: 2 | Accuracy: 1.0
comp:tensorboard | Total: 63 | Correct: 57 | Accuracy: 0.9047619047619048
comp:tf.function | Total: 6 | Correct: 0 | Accuracy: 0.0
comp:tfdag | Total: 6 | Correct: 4 | Accuracy: 0.6666666666666666
comp:tpus | Total: 48 | Correct: 34 | Accuracy: 0.7083333333333333
comp:xla | Total: 85 | Correct: 67 | Accuracy: 0.788235294117647
root@ubuntu: /home/ocelot/new_bob/5_Test#
```

可以看到，除了总标签数只有 6 的 comp:tf.function 标签正确率为 0 外，其余的标签的正确率都高于随机算法（10%）的概率，大多数都达到了 70% 以上，其中甚至有两个标签识别率达到了 100%。可以看到模型效果不错。

4. 遇到的困难和解决方法

1. 在 Windows 10 下尝试安装 Hadoop 的过程中，在启动节点时出现了以下报错：

```
Apache Hadoop Distribution - hadoop namenode
at org.apache.hadoop.ipc.Server$Listener$Reader.run(Server.java:1257)
2022-01-19 20:28:34,024 INFO ipc.Server: Socket Reader #1 for port 9820: readAndProcess from client 127.0.0.1:55932 threw exception [java.lang.IllegalArgumentException: Null user]
java.lang.IllegalArgumentException: Null user
    at org.apache.hadoop.security.UserGroupInformation.createRemoteUser(UserGroupInformation.java:1301)
    at org.apache.hadoop.security.UserGroupInformation.createRemoteUser(UserGroupInformation.java:1288)
    at org.apache.hadoop.util.ProtoUtil.getUgi(ProtoUtil.java:141)
    at org.apache.hadoop.util.ProtoUtil.getUgi(ProtoUtil.java:122)
    at org.apache.hadoop.ipc.Server$Connection.processConnectionContext(Server.java:2453)
    at org.apache.hadoop.ipc.Server$Connection.processRpcOutOfBandRequest(Server.java:2759)
    at org.apache.hadoop.ipc.Server$Connection.processOneRpc(Server.java:2569)
    at org.apache.hadoop.ipc.Server$Connection.readAndProcess(Server.java:2318)
    at org.apache.hadoop.ipc.Server$Listener.doRead(Server.java:1431)
    at org.apache.hadoop.ipc.Server$Listener$Reader.doRunLoop(Server.java:1286)
    at org.apache.hadoop.ipc.Server$Listener$Reader.run(Server.java:1257)
2022-01-19 20:28:39,035 INFO ipc.Server: Socket Reader #1 for port 9820: readAndProcess from client 127.0.0.1:55933 threw exception [java.lang.IllegalArgumentException: Null user]
java.lang.IllegalArgumentException: Null user
    at org.apache.hadoop.security.UserGroupInformation.createRemoteUser(UserGroupInformation.java:1301)
    at org.apache.hadoop.security.UserGroupInformation.createRemoteUser(UserGroupInformation.java:1288)
    at org.apache.hadoop.util.ProtoUtil.getUgi(ProtoUtil.java:141)
    at org.apache.hadoop.util.ProtoUtil.getUgi(ProtoUtil.java:122)
    at org.apache.hadoop.ipc.Server$Connection.processConnectionContext(Server.java:2453)
    at org.apache.hadoop.ipc.Server$Connection.processRpcOutOfBandRequest(Server.java:2759)
    at org.apache.hadoop.ipc.Server$Connection.processOneRpc(Server.java:2569)
    at org.apache.hadoop.ipc.Server$Connection.readAndProcess(Server.java:2318)
    at org.apache.hadoop.ipc.Server$Listener.doRead(Server.java:1431)
    at org.apache.hadoop.ipc.Server$Listener$Reader.doRunLoop(Server.java:1286)
    at org.apache.hadoop.ipc.Server$Listener$Reader.run(Server.java:1257)
```

```
Apache Hadoop Distribution - hadoop datanode
12b336747464d6fbbf370c9a20; jvm 11.0.11+9-LTS-194
2022-01-19 20:28:06,948 INFO server.session: DefaultSessionIdManager workerName=node0
2022-01-19 20:28:06,949 INFO server.session: No SessionScavenger set, using defaults
2022-01-19 20:28:06,957 INFO server.session: node0 Scavenging every 60000ms
2022-01-19 20:28:06,986 INFO handler.ContextHandler: Started o.e.j.s.ServletContextHandler@17f460bb{/logs,/logs,file:///D:/hadoop-3.2.2/logs/,AVAILABLE}
2022-01-19 20:28:06,988 INFO handler.ContextHandler: Started o.e.j.s.ServletContextHandler@88a8218{/static,/static,file:///D:/hadoop-3.2.2/share/hadoop/hdfs/webapps/static/,AVAILABLE}
2022-01-19 20:28:07,162 INFO handler.ContextHandler: Started o.e.j.w.WebAppContext@72805168{datanode/,file:///D:/hadoop-3.2.2/share/hadoop/hdfs/webapps/datanode/,AVAILABLE}{file:/D:/hadoop-3.2.2/share/hadoop/hdfs/webapps/datanode}
2022-01-19 20:28:07,180 INFO server.AbstractConnector: Started ServerConnector@4aa3d36{HTTP/1.1,[http/1.1]}{localhost:55836}
2022-01-19 20:28:07,181 INFO server.Server: Started @4806ms
2022-01-19 20:28:08,353 INFO web.DatanodeHttpServer: Listening HTTP traffic on /0.0.0.0:9864
2022-01-19 20:28:08,363 INFO util.JvmPauseMonitor: Starting JVM pause monitor
2022-01-19 20:28:08,363 INFO datanode.DataNode: dnUserName =
2022-01-19 20:28:08,365 INFO datanode.DataNode: supergroup = supergroup
2022-01-19 20:28:08,429 INFO ipc.CallQueueManager: Using callQueue: class java.util.concurrent.LinkedBlockingQueue, queueCapacity: 1000, scheduler: class org.apache.hadoop.ipc.DefaultRpcScheduler, ipcBackoff: false.
2022-01-19 20:28:08,457 INFO ipc.Server: Starting Socket Reader #1 for port 9867
2022-01-19 20:28:08,749 INFO datanode.DataNode: Opened IPC server at /0.0.0.0:9867
2022-01-19 20:28:08,770 INFO datanode.DataNode: Refresh request received for nameservices: null
2022-01-19 20:28:08,782 INFO datanode.DataNode: Starting BPOfferServices for nameservices: <default>
2022-01-19 20:28:08,795 INFO datanode.DataNode: Block pool <registering> (Datanode Uuid unassigned) service to localhost/127.0.0.1:9820 starting to offer service
2022-01-19 20:28:08,805 INFO ipc.Server: IPC Server Responder: starting
2022-01-19 20:28:08,805 INFO ipc.Server: IPC Server listener on 9867: starting
2022-01-19 20:28:08,951 WARN datanode.DataNode: Problem connecting to server: localhost/127.0.0.1:9820
2022-01-19 20:28:13,962 WARN datanode.DataNode: Problem connecting to server: localhost/127.0.0.1:9820
```

解决方法：在和助教和老师讨论了各种解决方法以后，发现都在 Windows 下不可行，最终只能在 Ubuntu 下重新尝试安装才能成功。主要参考的是[这篇文章](#)，当然过程也是非常复杂，不过在 Linux 系统下一般按部就班地安装都不会出大问题，这也是一个教训。

2. 由于 Ubuntu 20.10 已过期需要重装系统，而 Ubuntu 系统下载安装缓慢

解决方法：可以直接使用中大的 VPN 进入校内的镜像网站下载 Ubuntu 的安装盘，在第一次启动安装盘时选择最小的安装选项即可，从而节省时间。

3. 安装 Hadoop 后，在 root 用户下第一次启动节点不成功，出现了 HDFS_NAMENODE_USER, HDFS_DATANODE_USER & HDFS_SECONDARYNAMENODE_USER not defined 错误

解决方法：参考网络上的[这篇问答](#)，在 \$HADOOP_HOME/etc/hadoop/hadoop-env.sh 文件中加上以下命令来添加环境变量：

```
export HDFS_NAMENODE_USER=root  
  
export HDFS_DATANODE_USER=root  
  
export HDFS_SECONDARYNAMENODE_USER=root  
  
export YARN_RESOURCEMANAGER_USER=root  
  
export YARN_NODEMANAGER_USER=root
```

再次运行命令，就可以启动成功了。

4. 执行 `hdfs dfs -mkdir` 命令时，出现了 No such file or directory 的错误

解决方法：查阅了网上的资料，再用 `-ls` 命令检查文件系统，发现出现这个问题的原因是需要创建的目录的上级目录还没有被创建。在 `-mkdir` 后加入 `-p`，再次运行命令，就可以解决此问题。

5. 不清楚如何获取位于 HDFS 上的 Hadoop 任务的输出文件

解决方法：很简单，用 `hdfs dfs -get [file]` 即可，记得加上文件路径。

5. 实验总结与感想

本次实验是分布式系统的期末项目，在完成实验的过程中我系统地复习了 Hadoop Mapreduce 的使用并进行了实际练习，掌握了 MapReduce 类型的程序的设计思想与方法，体会到了前人的智慧与努力。同时我也锻炼了排查和解决问题，并且分析实验结果的能力，学会了如何与他人沟通，向他人请教等等。

在 Hadoop 的安装和程序的编写与运行的过程中，我还体会到了细节的重要性。尽管通常我们都是因为一些小的地方出现的问题而导致实验难以推进，但究其原因很多也是在大的方向上没有设计好，例如解决问题的整个思路和每一步具体需要完成什么目标等等。一旦大体的结构设计清晰，中途进行大改的可能性就会降低，细节处的问题很多也就迎刃而解了。

非常感谢老师和助教不厌其烦地回答我在实验中遇到的问题，希望自己在以后能够再接再厉，在分布式系统和其他领域学到更多知识，解决更多问题。

6. 参考链接

1. <https://phoenixnap.com/kb/install-hadoop-ubuntu>
2. <https://stackoverflow.com/questions/48129029/hdfs-namenode-user-hdfs-datanode-user-hdfs-secondarynamenode-user-not-defined>
3. <https://stackoverflow.com/questions/40143528/hdfs-dfs-mkdir-no-such-file-or-directory>
4. <https://community.cloudera.com/t5/Support-Questions/Copy-the-contents-of-quot-output-part-00000-quot-in-another/td-p/44685>
5. <https://stackoverflow.com/questions/54502279/org-apache-hadoop-fs-parentnotdirectoryexception-tmp-is-not-a-directory>