

基于hadoop的KNN实现

13349009 陈榕涛

Part 1 设计概述

这次实践中，我打算在hadoop系统上实现KNN（K Nearest Neighbors）算法。KNN算法比较简单，一个test sample的label有K个与它最相似（距离最近）的train sample的类别投票决定，各个类别的权重为1:1。

我实现的版本是——把train data分配到多个mapper上去计算，每个mapper计算其train sample与所有test sample的距离；然后在Combiner处做一遍本地的聚集，对于每一个test sample，从中选择K个最小的距离，作为reduce的输入；最后reduce的时候，再次从所有的距离中，选择K个最小的距离，投票决定类别。

Part 2 实现过程

1. 数据

在实现的时候，考虑到一般是train data比较庞大，所以我是将train data作为map的输入，输入格式按照默认的TextInputFormat，即读取每行作为一个mapper的输入。

而将所有的test data全部读取放在内存中，为了在内存里只保存一个拷贝，我是将其作为Mapper子类的一个static变量。另，其实test data也可以做成分布式的，但是鉴于自己初学，还未能对整个系统理解把握，所以就简单实现了。

2. 各级的功能和key-value说明

注：Elem是自定义的一个类型，它包含三个属性：

1) 两个sample之间的距离；2) 主sample的label；3) 从sample的label。

Mapper的输入：按照默认的的TextInputFormat读取的话，k-v为(LongWritable, Text)。

Mapper的输出：k-v为(IntWritable, Elem)，key是指test sample的下标，唯一指代这个test sample的键；value就是Elem，里面包含一个train sample对于这个test sample的距离和两者的label。

本地Combiner的输入：同Mapper的输出（实质上二者就是相同的）。

本地Combiner的输出：k-v为(IntWritable, Elem)，解释同Mapper的输出，这里做的事情，是将本地对于同个test sample的所有那些train sample中，先选出K个距离最小的输出，这样可以较少Reducer的负担和网络通信的负担。

Reducer的输入：同本地Combiner的输出。

Reducer的输出：k-v为(Text, Text)，第一个Text是实际的label，第二个Text是predict的label。

3. 如何选择出K个最小的

其实这个问题很简单，可以直接对所有数据进行排序，截取前K个数据就好了，这样的空间复杂度都是 $O(n)$ ，时间复杂度可以为 $O(n \log n)$ ，看起来好像挺友好的。但是，问题来了，如果train sample很多，多到单机的内存放不下，该怎么办？

应该有多种解决方法，我选择的是比较简单的方法——插入排序。即用一个size为k的数组，每次将数据插入到这个数组里，溢出的数据可以丢弃，这样的时间复杂度还是 $O(Nk)$ ，空间复杂度为 $O(k)$ ，比前者好多了多，而且一般k最大就是几十几百（KNN算法本身的性质决定的，因为选择太多的sample来投票的话，其中就夹杂了太多noise了），所以这样子实现起来，整个算法就是scalable的！

我将其在java中封装为一个类：

```
public static class TopKElem {
    private Elem[] topK;
    private int index, size;

    public TopKElem(int k) {
        index = -1;
        size = k;
        topK = new Elem[size+1];
    }

    public void insert(Elem x) {
        index = Math.min(index+1, size);
        int pos = index - 1;

        while (pos >= 0 && x.getDist().compareTo(topK[pos].getDist()) < 0) {
            topK[pos+1] = topK[pos];
            --pos;
        }
        if (pos < size)
            topK[pos+1] = x;
    }

    public Elem[] getTopK() {
        if (index < 0)
            return null;
        int num = Math.min(index+1, size);
        Elem[] data = new Elem[num];
        for (int i = 0; i < num; ++i)
            data[i] = topK[i];
        return data;
    }
}
```

图2.1 使用插入排序实现的Top-k-elements

4. 数据集的选择

我是在自己的服务器集群上面实现的，配置如下图所示：



Droplets			
<input type="text" value="Search By Droplet Name"/>			
Name	IP Address	Created	
 Slave1 1 GB / 30 GB Disk / NYC2	162.243.104.27	Yesterday	More ▾
 Master 1 GB / 30 GB Disk / NYC2	107.170.29.17	2 days ago	More ▾

图2.2 Droplets

然后在实践中发现稍微大一些的数据根本跑不动，所以我就选择了较小的并且著名的Iris鸢尾花数据集¹来作为程序的输入。

Part 3 编译运行程序

所有的步骤可以写成一个如下的脚本：

```
rm *.class *.jar

javac -cp ../share/hadoop/common/hadoop-common-2.6.4.jar:../share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.6.4.jar:../share/hadoop/common/lib/commons-cli-1.2.jar KNN.java -d ./

jar -cvf KNN.jar KNN*.class

hdfs dfs -rm -r KNN_Output

hadoop jar KNN.jar KNN KNN_input/train.data KNN_input/test.data KNN_Output

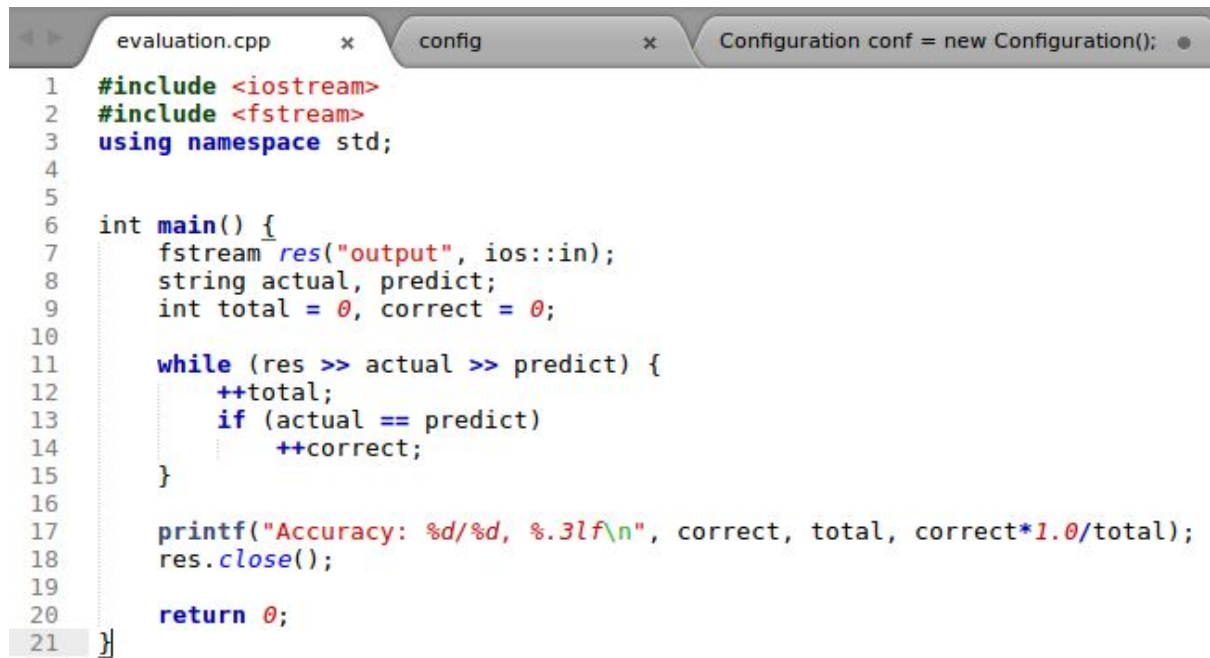
hdfs dfs -cat KNN_Output/* > output.txt

./out
```

图3.1 编译运行的sh脚本

注，最后一条指令./out，是指令我自己写的一个评估结果的脚本，它的内容很短，如图3.2所示：

¹ Iris鸢尾花数据集：<https://archive.ics.uci.edu/ml/datasets/Iris>



```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5
6  int main() {
7      fstream res("output", ios::in);
8      string actual, predict;
9      int total = 0, correct = 0;
10
11     while (res >> actual >> predict) {
12         ++total;
13         if (actual == predict)
14             ++correct;
15     }
16
17     printf("Accuracy: %d/%d, %.3lf\n", correct, total, correct*1.0/total);
18     res.close();
19
20     return 0;
21 }
```

图3.2 评估结果的C++代码

运行的结果如图3.3、3.4、3.5所示：


```
hadoop@Master: /usr/local/hadoop/KNN_code
hadoop@Master: /usr/local/hadoop/KNN_code$ rm *.class *.jar
hadoop@Master: /usr/local/hadoop/KNN_code$
hadoop@Master: /usr/local/hadoop/KNN_code$ javac -cp ../share/hadoop/common/hadoop-common-2.6.4.jar:../share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.6.4.jar:../share/hadoop/common/lib/commons-cli-1.2.jar KNN.java -d ./
Note: KNN.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
hadoop@Master: /usr/local/hadoop/KNN_code$
hadoop@Master: /usr/local/hadoop/KNN_code$ jar -cvf KNN.jar KNN*.class
added manifest
adding: KNN.class(in = 4011)(out= 2001)(deflated 50%)
adding: KNN$Elem.class(in = 1830)(out= 934)(deflated 48%)
adding: KNN$KNNCombiner.class(in = 2252)(out= 1048)(deflated 53%)
adding: KNN$KNNMapper.class(in = 4681)(out= 2117)(deflated 54%)
adding: KNN$KNNReducer.class(in = 3187)(out= 1580)(deflated 50%)
adding: KNN$TopKElem.class(in = 937)(out= 631)(deflated 32%)
hadoop@Master: /usr/local/hadoop/KNN_code$
hadoop@Master: /usr/local/hadoop/KNN_code$ hdfs dfs -rm -r KNN_Output
16/06/24 12:06:54 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Empty interval = 0 minutes.
Deleted KNN_Output
hadoop@Master: /usr/local/hadoop/KNN_code$
hadoop@Master: /usr/local/hadoop/KNN_code$ hadoop jar KNN.jar KNN KNN_input/train.data KNN_input/test.data KNN_Output
16/06/24 12:07:00 INFO client.RMProxy: Connecting to ResourceManager at Master/107.170.29.17:8032
16/06/24 12:07:03 INFO input.FileInputFormat: Total input paths to process : 1
16/06/24 12:07:03 INFO mapreduce.JobSubmitter: number of splits:1
16/06/24 12:07:03 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1466696409545_0030
16/06/24 12:07:04 INFO impl.YarnClientImpl: Submitted application application_1466696409545_0030
16/06/24 12:07:04 INFO mapreduce.Job: The url to track the job: http://Master:8088/proxy/application_1466696409545_0030/
16/06/24 12:07:04 INFO mapreduce.Job: Running job: job_1466696409545_0030
16/06/24 12:07:17 INFO mapreduce.Job: Job job_1466696409545_0030 running in uber mode : false
16/06/24 12:07:17 INFO mapreduce.Job: map 0% reduce 0%
16/06/24 12:07:27 INFO mapreduce.Job: map 100% reduce 0%
16/06/24 12:07:37 INFO mapreduce.Job: map 100% reduce 100%
16/06/24 12:07:37 INFO mapreduce.Job: Job job_1466696409545_0030 completed successfully
```

图3.3 运行结果1

```
hadoop@Master: /usr/local/hadoop/KNN_code$ hadoop jar KNN.jar KNN KNN_input/train.data KNN_input/test.data KNN_Output
16/06/24 12:07:00 INFO client.RMProxy: connecting to ResourceManager at Master/107.170.29.17:8032
16/06/24 12:07:03 INFO input.FileInputFormat: Total input paths to process : 15242
16/06/24 12:07:03 INFO mapreduce.JobSubmitter: number of splits:1
16/06/24 12:07:03 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1466696409545_0030
16/06/24 12:07:04 INFO impl.YarnClientImpl: Submitted application application_1466696409545_0030
16/06/24 12:07:04 INFO mapreduce.Job: The url to track the job: http://Master:8088/proxy/application_1466696409545_0030/
16/06/24 12:07:04 INFO mapreduce.Job: Running job: job_1466696409545_0030
16/06/24 12:07:17 INFO mapreduce.Job: Job job_1466696409545_0030 running in uber mode : false
16/06/24 12:07:17 INFO mapreduce.Job: map 0% reduce 0%
16/06/24 12:07:27 INFO mapreduce.Job: map 100% reduce 0%
16/06/24 12:07:37 INFO mapreduce.Job: map 100% reduce 100%
16/06/24 12:07:37 INFO mapreduce.Job: Job job_1466696409545_0030 completed successfully
16/06/24 12:07:38 INFO mapreduce.Job: Counters: 49

File System Counters
  FILE: Number of bytes read=23352
  FILE: Number of bytes written=261565
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=9218
  HDFS: Number of bytes written=4197
  HDFS: Number of read operations=7
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=2

Job Counters
  Launched map tasks=1
  Launched reduce tasks=1
  Rack-local map tasks=1
  Total time spent by all maps in occupied slots (ms)=7239
  Total time spent by all reduces in occupied slots (ms)=7511
  Total time spent by all map tasks (ms)=7239
  Total time spent by all reduce tasks (ms)=7511
  Total vcore-milliseconds taken by all map tasks=7239
  Total vcore-milliseconds taken by all reduce tasks=7511
  Total megabyte-milliseconds taken by all map tasks=7412736
  Total megabyte-milliseconds taken by all reduce tasks=7691264

Map-Reduce Framework
  Map input records=151
  Map output records=22500
  Map output bytes=1139225
  Map output materialized bytes=23352
  Input split bytes=116
```

图3.4 运行结果2

```
hadoop@Master: /usr/local/hadoop/KNN_code$ hdfs dfs -cat KNN_Output/* > output.txt
hadoop@Master: /usr/local/hadoop/KNN_code$
hadoop@Master: /usr/local/hadoop/KNN_code$ ./out
Accuracy: 60/150, 0.400
```

图3.5 运行结果3 (k=3, 准确率略低, 可以调参优化)

Part 4 遇到的困难和解决方法

1. 分布式环境配置集群各种“摸黑”：一开始是Master和Slave之间的通信出问题了，把Slave的防火墙关掉就好了（因为端口被deny），后来运行程序的时候，发现了新的问题，一直找不出，偶然一次看到一个帖子，说是硬盘存储不够，我升级了服务器的配置之后果然就好了（我使用的是digital ocean提供的VPS服务）。
2. 很难调试：基本只能靠userlogs中的stdout和stderr来查看程序里自己写打印语句输出的结果。然后一开始不知道log在哪里看，一个一个找之后，发现运行的log和输出结果是保存在运行的节点上而不是在Master上的，也就是slave上的，如下图所示：

```
hadoop@Slave1: /usr/local/hadoop/logs/userlogs/application_1466696409545_0029/container_1466696409545_0029_01_000002$ ls
stderr  stdout  syslog
```

图4.1 打印结果所在地

Part 5 总结

这次从配置到写程序到运行，整个流程下来，体验了一把分布式系统是怎样的。然后，虽然单机版的KNN很容易写出来，不过分布式环境下的程序并不是那么简单编写的，有了hadoop之后，工作就比较轻松了，只需要专注于几个层次的设计即可（即输入、Mapper、Combiner、Reducer等）！

虽然挺用心在做，不过最后还是迟交了半个小时（T_T因为是从倒数第二天才开始做的，中间也帮不少同学解决配置环境的问题）。
——2016.06.25 00:30:00

Part 6 参考

1. 存储不够的问题：

<http://stackoverflow.com/questions/30231508/yarn-mapreduce-job-dies-with-strange-message>

2. 讲解hadoop框架运行过程的博客：<http://www.cnblogs.com/hehaiyang/p/4484442.html>

3. yarn内存配置指南：<http://blog.csdn.net/yangfei001/article/details/37766747>

4. 清华学堂在线的大数据课程：

http://www.xuetangx.com/courses/course-v1:TsinghuaX+64100033X+2015_T1_/info