

一个Linux上用python直接处理csv文件的方法

汪兴元

2015 年7 月21 日

摘要

我们常常需要处理体积很大的csv数据或日志文件。一般来讲，导入RDBMS来处理是常用的一种办法，但此办法也不完美，尤其是SQL这种描述型的语言。在表达算法的时候不如一般的程序设计语言的表达力强，某些应用场合虽然能够实现，但是难度太高，优化不易；批量处理大数据，通过在RDBMS上运行SQL效率并不高。

另一种办法就是使用Hadoop或Spark，或导入NoSQL中，再使用MapReduce。但是使用类似Hadoop之类的工具又可能太重，数据量不够运作一个Hadoop集群，还得承受其代价。

本文介绍了直接处理csv文件的一套办法，作为另一个数据处理的选项，示例代码用python，运行的操作系统为Linux，源程序及csv数据文件的编码采用utf8。在从原始数据到最终的目标结果，需要组合本文提到的各种算法，才能达到目的。通过管道-过滤器连接每一个算法，能够将整个处理算法分而治之，同时得到比较好的性能。

本文假设要处理的整个数据集无法一次装入机器的内存。

目录

| | |
|--------------------|---|
| 1 校验数据 | 2 |
| 2 选择列 | 3 |
| 3 排序 | 4 |
| 4 转换ID字段 | 4 |
| 5 去重复值, distinct() | 6 |
| 6 连接 | 6 |
| 6.1 内连接 | 6 |
| 6.2 左连接 | 6 |
| 6.3 全外连接 | 6 |
| 7 过滤数据 | 6 |
| 7.1 谓词及表示方法 | 6 |
| 7.2 谓词的连接关系 | 6 |
| 7.2.1 AND | 6 |
| 7.2.2 OR | 6 |
| 7.2.3 优先级 | 6 |

| | |
|-------------------------|---|
| 8 聚集 | 6 |
| 8.1 max(), min(), avg() | 6 |
| 8.2 group by, having | 6 |
| 8.3 having | 6 |
| 9 补缺失值 | 6 |
| 10 转置 | 6 |

1 校验数据

csv文件可能存在错误。在正常情况下，csv文件不应存在错误。但是写入csv的过程并非事务型的，不象RDBMS那样有很好的ACID保证，故磁盘耗尽，进程异常退出、挂起等因素，直接导致csv文件的格式并非预期。

检查数据没有统一的方法，要视数据自身的特点来做检查。一般是检查一些数据正确所需的必要条件，但必要并不一定充分。一般的检查方法有：

1. 检查列数。如果列数不等于预期值，可以确定此行数据错误。
2. 检查每列数据的格式，比如，数字，日期，或其它满足指定格式的文本串。可以尝试将其解析为对应的类型，看能否成功；或用正则表达式验证。
3. 校验字段的值。前两项都对的情况下，可以检查数据值的取值范围。比如健在的人的年龄不能是负数，也不能是数百以上；历史数据中的记录时间不可能超过当前的日历时钟。

```
1 #! /usr/bin/python
2 # coding=utf-8
3 # validate_history_ai.py
4
5 import csv
6 import sys
7
8 if __name__ == '__main__':
9     infile = sys.stdin
10    outfile = sys.stdout
11
12    reader = csv.reader(infile, delimiter=',', quotechar='"',
13        quoting=csv.QUOTE_ALL, skipinitialspace=True)
14    writer = csv.writer(outfile, delimiter=',', quotechar='"',
15        quoting=csv.QUOTE_ALL, skipinitialspace=True)
16
17    for row in reader:
18        try:
19            # check if the number of columns is right.
20            if len(row) != 12
21                continue;
22            # check the format of each column...skipped.
23            # check the value range of each column...skipped.
24            writer.writerow(row)
25        except:
```

```
26         pass
27     finally:
28         pass
```

如果数据是已经通过有严格校验的系统中生成或导出的，原始数据本身无误，一般做以上三项检查可以查出我们能见到的全部错误。但是如果数据是人工填写或是有故障的机器生成的，这三项检查仍不能确保检查出全部错误。

对于错误的数据该如何处理，要视情况而定。有的业务场景，例如Web服务器的access log，价值不高；又如水温传感器输入的每分钟一次的历史数据，可以丢弃，之后使用插补法补缺；有的场景是不可以这样做的，如交易记录，需要重新导出此段数据或人工处理。样例代码中我们采用了丢弃的方法处理。

以上代码没有直接打开csv文件，而是从标准输入中读取文件，处理完毕之后再写到标准输出。这样的好处是便于通过管道过滤器连接多个处理进程，避免过高的耦合度。本文所有的处理程序都使用管道过滤器连接，不再赘述。

程序中对校验2和3未实现，可以练习下。对于机器导出的数据，做完1可以去掉大部分错误。如果不打算实现2和3。

可以将列数从命令行上读取，成为更通用的子程序。这个作为练习。

2 选择列

原始数据中有可能只有部分列是所要关心的，其它的与要解决的问题无关，可以丢弃。因此要选择列，类似SQL语句中的SELECT所要办的事情。样例程序如下所示，这里我们只对第1，3，4，5列感兴趣。

```
1  #!/usr/bin/python
2  # coding=utf-8
3  # select_history_ai.py
4
5  import csv
6  import sys
7
8  if __name__ == '__main__':
9      infile = sys.stdin
10     outfile = sys.stdout
11
12     reader = csv.reader(infile, delimiter=',', quotechar='"',
13         quoting=csv.QUOTE_ALL, skipinitialspace=True)
14     writer = csv.writer(outfile, delimiter=',', quotechar='"',
15         quoting=csv.QUOTE_ALL, skipinitialspace=True)
16
17     for row in reader:
18         outrow = []
19         outrow.append(row[0])
20         outrow.append(row[2])
21         outrow.append(row[3])
22         outrow.append(row[4])
23         writer.writerow(outrow)
```

可以将列下标从命令行输入，这样这个程序就成为一个通用的选择程序，而不是仅用于示例的场景。这个作为练习。

3 排序

我们对数据做去重、转置、分组、聚集，都需要先对数据排序。因此排序是非常重要的功能。不可或缺。

大数据的排序由于无法直接一次装入内存，故必须使用外部排序。如果能将数据切成能用内部排序的小块，排好序之后，再合并，那么我们就可以完成对大数据的排序。

我们打算从头实现一个大数据排序工具。利用Linux的工具`sort`来做排序。先将csv文件拆分。可以在导出或生成csv的时候，每达到某一固定尺寸就rotate一下，当前文件改名，再创建一个新文件当作当前文件。

假设文件已经切成能用内存装下的多个小文件。注意，使用切割工具切文件，必须从整行数据的位置断开，否则，切口处的那条数据被切坏了。

对每一个文件，执行`sort`。下面的两段代码是在Linux的终端上输入的命令，“\$”表示命令提示符。示例中有两个文件，这里只写了第一个文件的排序，另一个省略。

```
1 $ cat data/att-rec-utf8_part-1.txt \  
2 | python python-src/select_att_rec.py \  
3 | sort --field-separator=, --ignore-leading-blanks --stable \  
4 --key=1.2n,2 --key=2,3 >part-1.csv
```

此命令先用`select_att_rec.py`选择所要的列，再对选择的结果排序，之后将结果重定向到文件`part-1.csv`。所有小文件都排序好之后，再对排序后的文件做merge：

```
1 $ sort --field-separator=, --ignore-leading-blanks --stable \  
2 --key=1.2n,2 --key=2,3 -m part-*.csv > merge-sorted.csv
```

注意在做merge之前，必须先对每个文件排好序；merge时，使用的关键字、分隔符等参数必须与`sort`的时候所用的参数完全一致。

merge完之后，得到一个已经排好序的大文件。

4 转换ID字段

一般在集成两个系统的数据的时候，需要将两个系统的数据合并起来。但是两个系统的数据ID一般是相互独立分配的，因此，如果ID都是同一种类型，比如整数，几乎100% 会发生冲突，即不相同的两个对象，其ID相同。即使用字符串之类的做ID，也难以不发生冲突。因此必须要建立两个系统间的对象的对应关系。

例如，商品销售网站和库管理系统如果是两套独立的系统，分别由不同的团队开发，比如库存管理系统是外购的，商品销售网站是自建的，如果要将两套系统的数据整起来合做分析，又没有办法修改系统达到统一，就需要做ID转换。

下面的程序从标准输入读入历史数据，从命令行读取基础数据文件名，加载基础数据到字典中，利用基础数据中的 $R(ID_{src}, ID_{dest})$ 关系，做了一个映射。逐行将原ID映射到目标ID的方法替换掉ID，实现ID转换的目的。

```
1 #! /usr/bin/python  
2 # coding=utf-8  
3 # transform_data_id.py  
4
```

```

5 import csv
6 import sys
7 import traceback
8
9 dataObjectIds = {}
10
11 if __name__ == '__main__':
12     infile = sys.stdin
13     outfile = sys.stdout
14     cfgfile = None
15     if(len(sys.argv) > 1) :
16         cfgfile = open(sys.argv[1], 'rb')
17
18     cfgreader = csv.reader(cfgfile, delimiter=',', quotechar='"',
19         quoting=csv.QUOTE_ALL, skipinitialspace=True)
20     reader = csv.reader(infile, delimiter=',', quotechar='"',
21         quoting=csv.QUOTE_ALL, skipinitialspace=True)
22     writer = csv.writer(outfile, delimiter=',', quotechar='"',
23         quoting=csv.QUOTE_ALL, skipinitialspace=True)
24
25     for row in cfgreader:
26         try:
27             key = int(row[0]), int(row[1])
28             value = int(row[2])
29             dataObjectIds[key] = value
30         except ValueError:
31             pass
32         finally:
33             pass
34
35     for row in reader:
36         try:
37             key = int(row[0]), int(row[2])
38             dataId = dataObjectIds.get(key, None)
39             converted = []
40             if dataId != None:
41                 converted = [dataId, row[1]]
42                 converted.extend(row[3:])
43                 writer.writerow(converted)
44         except:
45             print >> sys.stderr, row, traceback.format_exc()

```

一般来讲，大多数系统中的某一基础数据对象的ID是可以一次全部装入内存的。比如超市的商品ID，虽然可能有数十万条，但是对于现代计算机的内存来讲并不算多。

对于历史记录，其身的ID，或Primary Key的数量大，其总体积有可能单台机器的内存装不下。不过一般情况下，需要做转换的必要性往往不大，只对其通过Foreign Key引用的基础数据对象的ID有转换的价值。如果实在需要转换，使用6.1提到的内连接方法来做大表的连接，再选择所

要的字段，包括新ID，不选原ID字段即可。

5 去重复值, distinct()

6 连接

6.1 内连接

6.2 左连接

6.3 全外连接

7 过滤数据

7.1 谓词及表示方法

7.2 谓词的连接关系

7.2.1 AND

7.2.2 OR

7.2.3 优先级

8 聚集

8.1 max(), min(), avg()

8.2 group by, having

8.3 having

9 补缺失值

10 转置