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。

本文界绍了直接处理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。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

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。可以偿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
```

```
if __name__ == '__main__':
  infile = sys.stdin
  outfile = sys.stdout

  reader = csv.reader(infile, delimiter=',', quotechar='"',
    quoting=csv.QUOTE_ALL, skipinitialspace=True)
  writer = csv.writer(outfile, delimiter=',', quotechar='"',
    quoting=csv.QUOTE_ALL, skipinitialspace=True)

  for row in reader:
    try:
      # check if the number of columns is right.
      if len(row) != 12
        continue;
      # check the format of each column...skipped.
      # check the value range of each column...skipped.
      writer.writerow(row)
    except:
      pass
    finally:
      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。

```
#! /usr/bin/python
# coding=utf-8
# select_history_ai.py

import csv
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。

我们不打算从头实现一个大数据排序工具。利用Linux的工具