数据分析训练题实验报告

## 问题描述

(实体识别)

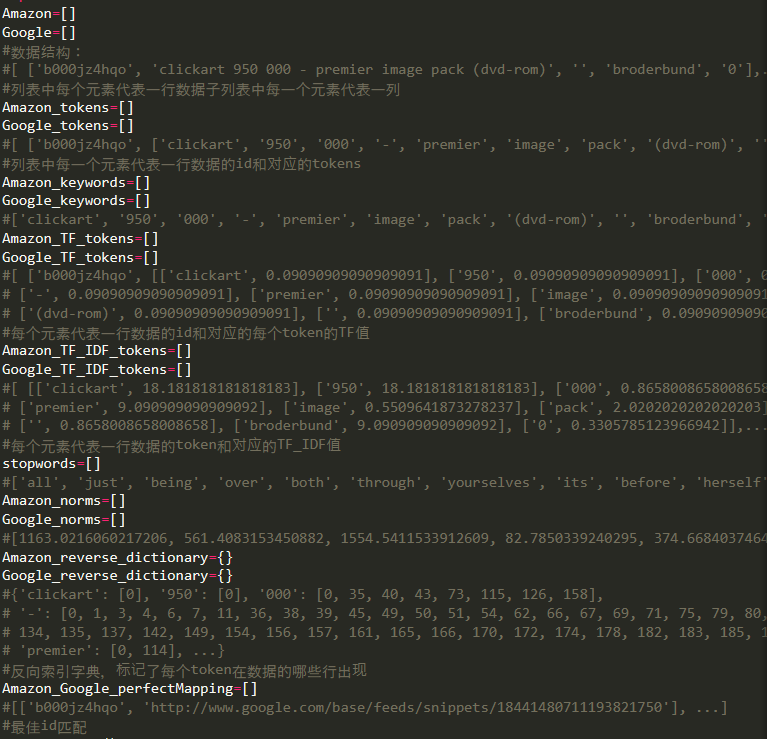
现有一个Amazon的众多商品的数据记录文件(Amazon.csv)，同时有Google对众多商品的数据库记录文件(Google.csv)。如何将两者的数据匹配起来？

原问题(以及问题解)：

http://nbviewer.ipython.org/github/biddata/datascience/blob/master/F14/hw1/hw1.ipynb

## 问题解决

1. 程序定义的列表及其数据结构



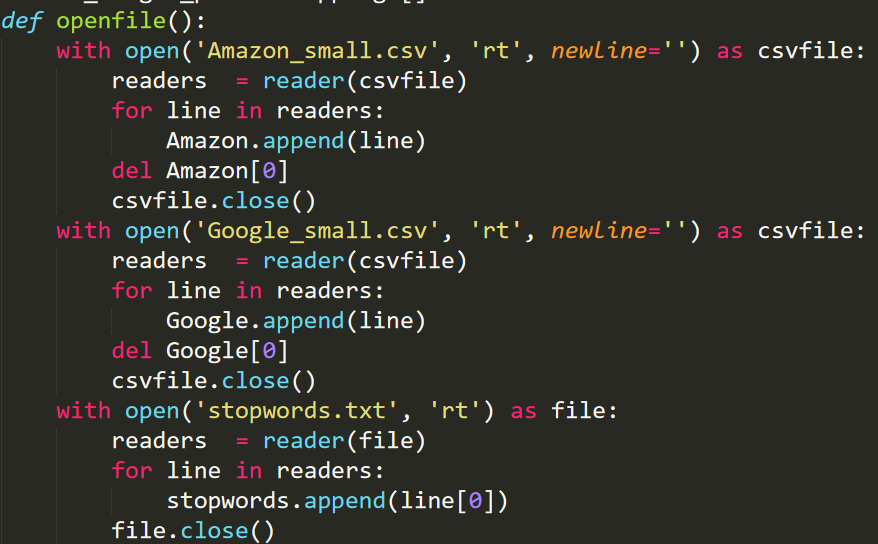
Amazon用于原始数据的存储，Amazon\_tokens存储数据的tokens，Amazon\_keywords存储的是每一行（单行去重）的token，Amazon\_TF\_tokens用来存储TF值，Amazon存储TF\_IDF值，stopwords存储常见词，Amazon\_norms存储tokens的模，Amazon\_reverse\_dictionary存储逆向索引，Amazon\_Google\_perfectMapping存储匹配结果。

二．分割文本，得到有效Tokens

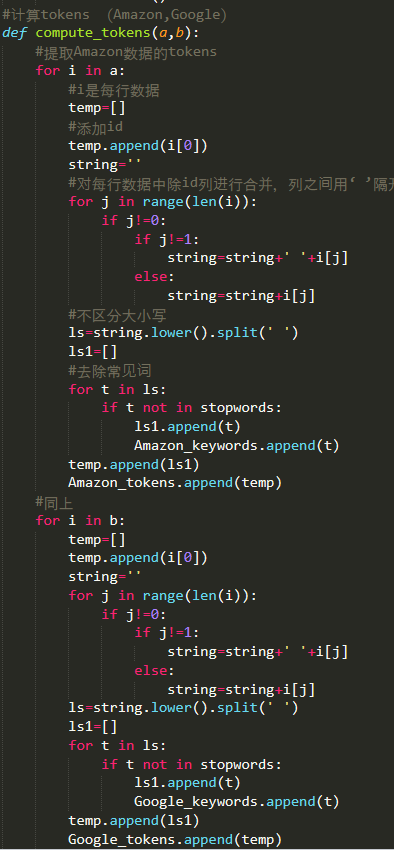
**"token" 其实就是指的一个不区分大小写单词，或者说是以空格等等的符号分割的一个个字符串。"tokens"则是token的列表。一个token允许在tokens中存在多次（在tokens中存在多次意味着该单词在原文中出现了多次）。**在读入csv文件之后，我们应分别为Amazon和Google的每条数据求它的tokens。

**同时像"is"、"of"这样的token，我们对今后的token分析没有贡献。**读入stopwords.txt以删除这些token。

**首先读入csv数据文件和stopwords文件：**



然后编写compute\_tokens函数求得每条数据的tokens：



在求tokens时，我的方法是先将每行数据id列提取出来，然后将id列后面的描述作为tokens，先将他们拼接为长字符串，然后根据空格分词，并存储在一个列表里。

二.求TF-IDF

求TF：对于**一个**tokens中的一个token，

**TF(token)=该token在tokens中的出现次数/tokens中的总token数**

这里用compute\_TF\_tokens函数计算TF值：



函数输入为Amazon\_tokens,Google\_tokens两个列表，在统计tokens时，同一行数据若一个token出现多次，我们就存储了多次，在这里需要计算每个tokens中每个token出现的次数，然后多个相同的token只存储一次TF值。

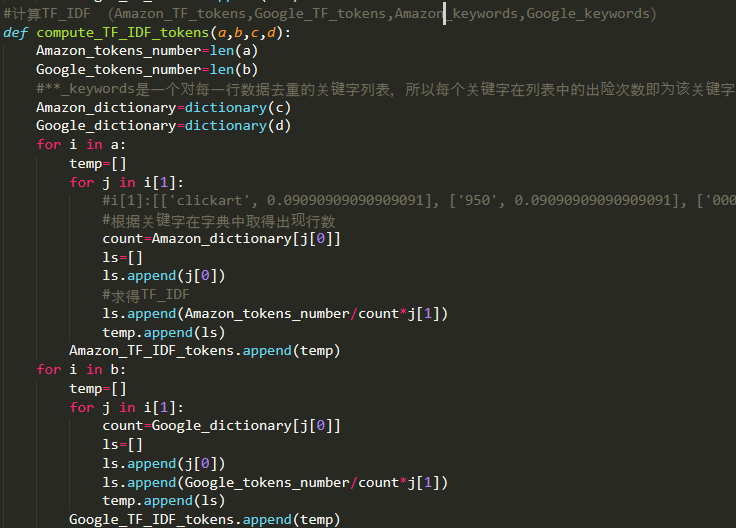
求IDF：对于**所有**tokens中的一个token，

**IDF(token)=tokens(token*列表*)*的个数*/出现过该token的tokens(token*列表*)*的个数***

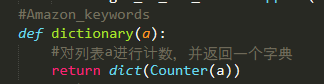
为了减少函数的数量，这里没有单独求IDF，而是将它和TF\_IDF放在一起求。

求TF-IDF:TF-IDF(token)=TF(token)\*IDF(token)

利用compute\_TF\_IDF\_tokens函数求TF\_IDF：



首先计算出数据总条数，在这里为了优化计算速度，采用了一个字典对每个token出现了几行先存储在一个字典里：



由于Amazon\_keywords列表对每行中的数据去重存储，但不同列中相同数据则存储多次，所以对该列表计数即为每个token出现的行数。

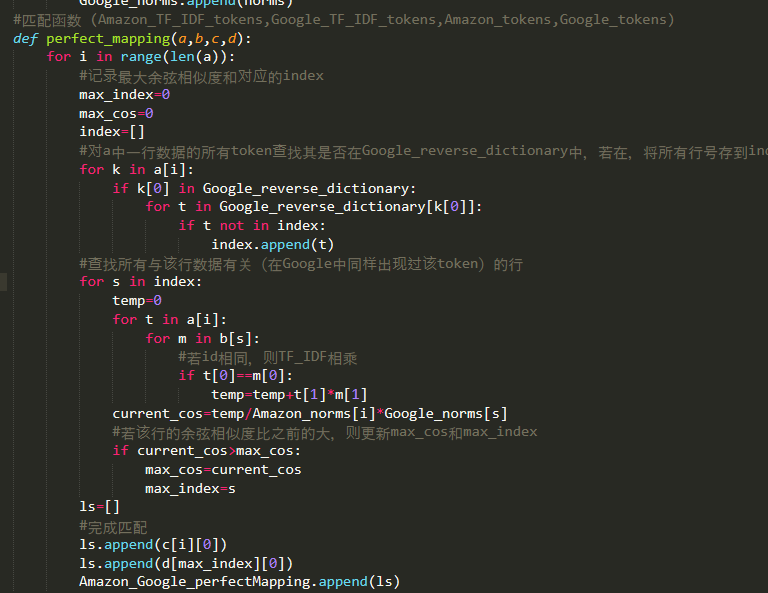
三.求最佳匹配

在求最佳匹配之前，为了优化，我们提前对Amazon中的token在Google中的位置存储了一个逆向索引字典：



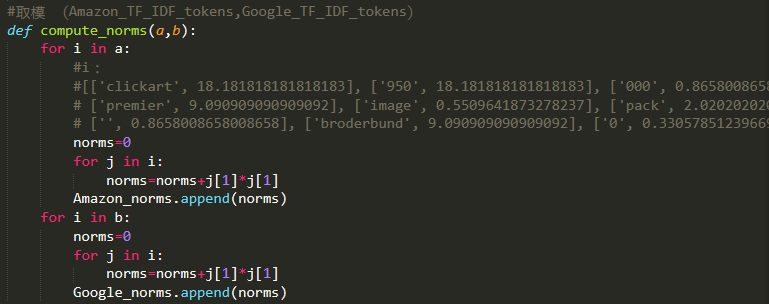
遍历Amazon的每个token，若它已经在Google的逆向索引字典中，就加上当前行号；若不在，就将当前行号存储在列表中作为值加入字典。

最后求得最佳匹配：

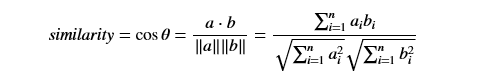


对Amazon中的每行数据，先在反向索引字典中查到所有相关的（token在Google的某一行中出现）Google行号，然后将Amazon的该行数据对Google中的这些行分别计算余弦相似度，然后取最大的余弦相似度的行号作为最佳匹配。

如果要求a,b的相似度，根据公式可知，即使我们还没确定b是哪一个，我们依然可以算出a的norm(即向量的模)，由此我们可以预先求出每条数据的norm。



注：把每一个token理解成一个维度，那么对于一条Amazon的数据和一条Google的数据，我们就可以计算二者TF-IDF的余弦值。可以把余弦值看作一种相似度。余弦值越大就意味着二者越相似。



## 问题总结

在用该算法求最佳匹配时，遇到一个问题，求出的结果中第一条就与正确结果不符，经过仔细检查，发现并不是代码问题，而是该算法存在缺陷。

Amazon中的第一条数据中有一个token是‘950’：



由于分词是按照空格，所以950和000被认为是两个token，而在计算该token的TF\_IDF值时，因为950在所有数据行中只出现了一次，这导致它的IDF值十分大。IDF值本来是为了避免那些常见词对该算法的影响，因为IDF的求法是总行数处以该token出现的行数，若一个词是常见词，那么他应该在多行数据中均存在，因此就不能作为某一条数据的“特征“，而该token若十分不常见，就会导致IDF值特别大，而恰巧在Google中有一条数据也有“950”token：



而“950”在Google数据中也只出现了一行，这导致它的IDF也十分大，而在计算Amazon的一条数据与Google中数据的余弦值时，它们分母中Amazon数据的模是一样的，然而Google数据中“950”的出现导致Google数据的模变大，例如“950”在Amazon中和Google中的IDF均为100（常见值远小于100），在计算余弦相似度时分子就多加了100\*100，而分母只除以了100，这就导致了该余弦相似度异常大，从而认为二者匹配。

所以该算法会对极其不常见的词语表现得异常敏感。