移动应用程序排名欺诈的发现

1. 前言

近年来，移动应用程序疯狂增长，APP排行榜是推动移动应用程序发展的最重要的方式。但是，随之而来的是移动应用程序的排名欺骗。本文通过挖掘活跃期——leading event和leading session，定位应用程序的排名欺骗行为，从而检测局部异常。此外，通过统计假设测试对应用程序的排名、评级、评论建模，得到基于它们的证据（有七种）。此外，还提出基于聚合的方法来整合欺骗检测的证据。最后，利用从iOS应用程序商店收集的真实数据进行评估，验证提出的系统的有效性。

1. 准备工作

确定移动应用程序的leading sessions

2.1 leading event（e）

对于移动应用程序a，有一个历史排名记录，表示为时间序列：

Leading event定义：给定阈值，时间范围,，排名满足,,对于,都有.

2.2 leading session（s）

定义：时间范围，n个邻近的leading event{e1,e2,…en}，满足，，并且无其他的s\*，使得，对于，都有。：合并主要时间的预定义时间阈值。



2.3挖掘 leading session

伪代码如下：



1. 排名欺骗检测的证据提取

应用程序在一个主要事件中的排名包含上升阶段、维持阶段、衰退阶段。

定义（一个主要事件的排序阶段）：给定一个APP a的主要事件e,事件范围是在这里，a的最高排序的位置为∈∆R。∆R：决定维持阶段的开始时间和结束时间的排名的范围。关于e的三个阶段的时间范围如下:上升阶段：[，维持阶段：[，衰落阶段：[。



3.1基于排名的证据

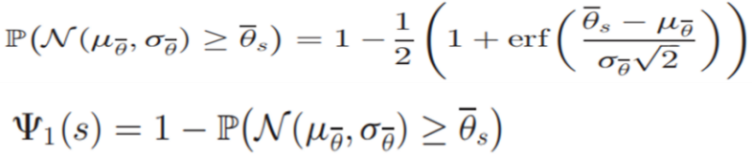
（1）evidence1



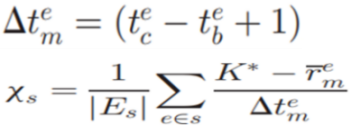


假设0：主要阶段s的指数对于排名检测无用

假设1：主要阶段s的指数明显比预期的好



1. evidence2



假设0：主要阶段s的指数对于排名检测无用

假设1：主要阶段s的指数明显比预期的高



1. evidence3

主要阶段的主要事件的个数|Es|

假设0：主要阶段s的指数|Es|对于排名检测无用

假设1：主要阶段s的指数|Es|明显比预期的大



5YF0[(~_HF4ENI84E]IX%KY

3.2基于等级的证据

（1）evidence4

为每个主要阶段定义一个欺骗指数∆Rs



假设0：主要阶段s的指数∆Rs对于排名检测无用

假设1：主要阶段s的指数∆Rs明显比预期的高

]SZ_[4HK_5LA[U_~AKY~OF4

（2）evidence5

等级的离散水平|L|，一个正常的应用程序，对于等级li的分布，P(li|Rs,a),应与a的历史等级记录的分布P(li|Ra)相一致。



余弦相似性估计两者差异，越大，越相似：



假设0：主要阶段s的指数D(s)对于排名检测无用

假设1：主要阶段s的指数D(s)明显比预期的低

3Y$6RZI7637OZHPD{C2SEEE

3.3基于评论的证据

（1）evidence6

定义评论欺骗指标Sim（s），表明主要阶段s中的评论的平均相似程度。

①规范评论c

②为评论c定义字矢量，=dim[n]，freqi,c是c中第i个字的频率

HOFOP1_%Q~[B4M{M0H23OYF

③计算评论ci和cj之间的余弦相似度Cos(,),得到Sim（s）如下：



假设0：主要阶段s的指数Sim（s）对于排名检测无用

假设1：主要阶段s的指数Sim（s）明显比预期的高

RZT0{11%0[])($)5QHW%}7F

（2）evidence7

a的一个主要阶段的话题分布P(z|a)应与一个正常的主要阶段的话题分布P(z|s)一致。用话题模型提取评论的潜在话题，隐含的狄利克雷分配（LDA）：（历史评论ca)

①ca形成之前，由Dirichletβ形成的潜在话题（∅Z)的K优先分布

②之前的潜在话题分布θa，由Dirichletα形成

③生成ca中第j个字表示为Wa,j:

θa中形成潜在话题z; ∅z中形成Wa,j



假设0：主要阶段s的指数DKL(s||a)对于排名检测无用

假设1：主要阶段s的指数DKL(s||a)明显比预期的高

`~6EB2{E3K}414NLPNZ4FIW

3.3证据聚合



假设原则1:给予与其他证据一致的证据更大的权重：

方差检测一致性：

~CIX]45ZEK)5T$`B@VTO21B

给定带有主要阶段集合{sj}的APP集合{ai},将证据聚集定位为在一个在所有主要阶段最小化证据的加权差异的优化问题



(幂梯度算法），初始：ωi=1/NΨ，每个阶段s，有



然后更新权重ωi，



假设原则2:假定有效的证据可以从一个相似的条件分布对主要阶段排序，而无效的将导致一个更均匀的随机排序分布。

对于给定的一组主要阶段，根据证据得分排序，得到NΨ排名表，为从返回的阶段s的排名，计算s的平均排名：



对于，用方差方法检测一致性：

`C2471BO3N@HH376`G$H5EJ

越小的，对应的被给予的权重越大。

1. 讨论

当前APP的排名为

（1）>K\*，不在主要阶段中，未参与排名欺骗

（2）<K\*，在一个主要事件e中，考虑特殊情况=,=0，可以被检测到。

APP的可信度，定义APP的排名欺骗得分：



一个有着更多主要阶段，很高的欺骗证据得分和很长持续事件的APP，将会有更高的APP欺骗得分。

1. 实验结果

5.1 实验数据

2010年2月2日到2012年9月17日在苹果应用排行榜的“前300免费的”和“前300付费的”中收集得到的数据集。数据集分别包含排名前300的免费应用程序和前300名付费应用的每日排行榜排名。此外，每个数据集还包含用户的等级以及评论的信息。数据集详细信息：



数据集中不同排名的应用数量的分布：

数据集中不同等级的应用程序的分布：



5.2 挖掘主要阶段

在算法1中，设定排名阈值K\*=300和阈值∅=7。

两个数据集中关于APP所包含的主要事件和主要阶段的不同的数量的APP数量的分布:





两个数据集中，关于主要阶段所包含的不同得主要事件的数量的主要阶段的个数的分布:



在每个主要阶段中，对于免费的APP来说，包含的平均主要事件的数目是1.7，对于付费的APP来说，是2.26。

5.3基于人类判断的评估

5.3.1基线

（1）将基于排序的证据用聚合原则2整合——排名RFD

（2）将基于等级的证据用聚合原则2整合——等级RFD

（3）将基于评论的证据用聚合原则2整合——评论RFD

（4）所有证据用聚合原则1整合得到——EA-RFD-1

（5）所有证据用聚合原则2整合得到——EA-RFD-2

（6）所有证据都以权重1/7整合——E-RFD

5.3.2实验设置

对于以上每个基线得到的结果（每个session的证据得分聚合结果），选择排名前50的主要阶段（例如，大部分最可疑的阶段），排名中间50的主要阶段（例如，大部分不确定的阶段）还有排名后50的主要阶段（例如，大部分正常的阶段）。将所有选定的阶段合并到一个池中，其中包含来自“顶级免费300”数据集中的281个独特应用程序的587个独特阶段，以及“Top Paid 300”数据集中的213个独特应用程序的541个独特阶段。

邀请了五位熟悉苹果App Store和移动应用程序的人力资源评估人员，以2分（即欺骗），1（即不确定）和0（即非欺骗）手动标记所选的主要阶段。所有五位评估人员同意了在前300免费的数据集中，有86个欺骗阶段和113个非欺骗阶段，在前300的付费的数据集中，有94个欺骗阶段和119个非欺骗阶段。计算了每对评估者之间的Cohen’s kappa系数在0:66到0:72之间。这表明了很一致。

用4种广泛使用的度量来评估每种方法，即Precision @ K，Recall @ K，F @ K和NDCG。（K：按照每种方法得分排序的前K名，Precision @ K：前K中是欺骗的阶段个数/(欺骗个数+不是欺骗个数); Recall @ K: 前K中是欺骗的阶段个数/欺骗阶段个数（实际）;F @ K:前两个按照公式计算; NDCG: DCG@K /IDCG@K(DCG@K:按照文章的方法得分排序; IDCG@K:按照人工评分排序)。





5.3.3 总体表现

（1）EA-RFD-2 / EA-RFD-1，一贯优于其他；

（2）EA-RFD-2在所有评估指标方面都远胜EA-RFD-1；

（3）EA-RFD-2 / EA-RFD-1优于E-RFD；

（4）E-RFD具有比排名RFD，等级RFD和评论RFD更好的检测性能；

（5）基于排名的证据比基于等级和评论的更有效。

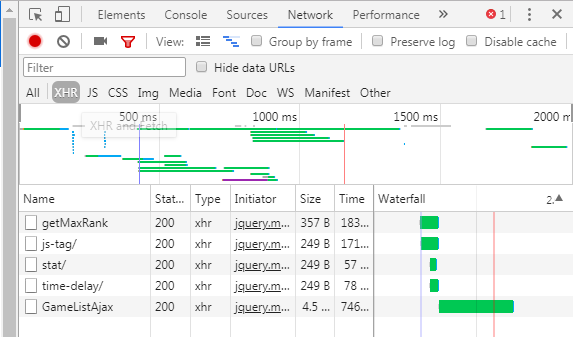
6.我的实验

6.1爬取数据（python）

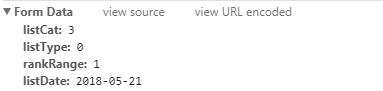
爬取数据的网址为：<http://fsight.qq.com/GameList?type=hotRank>



网页上点击F12，跳出网页源代码，选择Network，XHR，刷新，可以看到网页所用的接口列表。



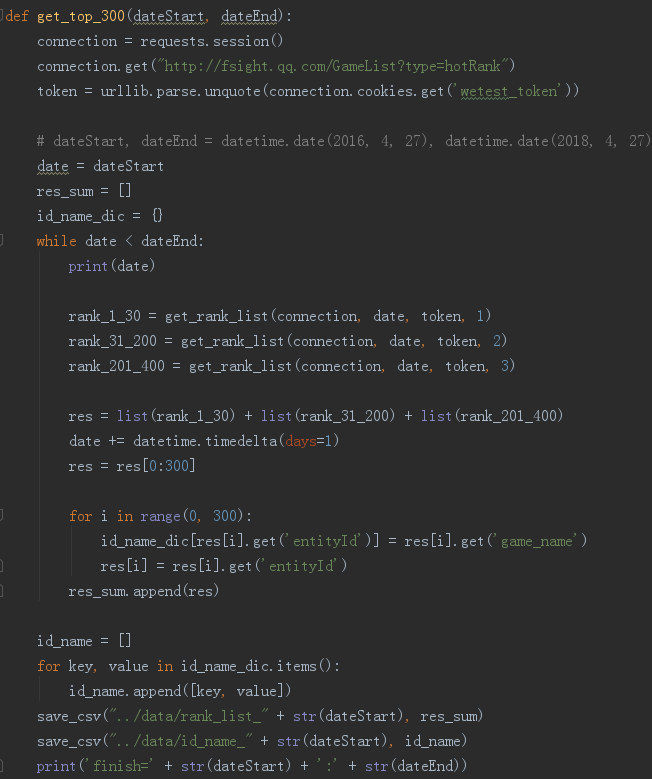
其中的GameListAjax接口就是可以获得每日排行的接口，接口的使用信息如下：



其中listCat代表的是三个榜单之一，1：免费榜 2：付费榜 3：畅销榜，listType为0代表总榜，rankRange表示的是网站的第几页，（第一页的排名为1-30，第二页为31-200等），listDate为要传入的日期。以下是该接口返回的2018年5月21日的畅销榜的第一页的排名信息（部分）：



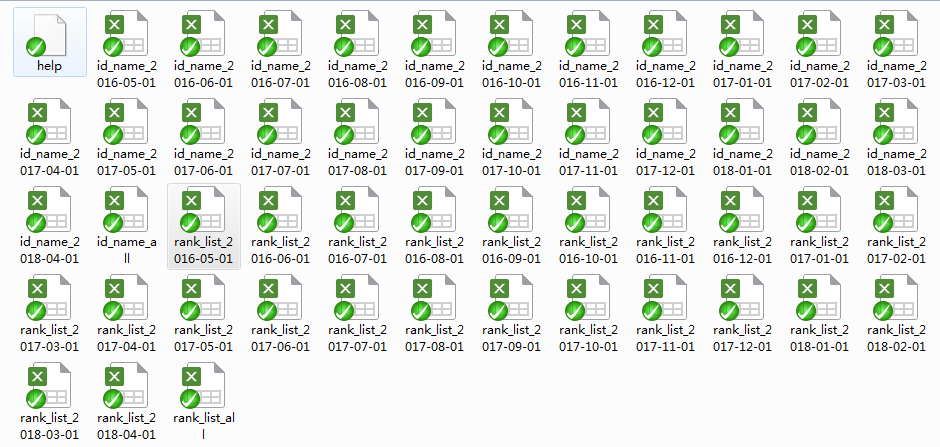
爬取数据用的是Python编写的文件crawler.py。由于代码较长，这里只展示主要部分get\_top\_300(dateStart, dateEnd)函数：



如图所示为爬取每天排名前三百名的APP的数据的函数。由于页面的限制，所以想要获得前三百名，要爬取三个页面的数据，并且在所获得的数据中截取前300。代码中的get\_rank\_list(connection, date, token, rankRange)和save\_csv(file\_name, data)为已定义的获取单个页面排名和存储csv文件的函数。在调用get\_top\_300(dateStart, dateEnd)传入开始时间和结束时间，即可获得两年的每天排名前三百的APP信息。

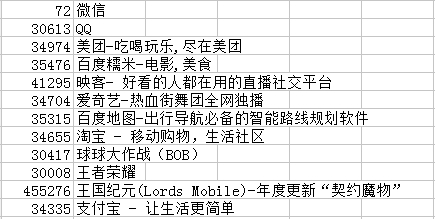
6.2处理数据

爬取的数据是两年的每天排名前300的app排行，所以在爬取数据步骤生成了如下文件（其中的id\_name\_all.csv 以及 rank\_list\_all.csv 是合成之后的文件，后续会介绍）：

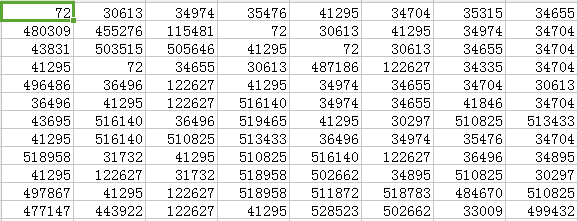


其中包括两种类型，一种是app的id对应的app名称，一种是每个月每天排名前300 的id信息。两种格式的文件部分数据如下：

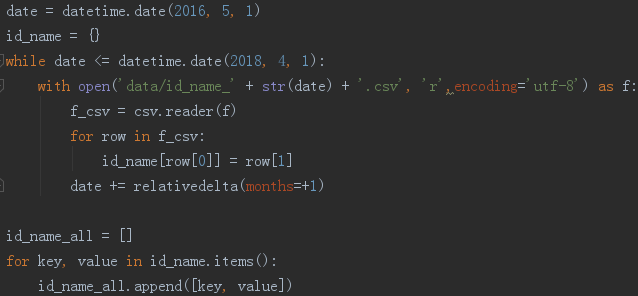
下图所示为id\_name\_2016\_05\_01.csv文件的部分数据：



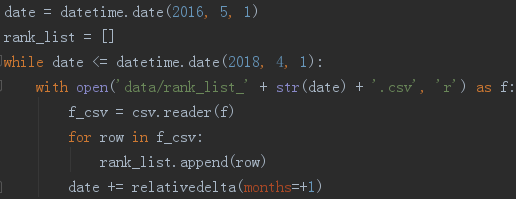
下图所示为rank\_list \_2016\_05\_01.csv文件的部分数据：



由于进行数据分析的时候，要尽量减少打开文件的次数，在此，对所获得的文件进行整合，整合代码存放在merge.py中。以下是整合id\_name文件的代码：



以下是整合rank\_list文件的代码



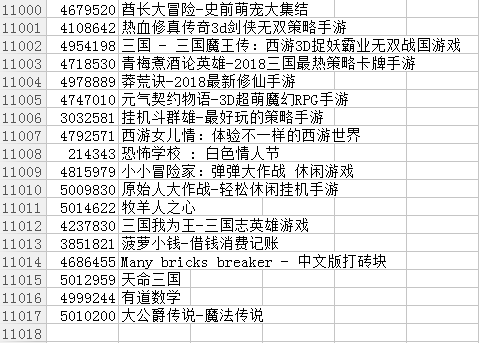
最后，调用save\_csv(file\_name, data)存储文件即可：

save\_csv('data/id\_name\_all', id\_name\_all)

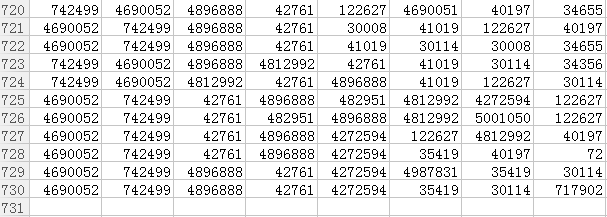
save\_csv('data/rank\_list\_all', rank\_list)

生成的文件即如图中所见的id\_name\_all.csv 以及 rank\_list\_all.csv，部分数据如下：

id\_name\_all.csv（可见共有11017个APP在2016.05.01-2018.04.30到达过前三百名）：

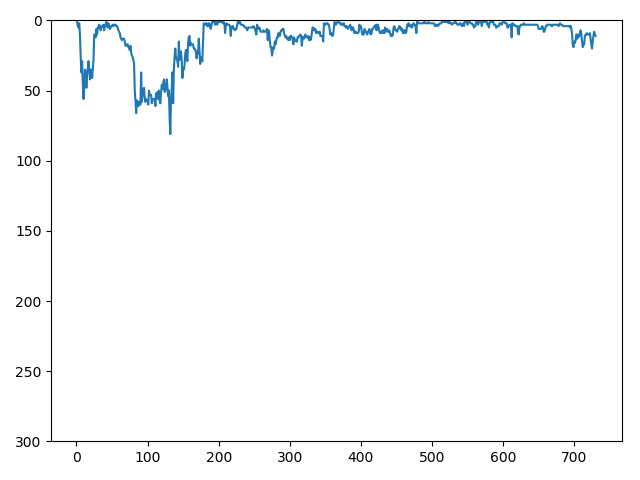


rank\_list\_all.csv（可见共有730天，每天排名前三百的APP排名信息）：

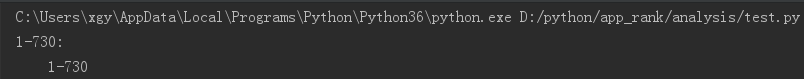


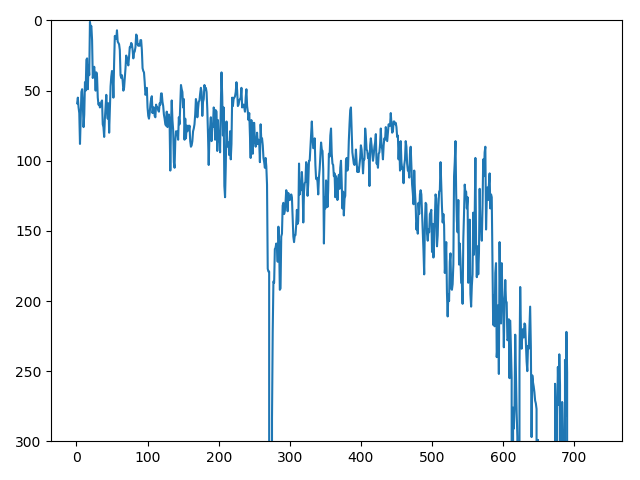
6.3挖掘leading session

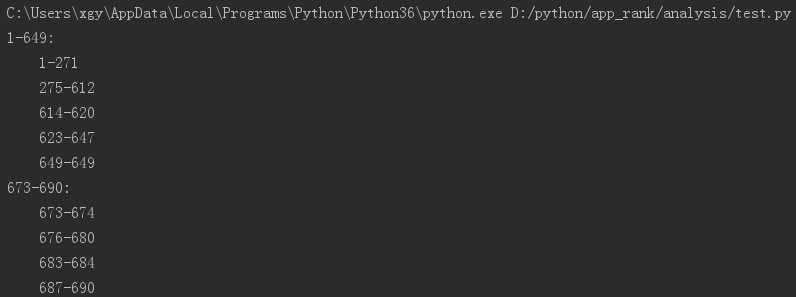
关于挖掘leading session，对文中原有的算法进行了调整。首先，通过分析得到所有的leading event,然后再根据设定的时间阈值获得leading session。挖掘leading session 的代码放在tool.py文件中。在test.py文件中传入APP的id值（这里的appId即为爬取的.csv文件中的id值），test.py的输出是该APP的排名折线图以及其leading event和leading session的时间段值，例如，输入微信的id:72,输出如下：



输入百度外卖的id:39968 ，输出如下：



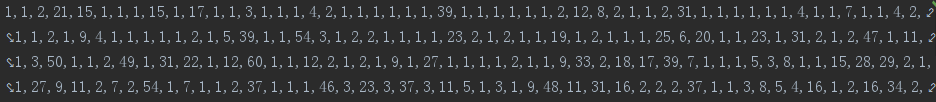




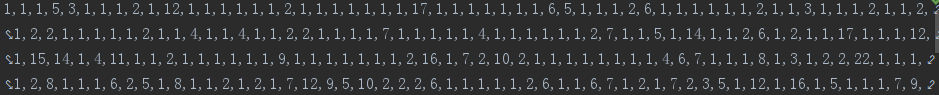
6.4 分析获得的数据

分析数据的代码存储在data\_info.py文件中，该文件共绘制了四张图片，包括不同排名的应用程序数量的分布、APP包含的event数量的分布、APP包含的session数量的分布以及session包含的event数量的分布。其中为了绘制后三个图像，使用id\_name\_all.csv 以及 rank\_list\_all.csv文件生成了event\_count.csv、session\_count.csv和session\_event\_count.csv文件，分别存储着所有APP含有的event个数，所有APP含有的session个数以及所有session含有的event个数，部分数据如下所示：

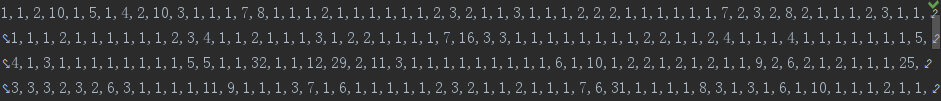
APP含有的event个数:



APP含有的session个数:

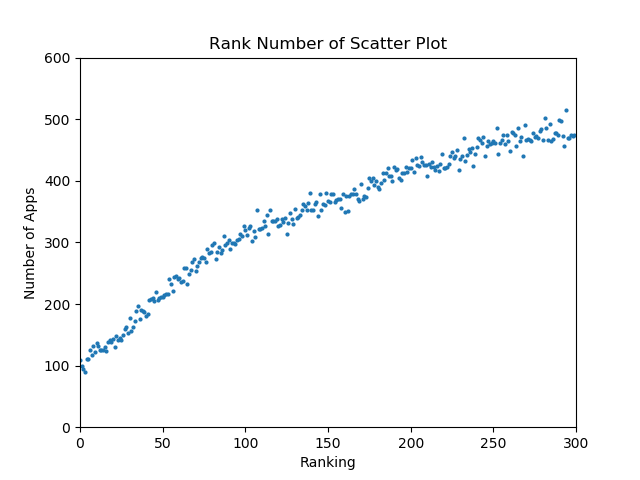


所有session含有的event个数:

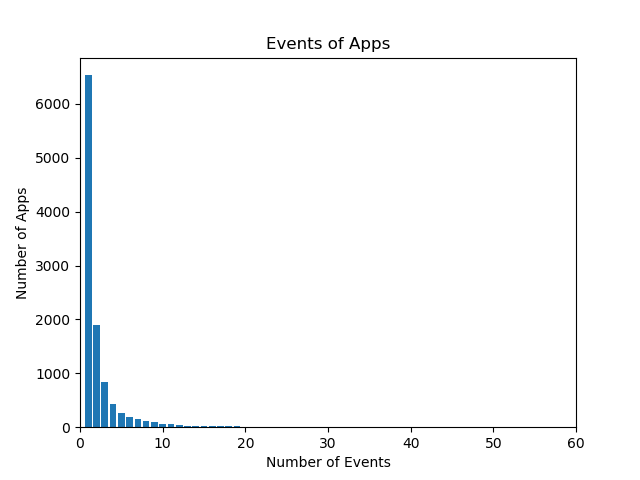


绘制的图像如下所示：

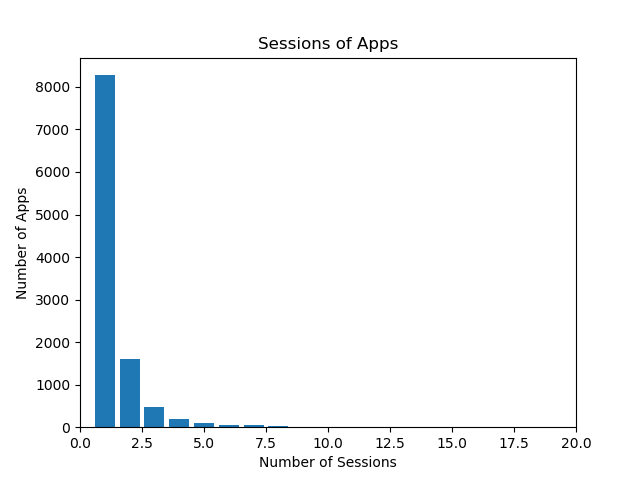
不同排名的应用程序数量的分布：



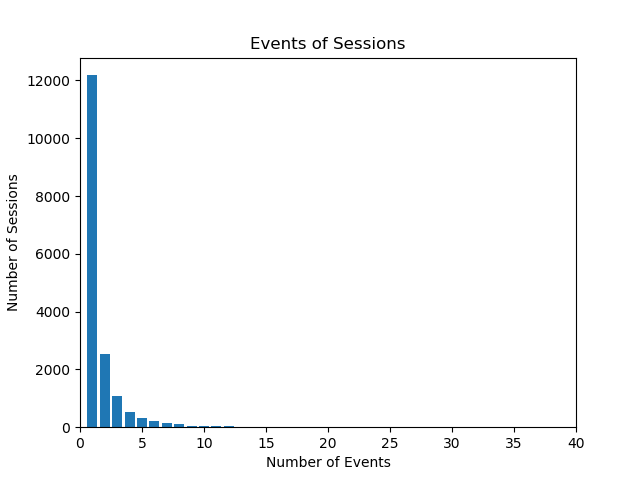
APP包含的event数量的分布：



APP包含的session数量的分布：



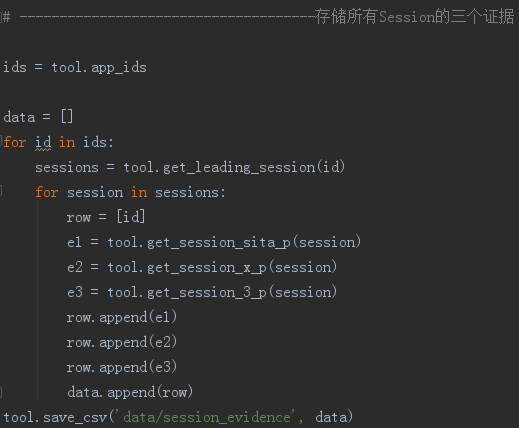
session包含的event数量的分布：



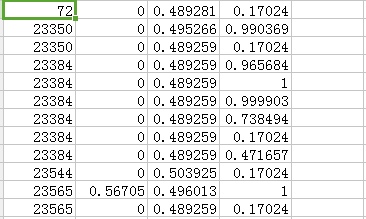
6.4 计算evidence1，evidence2，evidence3

计算得到evidence的代码主要存放在tool.py和test.py中。其中tool.py中存放着包括LeadingEvent、LeadingSession两个类、获得LeadingEvent和LeadingSession的函数以及获得LeadingSession的evidence1、evidence2、evidence3的函数，然后在test.py中通过调用这些函数就可以获得三个evidence。

存储APP-id与对应的三个证据的代码如下：

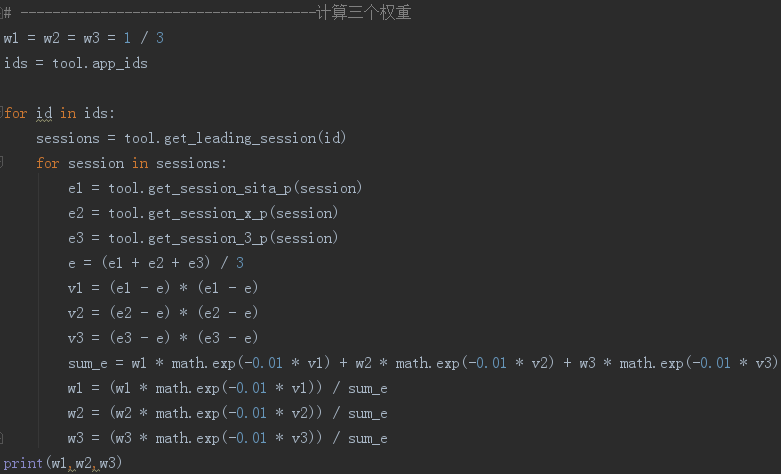


session\_evidence.csv部分内容如下：



6.5 证据聚合

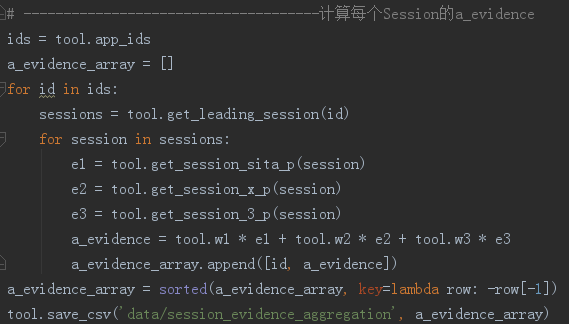
这里实现了文章中所说的基于第一种原则的证据聚合。也就是说根据evidence的一致性分配相应evidence的权重。计算三个权重的代码如下：



得到的三个权重值为：

IMG_256

然后将三个权重与三个证据聚合，得到的最终evidence按降序存储在session\_evidence\_aggregation.csv文件中，session\_evidence\_aggregation.csv文件内容如下：



session\_evidence\_aggregation.csv文件部分内容如下：

