Spark 在反作弊聚类场景的实践

周奥特 · 8 个月前

目前知乎站内的 spammer 为了快速取得收效，往往倾向于大批量地产生相似的 spam 内容，或者密集地产生特定的行为。针对这种大量，相似，和相对聚集的特点，我们最近开始尝试使用聚类的方式去发现和挖掘 spammer。 anti-spam 现阶段使用到聚类的场景主要有面向内容和行为的聚类。

聚类的目的在于把相似的内容和行为聚集在一起。常见的聚类方法有 k-means, 层次聚类。另外还有基于密度和图的聚类分析方案。

> 聚类分析仅根据在数据中发现的描述对象及其关系的信息，将数据对象分组。其目标是，组内的对象相互之间是相似的（相关的），而不同组之间的对象是不同的（不相关的）。组内的相似性（同质性）越大，组间差别越大，聚类就越好。
> **《数据挖掘导论》**

从上面的定义来看，相似度的度量是聚类的关键之一，常见的相似度算法有 edit distance, conscine similarity， Jaccard 相似度，pearson 相关系数等。本次聚类我们使用了一些文本相似度的算法，主要包括 jaccard 和 sim-hash.

Jaccard

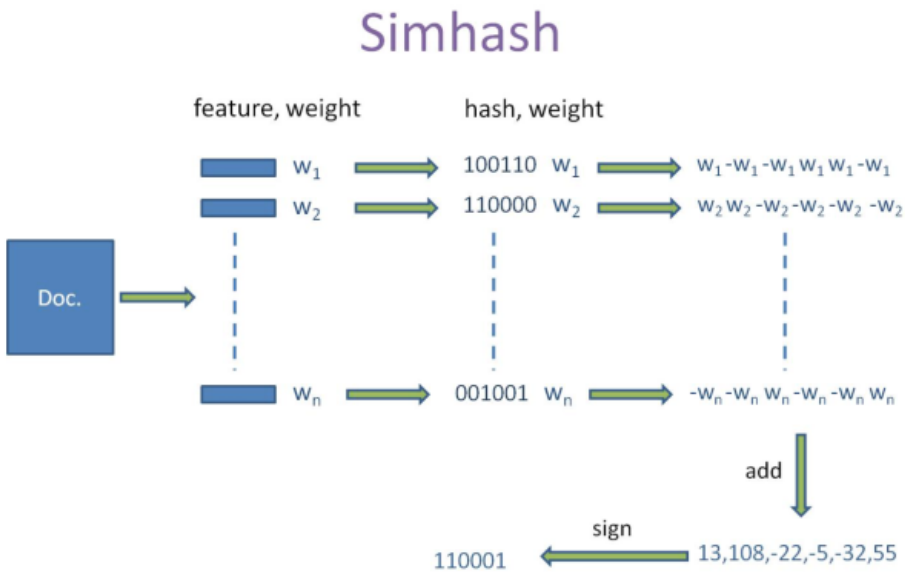
Jaccard 相似度以两个集合交集占并集的比例作为两个集合的相似度，e.g. 集合 A, B 的相似度 J(A, B) 可以表示成：

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

sim-hash

然而对于数据量比较大的场景下，jaccard 的表现差强人意， 于是我们将尝试使用 sim-hash。 sim-hash 由 Charikar 在 2002 年提出，后续在 google 被得以应用，用于近似网页的检测。sim-hash 为输入的文本生成一个 n 位的指纹，与传统的 MD5, SHA-1 这类哈希函数不同的是，对于近似的文本，sim-hash 生成的指纹也近似; 越近似的文本，其指纹不同的二进制位数（记为 k）越少。对于两个文本，比较其 sim-hash 相似度的步骤如下：

1. **分词**，对文本进行分词，为了减少停用词和其他常见词（e.g. 的，是，在..）带来的影响，使用 tf-idf（Term Frequency-Inverse Document Frequency） 为每个词增加权重，这里 tf 指的是某个词在该文本中出现的频率，idf 则与一个词的常见程度相关（即包含这个词的文本数），越常见的词，其 idf 值越低。tf-idf 权重就是 tf 与 idf 的乘积，在一段文本中，tf-idf 权重相对高的词就成为了这段文本的关键词。更详细的解释请参考 这篇文章。
2. **hash**，计算每个词的 hash 值，通过将文字转换成数字来提高计算效率。
3. **加权**，将 hash 中的 1 乘以正数的权重，0 乘以负数的权重。
4. **合并**，将加权后的hash值按列相加，得到一个数字组成的序列。
5. **降维**，将步骤 4 得到的数字序列转换成 0，1 串，大于 0 的数字转换成 1，小于 0 的数字转换成 0。
6. **相似度比较**，比较生成的 sim-hash 值的 hamming distance。hamming distance 即两个 hash 值的汉明距离，相信大家都很熟悉了，更多的介绍可以参考链接里面的解释。



simhash 生成过程示意图

我们测试了两种方法运行 100w 次的时间耗费，测试代码如下：

```
def test(): Unit ={
  var s1 = "这是一个测试测试测试啦，哈哈哈哈哈哈" ;
  var s2 = "这是一个测试测试测试哈，啦啦啦啦啦啦" ;
```

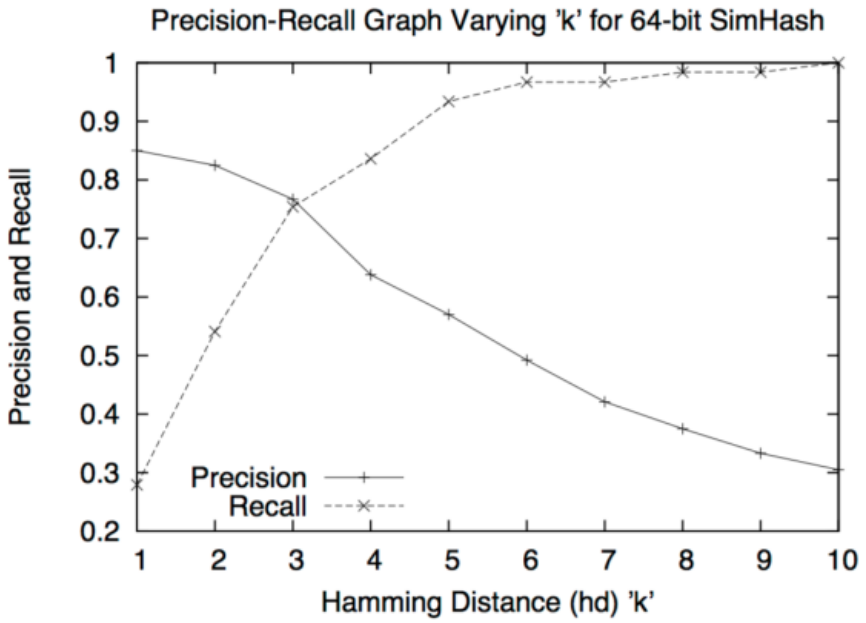
```
    var t1 = System.currentTimeMillis();
    for (i <- 0 to 1000000) {
      var dis = ZSimilarity.jaccard(s1.split(""), s2.split(""));
    }
    var t2 = System.currentTimeMillis();
    println("jaccard 耗费时间： " + (t2 - t1) + " ms ");

    t1 = System.currentTimeMillis();
    val hash_s1 = ZSimHash.hash(s1, 64)
    val hash_s2 = ZSimHash.hash(s2, 64)
    for (i <- 0 to 1000000) {
      var dis = ZSimHash.hammingDistance(hash_s1, hash_s2, 64);
    }
    t2 = System.currentTimeMillis();
    println("sim-hash 耗费时间： " + (t2 - t1) + " ms ");
  }
```

结果显示：

```
jaccard 耗费时间： 21772 ms
sim-hash 耗费时间： 9981 ms
```

在文本较短的情况下，sim-hash 可以提升至少一半的检测效率；检测长文本的差距更为明显。所以采用 sim-hash 可以有效的缩短相似度检测的时间。

经过试验，对于 64 位的 sim-hash 指纹，k=3 是判断两个文本是否相似比较合理的阈值，因为在 k=3 的时候, 召回率和准确率都能处在一个比较满意的水平（75%左右）。为了提高召回和准确率，在实践当中，我们使用 k=4 来保证召回，并在 sim-hash 的基础上，对每一组再次进行一次 jaccard 相似度计算，提高其准确率。



不同阈值下的准确率和召回率曲线

目前站内每天 web 端产生近千万的写行为。拿私信举例， 如果用单进程去比较每天 10w 条私信的相似度，每一条私信需要和其他 99999 条私信进行两两比较，根据实验数据, 使用 sim-hash 一次遍历需要近 1s， 将 10w 的数据全都检测一遍则需要接近 27个小时。在这种场景

下，如何有效，快速地对全量的数据进行聚类呢?

spark 是目前我们采用的方案。spark，是一个高性能分布式计算框架。spark 的计算是基于内存的，spark 支持将计算的中间结果和数据集存储在内存，大大减少了磁盘 io，网络通信的时间。与 map-reduce 模型相比，提供了更为丰富的算子 （e.g. filter，flatMap等）。这些算子被分成两类，转换（transformation）和执行 (action)，所有的 transformation 操作具有延迟性，当一个 transformation 操作被调用时，计算不会立即触发，只有 action 被调用时，计算才会被触发。这一点也良好的保证了 spark 的容错性，当一个 task 在某个节点挂掉时，在一个新的节点重新进行计算的成本不会很高。

内容聚类

- 数据准备：

在使用 spark 之前，数据准备是由 HiveQL 结合 python 脚本完成的，在数据量大的情况下，效率存在问题。而 spark 可以与 hive 进行无缝整合，因此数据处理的效率提升了不少。在 hive 上，HiveQL 实际上被转换成一系列的 map-reduce 过程，在 hadoop 平台上执行计算；而在 spark 上执行 hive 语句，HiveQL 会被转化成一系列的 transformation 和 action。spark 会直接读取 hive 的元数据，将元数据转化成 spark RDD, 并在此基础上进行计算。得益于 spark 丰富的算子和基于内存的特点，加上原来由 python 脚本完成的数据清洗工作也可以由 spark 的算子来代替完成，spark sql 的执行效率要比原来采用 hiveQL 的执行效率高至少 10 倍有余。

- 聚类实现：

内容聚类的实现采用了图分割的方式，即构建一个相似度图 G=(V, E)，以每个文档为顶点，文档之间的相似度作为相连边的权重。以 sim-hash 为例，两点之间相连边的权重就是两个 hash 值的 hamming distance。如下图所示，假如我们以 k=3 作为阈值，取所有权重大于阈值的边作为新的子图（即图中黑色的边），并计算子图中的连通子图即可得到（1，2，3）和（4，5，6，7）两个cluster。

在实际使用中，我们本来使用 Graphx（spark 对于图和图的并行计算的 api，详情见GraphX | Apache Spark） 提供的 connectedComponents 接口，但是后续发现在数据量比较大的情况下，反复的迭代带来了比较大的性能问题。于是利用文本相似度的传播性（a与b相似，且 b与c相似，则a与c相似），我们使用 spark SQL 将问题转化为 “寻找最小相似节点” 的问题。举个例子，在下图中，与1相似的最小节点是1，与2相似最小的节点为1，与3相似最小的节点也为1，这三个点最小相似节点均为1，所以他们属于同一个 cluster。而 4，5，6，7 这四个节点，因为最小相似节点为 4，所以他们属于另外一个独立的 cluster。

cluster 分割示意图

由于 Jaccard 相似度计算成本比较高，实践中使用 sim-hash 来提升相似度计算的效率。使用 64 位哈希值，将 k=4 作为阈值，将输入的数据进行一个预先分组。由于这种条件下可以保证比较高的召回，而准确率相对来说比较低， 需要再针对每个分组使用 Jaccard 进行细分，进而提高准确度。这种方式减少了需要计算 Jaccard 相似度的数量，也弥补了 sim-hash 相似度在召回高时准确率不足的问题。另外，为了减少不必要的计算资源浪费，相连两个节点的相似度只单向计算了一次。但是相似度的比较仍然是一个近似笛卡尔积的计算，为了提高这部分的计算效率，我们采用 spark 的广播机制，在所有节点的内存缓存一份变量，从而减少在计算过程中的通信开销。

目前针对私信，聚类可以在 1-3 min 之内完成，使用 spark 充分提高了数据处理的效率。

行为聚类

行为聚类的主要思路是将用户的行为路径以文本的方式表达出来，将行为聚类转换成内容聚类，通过文本相似度聚类，将相似的行为聚集在一起。

- 数据准备：

行为路径的表达：以用户的一个 post 行为为单位，取前后至少两个请求，并计算每个行为之间的时间间隔，将用户的行为序列表达成由“请求路径|请求方法|与上一次请求的时间间隔|”构成的文本组合。

- 聚类实现：

相对于内容聚类，行为聚类面对着更大的数据量, 在数据清洗过后大概每天有 30w+ 的关键业务写行为。考虑到比较不同类别的写行为之间的相似度没有太大意义，因此针对每个业务单独进行聚类，这样一来将 30w * 30w 的计算规模减少到了 1w * 1w + 3w * 3w + ...。由于聚类的实现与内容聚类的逻辑大体一致，这里就不再多做介绍。

总结：针对批量的 spammer 内容和行为，聚类是一种替代人工策略行之有效的方法。目前行为和内容聚类均以离线处理的方式上线，聚类是 antispam 使用 spark 的初步尝试，后续会继续优化提高其处理效率，也会尝试使用 spark streaming 来提高聚类的实时性。

作者：周奥特 孙先陈磊

同时感谢反作弊团队其他同学的帮助

Reference

[1] Detecting Near-Duplicates for Web Crawling

[2] TF-IDF与余弦相似性的应用

反作弊 Spark

文章被以下专栏收录

Hacker's log
知乎技术日志

进入专栏

32 条评论

写下你的评论

李明亮

感觉可以投稿到
https://zhuanlan.zhihu.com/hackers

8 个月前 4 赞

周奥特（作者） 回复 李明亮 查看对话

投稿中😀

8 个月前

Comzyh

汉明距离最小我记得有算法的呀，只靠Spark硬刚能行吗...

8 个月前

Alex.黄太厚

酷炫啊，朝大数据量迈进一小步！

8 个月前

北软的猫

手动点赞，没抢到沙发

8 个月前

吴珊珍

捕捉到一列攻城狮，并关注之 (*￣▽￣*)

8 个月前

穆琳

呀，悟空师傅的真身终于冒泡了😝

8 个月前 1 赞

刘大头

赞，前排全是熟人。。。

8 个月前 2 赞

hh15

从你们莫名其妙把我账号当作spam没有负过一点责任可以看出，悟空这反作弊系统可能质

量堪忧

8 个月前

JianMing

=.=你冒泡了

8 个月前

1 2 3 4 下一页

推荐阅读

【法律百科】婚内男方有权阻止女方中止妊娠吗

作者/袁霄霄 陕西学高律师事务所律师明天，你好牛奶咖啡 - Lost & Found去寻... 查看全文 >

袁霄霄 · 1 个月前 · 编辑精选 · 发表于 袁律说法

《不义联盟2》评测 DC粉丝盛宴2.0版

偏粉丝向的格斗游戏，最好是能给出一个合适的理由，来让那些本来可能是队友的角色互相对... 查看全文 >

战术大米 · 25 天前 · 编辑精选 · 发表于 游民星空独家专栏

VC迹象投资法 （心得 丁敏 v2017.5.29）

前言阅读《Venture Capitalists at Work》让我受益匪浅，个人总结出一套VC投资方法论，暂且... 查看全文 >

丁敏 · 22 天前 · 编辑精选 · 发表于 环保行业私募股权投资

第70届戛纳电影节整体扑街？笑话！

又是一次诈尸，差不多有一个多月没有更新过公众号了，感谢没有取关还能看见这条推送的你们。... 查看全文 >

郭连凯 · 17 天前 · 编辑精选 · 发表于 迷影至下