**SimHash+Hamming**

背景

如何设计一个比较两篇文章相似度的算法？可能你会回答几个比较传统点的思路：

* 一种方案是先将两篇文章分别进行分词，得到一系列特征向量，然后计算特征向量之间的距离（可以计算它们之间的欧氏距离、海明距离或者夹角余弦等等），从而通过距离的大小来判断两篇文章的相似度。
* 另外一种方案是传统hash，我们考虑为每一个web文档通过hash的方式生成一个指纹（finger print）。

下面，我们来分析下这两种方法。

* 采取第一种方法，若是只比较两篇文章的相似性还好，但如果是海量数据呢，有着数以百万甚至亿万的网页，要求你计算这些网页的相似度。你还会去计算任意两个网页之间的距离或夹角余弦么？想必你不会了。
* 而第二种方案中所说的传统加密方式md5，其设计的目的是为了让整个分布尽可能地均匀，但如果输入内容一旦出现哪怕轻微的变化，hash值就会发生很大的变化。

举个例子，我们假设有以下三段文本：

* the cat sat on the mat
* the cat sat on a mat
* we all scream for ice cream

使用传统hash可能会得到如下的结果：

* irb(main):006:0> p1 = 'the cat sat on the mat'
  + irb(main):007:0> p1.hash => 415542861
* irb(main):005:0> p2 = 'the cat sat on a mat'
  + irb(main):007:0> p2.hash => 668720516
* irb(main):007:0> p3 = 'we all scream for ice cream'
  + irb(main):007:0> p3.hash => 767429688 "

可理想当中的hash函数，需要对几乎相同的输入内容，产生相同或者相近的hash值，换言之，hash值的相似程度要能直接反映输入内容的相似程度，故md5等传统hash方法也无法满足我们的需求。

出世

车到山前必有路，来自于GoogleMoses Charikar发表的一篇论文“detecting near-duplicates for web crawling”中提出了simhash算法，专门用来解决亿万级别的网页的去重任务。

simhash作为locality sensitive hash（局部敏感哈希）的一种：

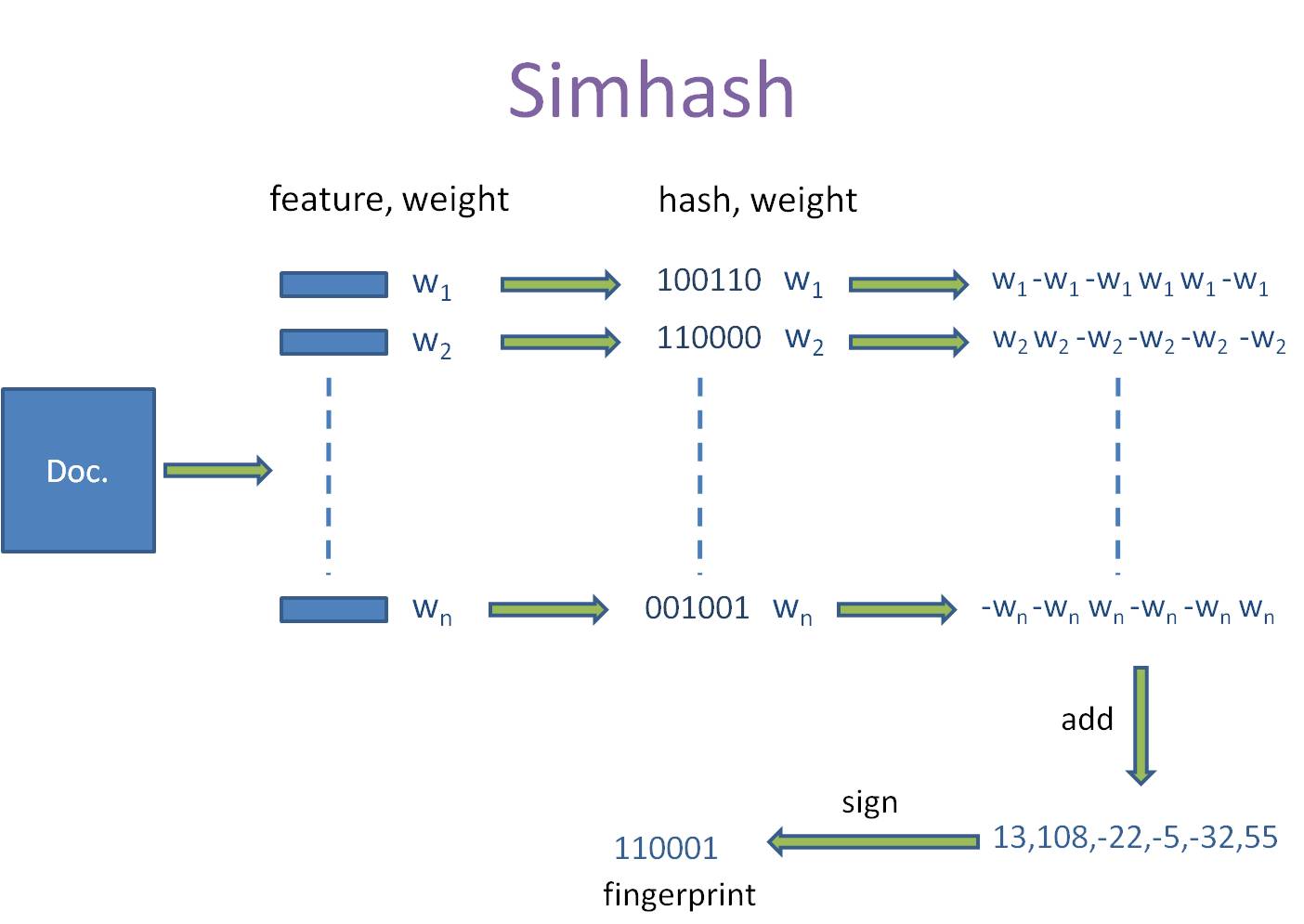
* 其主要思想是降维，将高维的特征向量映射成低维的特征向量，通过两个向量的Hamming Distance来确定文章是否重复或者高度近似。
  + 其中，Hamming Distance，又称汉明距离，在信息论中，两个等长字符串之间的汉明距离是两个字符串对应位置的不同字符的个数。也就是说，它就是将一个字符串变换成另外一个字符串所需要替换的字符个数。例如：1011101 与 1001001 之间的汉明距离是 2。至于我们常说的字符串编辑距离则是一般形式的汉明距离。

如此，通过比较多个文档的simHash值的海明距离，可以获取它们的相似度。

流程

simhash算法分为5个步骤：分词、hash、加权、合并、降维，具体过程如下所述：

* 分词
  + 给定一段语句，进行分词，得到有效的特征向量，然后为每一个特征向量设置1-5等5个级别的权重（如果是给定一个文本，那么特征向量可以是文本中的词，其权重可以是这个词出现的次数）。例如给定一段语句：“CSDN博客结构之法算法之道的作者July”，分词后为：“CSDN 博客 结构 之 法 算法 之 道 的 作者 July”，然后为每个特征向量赋予权值：CSDN(4) 博客(5) 结构(3) 之(1) 法(2) 算法(3) 之(1) 道(2) 的(1) 作者(5) July(5)，其中括号里的数字代表这个单词在整条语句中的重要程度，数字越大代表越重要。
* hash
  + 通过hash函数计算各个特征向量的hash值，hash值为二进制数01组成的n-bit签名。比如“CSDN”的hash值Hash(CSDN)为100101，“博客”的hash值Hash(博客)为“101011”。就这样，字符串就变成了一系列数字。
* 加权
  + 在hash值的基础上，给所有特征向量进行加权，即W = Hash \* weight，且遇到1则hash值和权值正相乘，遇到0则hash值和权值负相乘。例如给“CSDN”的hash值“100101”加权得到：W(CSDN) = 100101 4 = 4 -4 -4 4 -4 4，给“博客”的hash值“101011”加权得到：W(博客)=101011 5 = 5 -5 5 -5 5 5，其余特征向量类似此般操作。
* 合并
  + 将上述各个特征向量的加权结果累加，变成只有一个序列串。拿前两个特征向量举例，例如“CSDN”的“4 -4 -4 4 -4 4”和“博客”的“5 -5 5 -5 5 5”进行累加，得到“4+5 -4+-5 -4+5 4+-5 -4+5 4+5”，得到“9 -9 1 -1 1”。
* 降维
  + 对于n-bit签名的累加结果，如果大于0则置1，否则置0，从而得到该语句的simhash值，最后我们便可以根据不同语句simhash的海明距离来判断它们的相似度。例如把上面计算出来的“9 -9 1 -1 1 9”降维（某位大于0记为1，小于0记为0），得到的01串为：“1 0 1 0 1 1”，从而形成它们的simhash签名。

其流程如下图所示：[](https://camo.githubusercontent.com/efd5caf946abb03b9fe83ccb1ab85208d32d1c60/687474703a2f2f646c2e69746579652e636f6d2f75706c6f61642f6174746163686d656e742f3433373432362f62616634323337382d653632352d333564322d396138392d3437313532346133353564382e6a7067)

应用

* 每篇文档得到SimHash签名值后，接着计算两个签名的海明距离即可。根据经验值，对64位的 SimHash值，海明距离在3以内的可认为相似度比较高。
  + 海明距离的求法：异或时，只有在两个比较的位不同时其结果是1 ，否则结果为0，两个二进制“异或”后得到1的个数即为海明距离的大小。

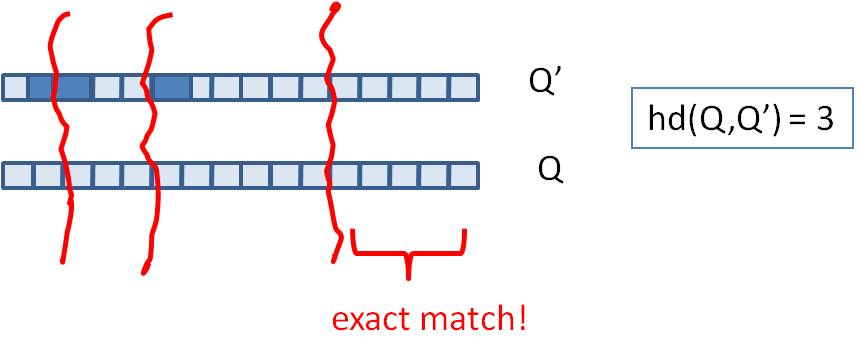
举个例子，上面我们计算到的“CSDN博客”的simhash签名值为“1 0 1 0 1 1”，假定我们计算出另外一个短语的签名值为“1 0 1 0 0 0”，那么根据异或规则，我们可以计算出这两个签名的海明距离为2，从而判定这两个短语的相似度是比较高的。

换言之，现在问题转换为：对于64位的SimHash值，我们只要找到海明距离在3以内的所有签名，即可找出所有相似的短语。

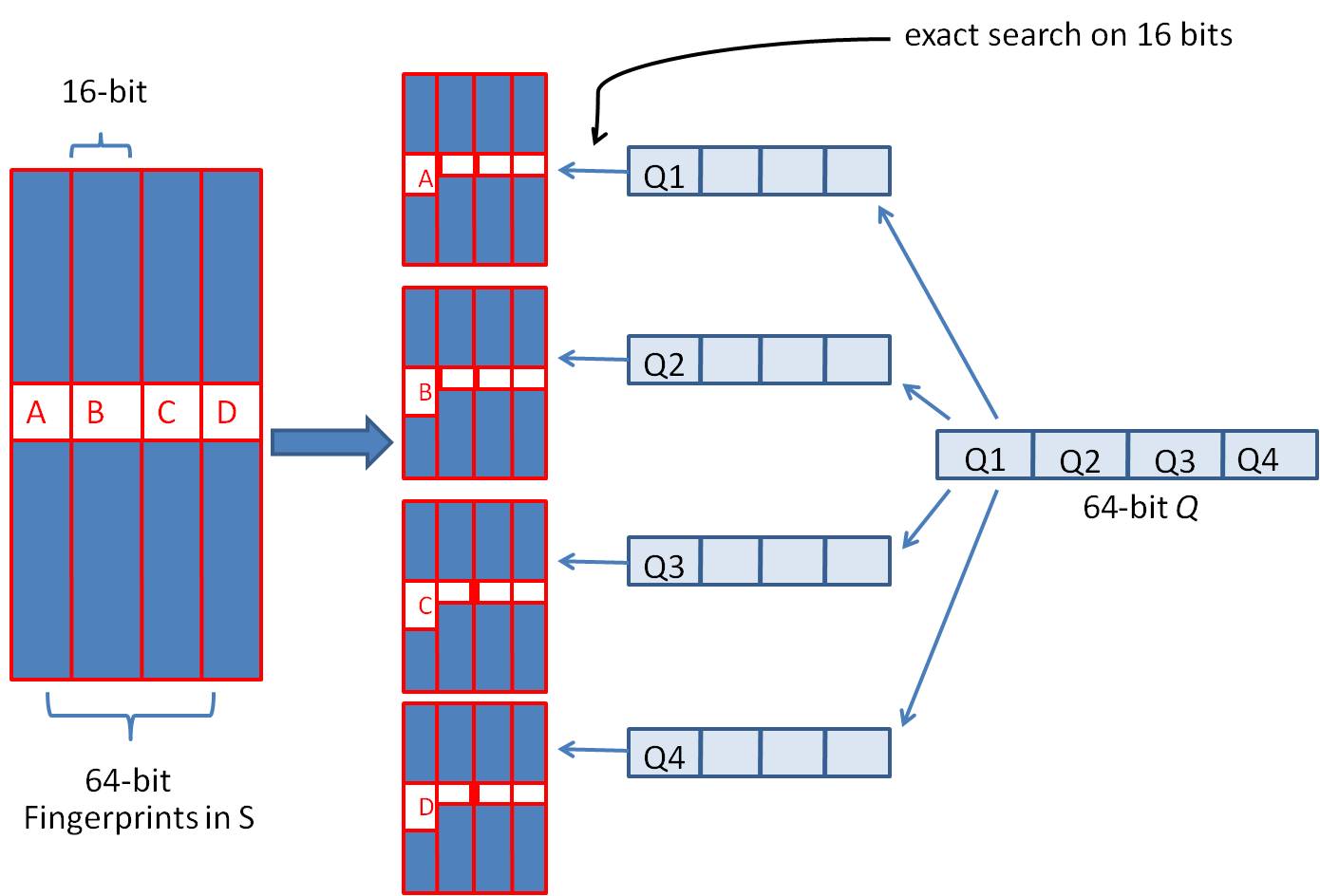
但关键是，如何将其扩展到海量数据呢？譬如如何在海量的样本库中查询与其海明距离在3以内的记录呢？

* 一种方案是查找待查询文本的64位simhash code的所有3位以内变化的组合
  + 大约需要四万多次的查询。
* 另一种方案是预生成库中所有样本simhash code的3位变化以内的组合
  + 大约需要占据4万多倍的原始空间。

这两种方案，要么时间复杂度高，要么空间复杂度复杂，能否有一种方案可以达到时空复杂度的绝佳平衡呢？答案是肯定的：

* 我们可以把 64 位的二进制simhash签名均分成4块，每块16位。根据鸽巢原理（也称抽屉原理），如果两个签名的海明距离在 3 以内，它们必有一块完全相同。如下图所示：[](https://camo.githubusercontent.com/d6cc444c5a3db896a0b3aae6333cb6dcc42598cc/687474703a2f2f646c2e69746579652e636f6d2f75706c6f61642f6174746163686d656e742f3433373535392f36383937313964662d353462372d333138632d626339302d6532383966383433343462392e6a7067)
* 然后把分成的4 块中的每一个块分别作为前16位来进行查找，建倒排索引。

具体如下图所示：

[](https://camo.githubusercontent.com/52160c9c5b3160034d76d1ccd0e9e7f514b46c80/687474703a2f2f646c2e69746579652e636f6d2f75706c6f61642f6174746163686d656e742f3433373538362f62373262386463322d393133392d333037382d616432342d6236383966363466643731612e6a7067)

如此，如果样本库中存有2^34（差不多10亿）的simhash签名，则每个table返回2^(34-16)=262144个候选结果，大大减少了海明距离的计算成本。

* 假设数据是均匀分布，16位的数据，产生的像限为2^16个，则平均每个像限分布的文档数则为2^34/2^16 = 2^(34-16)) ，四个块返回的总结果数为 4\* 262144 （大概 100 万）。
  + 这样，原本需要比较10亿次，经过索引后，大概只需要处理100万次。

（部分内容及图片参考自： <http://grunt1223.iteye.com/blog/964564>，后续图片会全部重画）