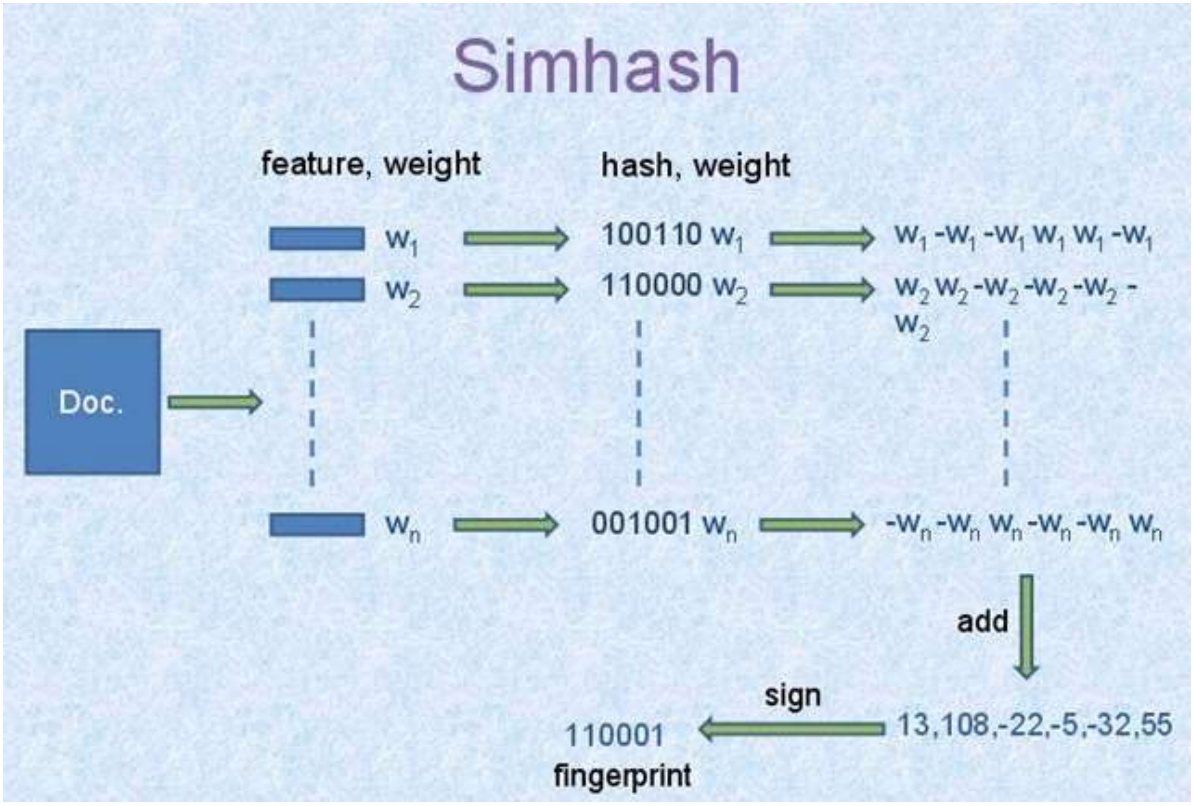
30 Jan 2014

simhash算法原理及实现

simhash是google用来处理海量文本去重的算法。 google出品，你懂的。 simhash最牛逼的一点就是将一个文档，最后转换成一个64位的字节，暂且称之为特征字，然后判断重复只需要判断他们的特征字的距离是不是<n（根据经验这个n一般取值为3），就可以判断两个文档是否相似。

原理

simhash值的生成图解如下



大概花三分钟看懂这个图就差不多怎么实现这个simhash算法了。特别简单。谷歌出品嘛，简单实用。

算法过程大概如下：

1. 将Doc进行关键词抽取(其中包括分词和计算权重)，抽取出n个(关键词，权重)对， 即图中的`(feature, weight)`们。记为 `feature_weight_pairs` = [fw1, fw2 ... fwn]，其中 `fwn = (feature_n, weight_n`)。
2. `hash_weight_pairs` = [(hash(feature), weight) for feature, weight in

`feature_weight_pairs`] 生成图中的`(hash,weight)`们, 此时假设hash生成的位数`bits_count = 6` (如图)；

3. 然后对 `hash_weight_pairs` 进行位的纵向累加，如果该位是1，则`+weight`,如果是0，则`-weight`，最后生成`bits_count`个数字，如图所示是`[13, 108, -22, -5, -32, 55]`, 这里产生的值和hash函数所用的算法相关。
4. `[13,108,-22,-5,-32,55] -> 110001`这个就很简单啦，正1负0。

到此，如何从一个doc到一个simhash值的过程已经讲明白了。 但是还有一个重要的部分没讲，

『simhash值的海明距离计算』

二进制串A 和 二进制串B 的海明距离 就是`A xor B` 后二进制中1的个数。

举例如下:

```
A = 100111;
B = 101010;
hamming_distance(A, B) = count_1(A xo
```

当我们算出所有doc的simhash值之后，需要计算doc A和doc B之间是否相似的条件是：

A和B的海明距离是否小于等于n，这个n值根据经验一般取值为3，

simhash本质上是**局部敏感性的hash**，和md5之类的不一样。 正因为它的局部敏感性，所以我们可以使用海明距离来衡量simhash值的相似度。

『高效计算二进制序列中1的个数』

```
/* src/Simhasher.hpp */
bool isEqual(uint64_t lhs, uint64_t r
{
    unsigned short cnt = 0;
```

```
    lhs ^= rhs;
    while(lhs && cnt <= n)
    {
        lhs &= lhs - 1;
        cnt++;
    }
    if(cnt <= n)
    {
        return true;
    }
    return false;
}
```

由上式这个函数来计算的话，时间复杂度是 O(n); 这里的n默认取值为3。由此可见还是蛮高效的。

『计算二进制序列中1的个数之O(1)算法实现』

感谢 @SCatWang 的评论分享:

> 感谢您做的simhash库，感觉会很方便。 有关求二进制中1的个数，其实有各种O(1)的实现。可以参考这个地方：http://stackoverflow.com/a/14682688

simhash 实现的工程项目

- C++ 版本 simhash
- Golang 版本 gosimhash

主要是针对中文文档，也就是此项目进行simhash之前同时还进行了分词和关键词的抽取。

对比其他算法

『百度的去重算法』

百度的去重算法最简单，就是直接找出此文章的最长的n句话，做一遍hash签名。n一般取3。 工程实现巨简单，据说准确率和召回率都能到达80%以上。

『shingle算法』

shingle原理略复杂，不细说。 shingle算法我认为过于学院派，对于工程实现不够友好，速度太慢，基本上无法处理海量数据。

『其他算法』

具体看微博上的讨论

参考

- Similarity estimation techniques from rounding algorithms
- simhash与Google的网页去重
- 海量数据相似度计算之simhash和海明距离

转载请注明出处: simhash算法原理及实现