# 相似度发现

在大数据领域中，需要处理的数据一般是海量并且是高纬度的非结构化数据，怎么快速的从海量高纬度的数据中找到与某个数据最相似的数据或者多条数据以及成为一个难点。在低纬度的小数据集上，我们可以通过线性查找（Linear Search）就可以容易解决，但是如果是对海量高纬度的数据采用这样的查找，往往会非常耗时，因此为了解决这一个问题，我们需要采用一些类似索引的技术来加快查找过程，通常这类技术称为最近邻查找（Nearest  Neighbor,AN），例如K-d tree；或近似最近邻查找（Approximate Nearest  Neighbor, ANN），例如K-d tree with BBF, Randomized Kd-trees, Hierarchical K-means Tree。而LSH是ANN中的一类方法。

## 算法流程

在做相似度查询时，选择一个合适的距离公式是第一步，也是关键性一步。一般选择依据多种多样，其中一种依据就是数量的格式，距离公式一般有如下几种。

#### 欧式距离

在纬欧式空间中，每个点是一个纬实数向量。在该空间中的传统的距离度量其实就是 范式( )，定义如下：



范式其实就是曼哈顿距离，如果是范式，那么公式如下：



#### Jaccard距离

集合的Jaccard距离公式如下：



这里 。

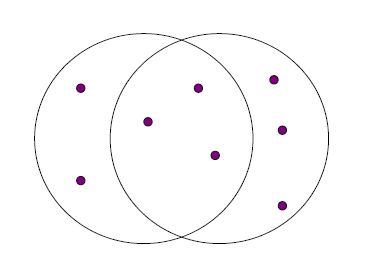


图4-1 Jaccard相似度为3/8的两个集合

#### 余弦距离

在有纬度的空间下，余弦距离才有意义，这些空间包括欧式空间及离散欧式空间，后者是的坐标是整数值或者布尔值表示的空间。在上述空间中的点是可以有方向的，那么两个点的距离就是点代表的向量的夹角，夹角的范围在。

比如向量分别是，那么这两个向量的余弦距离就是

#### 编辑距离

假设有两个字符串，那么这两个字符串的编辑距离就是 转换成 所需要的单字符插入及删除操作的最小操作数。

#### 海明距离

给定一个向量空间，海明距离定义为两个向量中不同分量的个数。

比如 的海明距离就是3，因为 和 的第2、4、5位元素不同。

## 文档相似度查询

假设我们求的是文档的相似度，给定的每条数据都是一些字符串，我们可以采用Jaccard距离来求文档的相似度，那么一种典型的算法流程如下图所示：

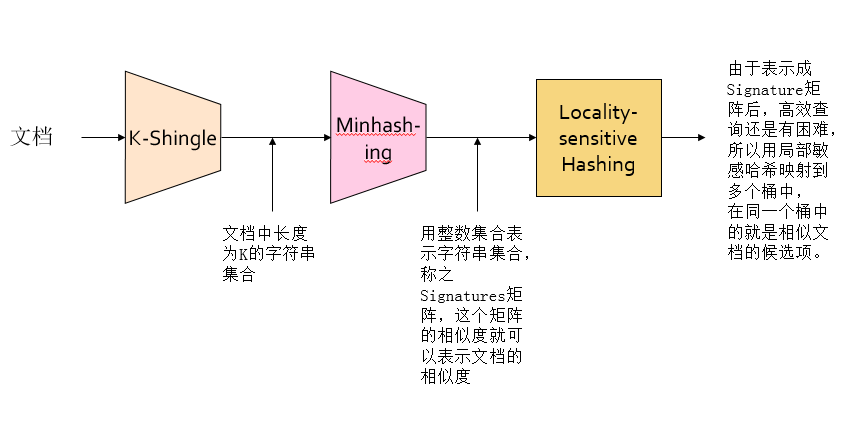


图5-1 文档相似度查询流程

#### K-shingle

为了识别相似的文档，需要将文档表示成一个集合，其中最有效的方法是构建文档的短字符集合。那么一篇文档就会有很多公共的短字符集合，这样改变句子或者单词的顺序对文档的相似性就不会产生大的影响。

这里文档的k-shingle定义为其中任意长度为k的子串。

假设文档 为字符串，那么组成的集合就是,注意这里的 在文档中出现了两次，但是在集合中只能算一次。

关于 中k的选择经验值是8，9，10。

#### Min-hashing(最小哈希)

通过K-shingle把文档表示成短字符集合后，数据量反而比原文档大很多，一般是4倍左右，那么就需要替换成规模更小的签名(Signature)矩阵，称之为特征矩阵，用这样特征矩阵来表示K-shingle集合，其实也是个降纬的过程。

这里假设有四篇文档 ，通过K-shingle提取出四个集合 ，那么全集就是，如果这四个集合的元素在全集中有则表示成1，如果没有表示成0，那么这四个集合可以表示成集合 如下图所示：

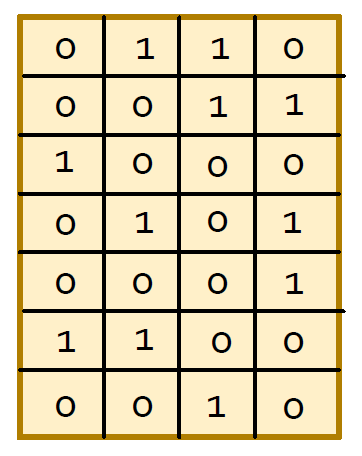


图5-2 文档集合的特征矩阵

这里给出一个定理：上述集合经过多次随机排列(多次哈希)之后，得到的最小哈希值的概率就是这四个集合的Jaccard相似度。

那么这里我们定义三个哈希函数 作用于集合的行号上，目的是为了随机排列得到新的矩阵 如下图所示：

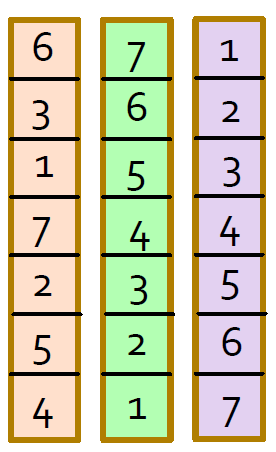


图5-3 三个哈希函数作用到数据行号后的结果

通过这个矩阵就可以求最小哈希值，这里最小哈希值组成的矩阵我们称之为最小哈希签名(minhashing signature)矩阵。

计算过程如下，假设在这个最小哈希签名矩阵中初始值是无穷大。



观察集合中的第一行，发现有值且对应的哈希值是1，7，6，那么最小哈希签名矩阵修改为



观察集合中的第二行，发现有值且对应的哈希值是2,6,3，那么最小哈希签名矩阵修改为



依次类推，最后得到的最小哈希签名矩阵如下



然后可以求得， 和 真实的Jaccard相似度也是0。当然在真实环境中，随着数据的增加和最小哈希函数的增多，这个估计值也会和真实值越接近。

## 局部敏感哈希

虽然用最小哈希将大文档压缩成小的签名矩阵，纬度大大降低，但是随着文档的增多，高效的查询相似文档还是有很大的困难。这里我们可以用LSH(Locality Sensitive Hashing)局部敏感哈希算法来把文档分类，相似的文档就哈希到同一个桶中，在一个桶中的文档就称之为候选对(candidate pair)。这里有几个评判相关的概念：

* 伪正例(false positive)(fp):哈希到同一个桶中的非相似文档。
* 伪反例(false negative)(fn):没有映射到相同的桶中的真正相似的文档。
* 正例(true positive)(tp):哈希到同一个桶中的相似文档。
* 反例(true positive)(tp):没有映射到相同的桶中的非相似文档。

我们希望伪正例和伪反例都越少越好。正例和反例越多越好。

通过上述概念可以定义召回率(recall)和准确率(precisioin)。





我们希望Precision和Recall都是越高越好，但是事实上这两者有时候是矛盾的，所以需要更综合的指标，一般采用， 的计算公式如下：



当 时就是



当F较高时，效果比较理想，所以为了提高F的值，我们希望对目标项进行多次哈希，只要有一次能够哈希到同一个桶中，那么都是候选项。假设其中一个比较有效的方法是将签名矩阵划分成b个行条(band)，每个行条由r行组成。对每个行条存在一个哈希函数能够将行条中每r个整数组成的列向量(行条中的每一列)映射到某个大数目范围桶中，如下图所示。

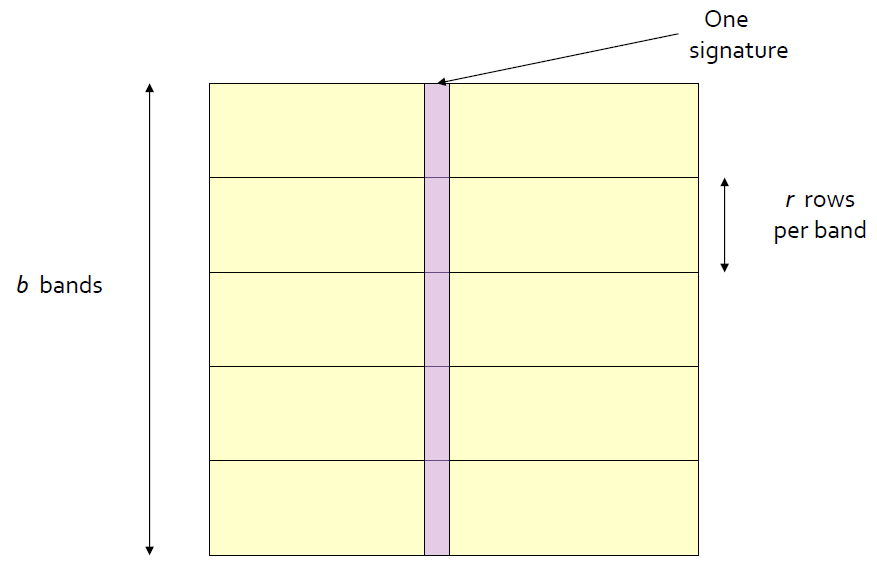


图6-1 签名矩阵转为b行条，每个行条有r行

假设某对具体文档之间的相似度为s，根据前面所述，那么这对具体文档对应的签名矩阵相等的概率也是s，具体的计算过程如下：

1. 在某个具体行条中所有行的的两个签名相等的概率是。
2. 在某个具体行条中中至少有一对签名不相等的概率是。
3. 在任何行条中的任意一行的签名对都不相等的概率为。
4. 签名至少在一个行条中全部相等的概率，也就是成为候选对的概率为。

这个概率函数的图像大致如下：

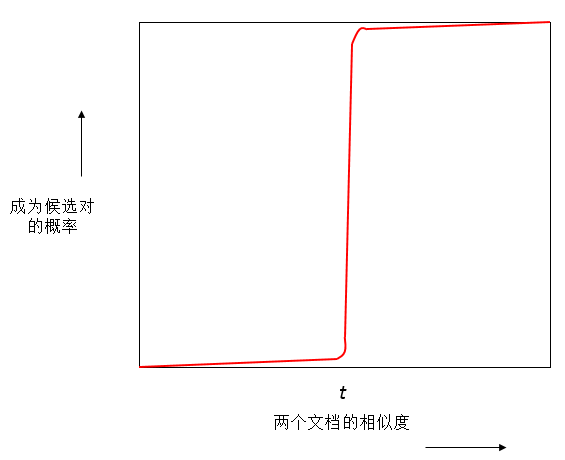


图6-2 文档Jaccard相似度

这里 ，我们就是要找到这个合适的阈值，使得能够足够大。

这里给出一个例子，假设，也就是说签名个数为100，分成20行条，每个行条包含5行。那么



比如两个文档的相似度在，，也就是说任意行条中，这两个文档的是候选对的概率才，但是，有这么大的概率成为候选对。

这就是局部敏感哈希采用行条也就是多次哈希的好处。

## 参考文献

[1]Rajaraman, A.; Ullman, J. (2010). "Mining of Massive Datasets, Ch. 3."

[2] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. "Hashing for similarity search: A survey." CoRR, abs/1408.2927, 2014

[3] Loic Pauleve, Herve Jegou, Laurent Amsaleg. "Locality sensitive hashing: a comparison of hash function types and querying mechanisms." Pattern Recognition Letters, 2010

[4] Wei Dong, Zhe Wang, William Josephson, Moses Charikar, and Kai Li. "Modeling LSH for performance tuning" CIKM, 2008