Locality Sensitive Hashing

Based on Min-Hashing

Yugang Yang　Hunan University

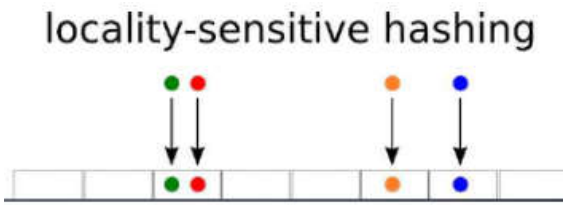
Abstract

应用背景：

在很多应用领域中，我们面对和需要处理的数据往往是海量并且具有很高的维度，怎样快速地从海量的高维数据集合中找到与某个数据最相似（距离最近）的一个数据或多个数据成为了一个难点和问题。如果是低维的小数据集，我们通过线性查找（Linear Search）就可以容易解决，但如果是**对一个海量的高维数据集采用线性查找匹配的话，会非常耗时**（比如我最近在研究的 content-based image retrieval 技术，就会遇到海量高维数据线性查找太慢的问题）,因此，为了解决该问题，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。

基本思想：

将原始数据空间中的两个相邻数据点通过相同的映射或投影变换（projection）后，这两个数据点在新的数据空间中仍然相邻的概率很大，而不相邻的数据点被映射到同一个桶的概率很小。经过它们的哈希映射变换后，原始空间中相邻的数据落入相同的桶内的话，那么我们在该数据集合中进行近邻查找就变得容易了，我们只需要将查询数据进行哈希映射得到其桶号，然后取出该桶号对应桶内的所有数据，再进行线性匹配即可查找到与查询数据相邻的数据。



Locality Sensitive Hashing

Definition

一个哈希函数族满足如下条件时，被称为是 $(\mathrm{R}, cR, P_1, P_2)$ -sensitive，对于任意两个点（高维特征向量） $\mathrm{p}, q \in R^d$ ，

$$\text{如果}\ \|p-q\| \le R\ \ \text{那么}\ \ \Pr_H[h(q)=h(p)] \ge P_1$$

$$\text{如果}\ \|p-q\| \ge cR\ \ \text{那么}\ \ \Pr_H[h(q)=h(p)] \le P_2$$

$$s.t.\ \ c>1,\ \ P_1>P_2$$

通俗解释就是，两个特征向量 p 和 q 的向量距离（L1 或 L2 或其它）小于等于某个值（R），那么经过哈希函数 h()，p 和 q 被映射到同一个桶的概率应该大于等于 P_1。如果 p 和 q 的向量距离大于等于某个值(cR），那么 p 和 q 经过哈希函数 h()被映射到同一个桶的概率应该小于等于 P_2。H 是函数族，代表 H 中的所有哈希函数 h（）都是 (R, cR, P_1, P_2)-sensitive 的哈希函数。

LSH 不像树形结构方法可以得到精确的经过，LSH 得到的是一个近似的结果，因为很多领域中并不需要非常高的精确度。即使是近似解，有时候这个近似程度几乎和精确解一致。所以 LSH 的主要思想是，高维空间的两点若距离很近，那么设计一种哈希函数对这两点进行哈希值计算，使得他们哈希值有很大的概率是一样的。同时若两点之间的距离较远，他们哈希值相同的概率会很小。

LSH 的原理核心有两个：

1. 两个高维向量的相似性度量方法 （比如我们之前接触过的 min-hash 求得签名矩阵并得到最后的相似性矩阵）

2. (R, cR, P_1, P_2)-sensitive 哈希函数的选择。

LSH 的哈希函数的选择取决于其选择的相似性度量方法，当然并不是所有向量相似性度量的方法都能找到相应的 LSH 函数，比如 LSH 最初提出的时候基于欧式距离的度量方法就没有找到合适的 LSH 函数。

看到这里，我们知道，LSH 需要四个参数 R,c, P_1, P_2

以上讲的 LSH 都还只是一个思想，不同的相似性度量方法在 LSH 思想下，其真正且具体的 LSH 设计也不同，我们主要看看在两种最常用的相似度下，两种不同的 LSH 设计是什么样的！！

1. 基于 Jaccard 系数度量的 min-hash 的 LSH 设计

关于 Jaccard 系数度量的 min-hash 可以参考我写的文档，这里稍微提一下！

为了能实现前面 LSH 定义中的 2 个条件的要求，我们通过多次置换，求取向量，构建了一组 hash 函数。也就是最终得到了一个 signature matrix 如下图所示!(你没看过我的文档你肯定不知道我说啥~~)

Signature matrix *M*

2	1	2	1
2	1	4	1
1	2	1	2

图中每一行代表一个哈希函数，每一列代表一个文档，这个矩阵可以得到 Jaccard 相似度矩阵，如下图所示：

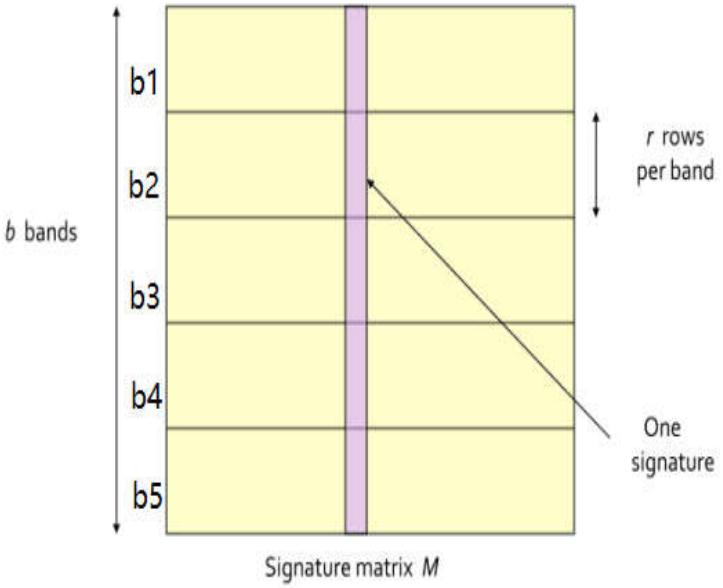
Similarities:

	1-3	2-4	1-2	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0

以上是关于最小哈希涉及的东西，现在来开始阐述和 LSH 有关的东西

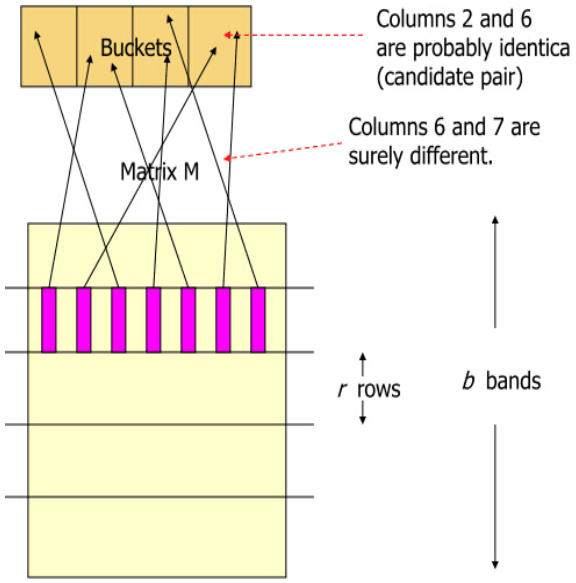
构造 LSH 函数族

将 signature matrix 水平分割成一些区块（记为 band），每个 band 包含了 signature matrix 中的 r 行。需要注意的是，同一列的每个 band 都是属于同一个文档的。如图 1 所示：



就这一列浅紫色的向量，可以是很多东西，比如一个样本的特征之类的

图 1



接下来听好了!!

我们把签名矩阵划分成 b 个行条，每个行条由 r 行组成。对于每个行条，存在一个哈希函数能够将行条中的 每 r 个整数组成的列向量 (行条中的每一列，比如左图中某一条粉色的竖棒) 映射到某个桶中。可以对所有行条使用相同的哈希函数，但是对于每个行条我们都使用一个独立的桶数组，如左图所示，某一个行条都有各自的桶数组。即便是不同行条中相同的列向量，也不会被哈希到同一个桶中。这样，只要两个集合中存在某个行条中有落在相同桶的两列，这两个集合就被认为可能相似度比较高，作为后续计算的候选对；而那些在所有行条中都不落在同一个桶中的两列，就会被认为相似度不会很高，而直接被忽略。

综上，我们来总结一下：

首先，我们用来映射紫色棒棒的函数是随意的哈希函数（抗碰撞和安全性好就行）。

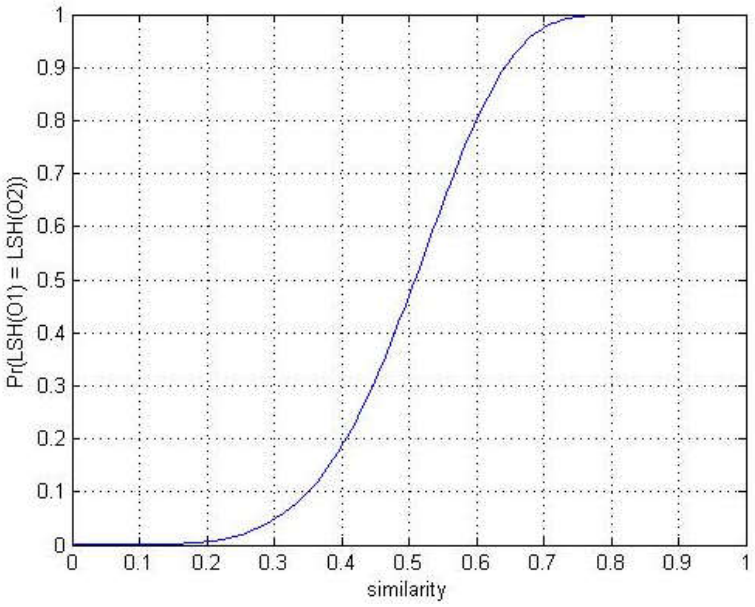
1. 对于处于同一个 band 的两个文档（上图中的两条粉棒棒，也即两个列向量），这两个列向量每一对元素一一相同的概率是 s^r，其中 s 是这两个文档的 Jaccard similarity，就是说我们拿任意的哈希函数来映射一个 band 中的紫色棒棒，这两个紫色棒棒会被映射到同一个桶的概率是 s^r。

> 第一次看到这个 s^r 概率，思考良久。首先，两个列向量之间任意一对元素相等的概率为什么是 s。那是因为这里我们拿了经验概率来替换期望概率。因为我们现在有的样本只有两个列向量，比如[1,2,1,3,4]和[1,2,3,1,4] 这两个列向量有 3 对位置对应的值相同，向量长度为 5，所以我们就拿这个经验概率 0.6 当作任意一处位置，这两列向量对应这一处位置其值相等的概率是 0.6 了，

2. 也就是说，这在同一个 band 中的两个列向量（图中的小粉棒）不相同的概率是 $1-s^r$
3. 这两个文档一共有 b 个 band，这 b 个 band 中都不相同的概率是 $(1-s^r)^b$，
4. 所以说，这 b 个 band 至少有一个相同的概率是 $1-(1-s^r)^b$

到这里突然大彻大悟，原来实现下图中曲线的核心并不是哈希函数的选择，而是他的这个策略！即把两个长的列向量分为 b 个 band，然后用哈希寻找相同输入输出的长度为 r 的列向量的这个套路，实现了 $1-(1-s^r)^b$ 。

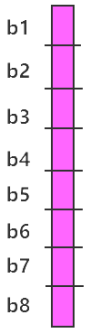
这样一来，实际上可以通过控制参数 r,b 的值来控制两个文档被映射到同一个哈希桶的概率。而且效果非常好。比如，令 b=20, r=5，我们来画一个概率图：



这个曲线说白了就是 $1-(1-s^r)^b$ 曲线。

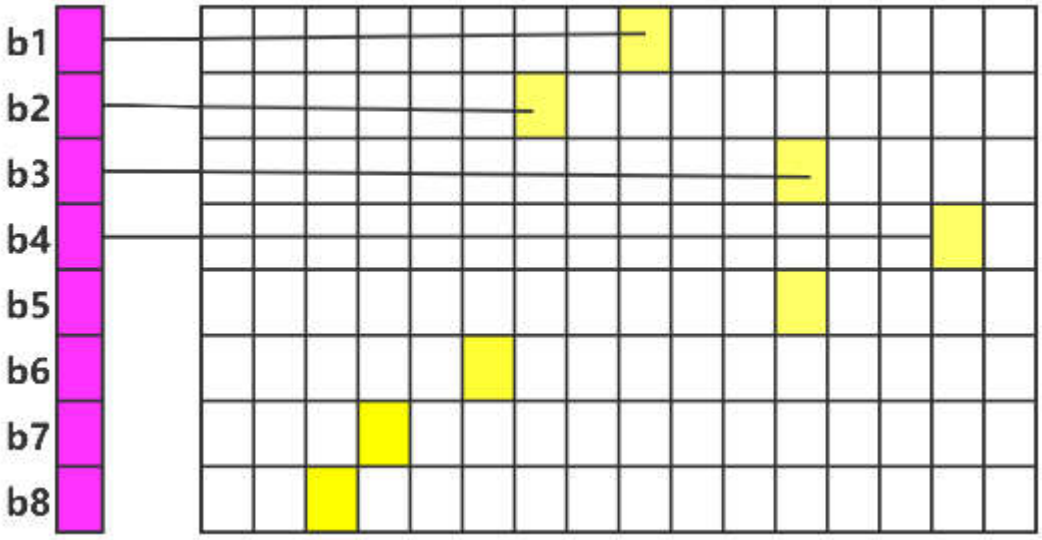
横坐标代表文档 p 和文档 q 的 Jaccard 相似度，如果两个文档的 Jaccard 相似度越高，就越有可能被映射到同一个 hash 桶内，反之就越不可能被映射到一个 hash 桶内，处于中间态的 Jaccard 系数比较短。

现在我们离线索引建立好了，来试试查询？



比如我有一个 query 文档，然后我通过 min-hash 的方法变成一个 query signature 列向量。然后送进 LSH 方法里，LSH 方法会把我这个 query signature 切成 b 个 band，然后拿它哈希函数映射我这 b 个 band，如左图所示！b=8

现在我们拿哈希函数分别对这 b 个 band 去映射（注意，我推测你当初离线建立索引时拿的什么哈希函数去映射，这里就同样的。比如你是一个 band 一个哈希函数？还是所有 band 同一个哈希函数？哈希函数的函数式子是什么？ 都要和你离线建立索引时要完全相同）



对每个 band 进行映射。每个 band 在其对应的桶空间中都会映射到对应的桶（黄色的方块），把黄色方块对应的特征取出来，与索引一一进行相似性度量！！！！

大功告成!!!

Reference

http://blog.rexking6.top/2018/10/09/%E5%B1%80%E9%83%A8%E6%95%8F%E6%84%9F%E5%93%88%E5%B8%8C-Locality-Sensitive-Hashing-LSH/ 写的这么好你不看，你良心呢？

https://blog.csdn.net/liujan511536/article/details/47729721

https://blog.csdn.net/guoziqing506/article/details/53019049 这个写的也好！

https://blog.csdn.net/yc461515457/article/details/48845775

https://www.cnblogs.com/wangguchangqing/p/9796226.html 有实例，而且写的也比较一阵见血

https://blog.csdn.net/baidu_21807307/article/details/51794373

https://www.cnblogs.com/fengfenggirl/p/lsh.html