局部敏感哈希 Locality-sensitive Hashing

摘自网络博客-----By 李欣

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。也就是说，如果我们对原始数据进行一些 hash 映射后，我们希望原先相邻的两个数据能够被 hash 到相同的桶内，具有相同的桶号。对原始数据集合中所有的数据都进行 hash 映射后，我们就得到了一个 hash table，这些原始数据集被分散到了 hash table 的桶内，每个桶会落入一些原始数据，属于同一个桶内的数据就有很大可能是相邻的，当然也存在不相邻的数据被 hash 到了同一个桶内。因此，如果我们能够找到这样一些 hash functions，使得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。换句话说，我们通过 hash function 映射变换操作，将原始数据集合分成了多个子集合，而每个子集合中的数据间是相邻的且该子集合中的元素个数较小，因此将一个在超大集合内查找相邻元素的问题转化为了在一个很小的集合内查找相邻元素的问题，显然计算量下降了很多。

那具有怎样特点的 hash functions 才能够使得原本相邻的两个数据点经过 hash 变换后会落入相同的桶内？这些 hash function 需要满足以下两个条件：

1）如果 $d(x,y) \le d1$， 则 $h(x) = h(y)$的概率至少为 p1；

2）如果 $d(x,y) \ge d2$， 则 $h(x) = h(y)$的概率至多为 p2；

其中 d(x,y)表示 x 和 y 之间的距离，$d1 < d2$， h(x)和 h(y)分别表示对 x 和 y 进行 hash 变换。

满足以上两个条件的 hash functions 称为(d1,d2,p1,p2)-sensitive。而通过一个或多个(d1,d2,p1,p2)-sensitive 的 hash function 对原始数据集合进行 hashing 生成一个或多个 hash table 的过程称为 Locality-sensitive Hashing。

使用 LSH 进行对海量数据建立索引（Hash table）并通过索引来进行近似最近邻查找的过程如下：

1. 离线建立索引

（1）选取满足(d1,d2,p1,p2)-sensitive 的 LSH hash functions；

（2）根据对查找结果的准确率（即相邻的数据被查找到的概率）确定 hash table 的个数 L，每个 table 内的 hash functions 的个数 K，以及跟 LSH hash function 自身有关的参数；
（3）将所有数据经过 LSH hash function 哈希到相应的桶内，构成了一个或多个 hash table；

2. 在线查找
（1）将查询数据经过 LSH hash function 哈希得到相应的桶号；
（2）将桶号中对应的数据取出；(为了保证查找速度，通常只需要取出前 2L 个数据即可)；
（3）计算查询数据与这 2L 个数据之间的相似度或距离，返回最近邻的数据；

LSH 在线查找时间由两个部分组成： （1）通过 LSH hash functions 计算 hash 值（桶号）的时间；(2）将查询数据与桶内的数据进行比较计算的时间。因此，LSH 的查找时间至少是一个 sublinear 时间。为什么是“至少”？因为我们可以通过对桶内的属于建立索引来加快匹配速度，这时第（2）部分的耗时就从 O(N)变成了 O(logN)或 O(1)（取决于采用的索引方法)。

LSH 为我们提供了一种在海量的高维数据集中查找与查询数据点（query data point）近似最相邻的某个或某些数据点。需要注意的是, LSH 并不能保证一定能够查找到与 query data point 最相邻的数据，而是减少需要匹配的数据点个数的同时保证查找到最近邻的数据点的概率很大。

二、LSH 的应用

LSH 的应用场景很多，凡是需要进行大量数据之间的相似度（或距离）计算的地方都可以使用 LSH 来加快查找匹配速度，下面列举一些应用：

（1）查找网络上的重复网页

互联网上由于各式各样的原因（例如转载、抄袭等）会存在很多重复的网页，因此为了提高搜索引擎的检索质量或避免重复建立索引，需要查找出重复的网页，以便进行一些处理。其大致的过程如下：将互联网的文档用一个集合或词袋向量来表征，然后通过一些 hash 运算来判断两篇文档之间的相似度，常用的有 minhash+LSH、simhash。

（2）查找相似新闻网页或文章

与查找重复网页类似，可以通过 hash 的方法来判断两篇新闻网页或文章是否相似，只不过在表达新闻网页或文章时利用了它们的特点来建立表征该文档的集合。

（3）图像检索

在图像检索领域，每张图片可以由一个或多个特征向量来表达，为了检索出与查询图片相似的图片集合，我们可以对图片数据库中的所有特征向量建立 LSH 索引，然后通过查找 LSH 索引来加快检索速度。目前图像检索技术在最近几年得到了较大的发展，有兴趣的读者可以查看基于内容的图像检索引擎的相关介绍。

（4）音乐检索

对于一段音乐或音频信息，我们提取其音频指纹（Audio Fingerprint）来表征该音频片段，采用音频指纹的好处在于其能够保持对音频发生的一些改变的鲁棒性，例如压缩，不同的歌手录制的同一条歌曲等。为了快速检索到与查询音频或歌曲相似的歌曲，我们可以对数据库中的所有歌曲的音频指纹建立 LSH 索引，然后通过该索引来加快检索速度。

（5）指纹匹配

一个手指指纹通常由一些细节来表征，通过对比较两个手指指纹的细节的相似度就可以确定两个指纹是否相同或相似。类似于图片和音乐检索，我们可以对这些细节特征建立 LSH 索引，加快指纹的匹配速度。

三、LSH family

我们在第一节介绍了 LSH 的原理和 LSH hash function 需要满足的条件，回顾一下：

满足以下两个条件的 hash functions 称为(d1,d2,p1,p2)-sensitive：

1）如果 $d(x,y) \leq d1$， 则 $h(x) = h(y)$的概率至少为 p1；

2）如果 $d(x,y) \geq d2$， 则 $h(x) = h(y)$的概率至多为 p2；

d(x,y)是 x 和 y 之间的一个距离度量（distance measure），需要说明的是，并不是所有的距离度量都能够找到满足 locality-sensitive 的 hash functions。

下面我们介绍一些满足不同距离度量方式下的 locality-sensitive 的 hash functions：

1. Jaccard distance

Jaccard distance：（1 - Jaccard similarity），而 Jaccard similarity = (A intersection B) / (A union B)，Jaccard similarity 通常用来判断两个集合的相似性。

Jaccard distance 对应的 LSH hash function 为：**minhash**，其是(d1,d2,1-d1,1-d2)-sensitive 的。

2. Hamming distance

Hamming distance：两个具有相同长度的向量中对应位置处值不同的次数。

Hamming distance 对应的 LSH hash function 为：H(V) = 向量 V 的第 i 位上的值，其是(d1,d2,1-d1/d,1-d2/d)-sensitive 的。

3. Cosine distance

Cosine distance：cos(theta) = A•B / |A||B| ，常用来判断两个向量之间的夹角，夹角越小，表示它们越相似。

Cosine distance 对应的 LSH hash function 为：H(V) = sign(V•R)，R 是一个随机向量。V•R 可以看做是将 V 向 R 上进行投影操作。其是(d1,d2,(180-d1)180,(180-d2)/180)-sensitive 的。

理解：利用随机的超平面（random hyperplane）将原始数据空间进行划分，每一个数据被投影后会落入超平面的某一侧，经过多个随机的超平面划分后，原始空间被划分为了很多cell，而位于每个 cell 内的数据被认为具有很大可能是相邻的（即原始数据之间的 cosine distance 很小）。

4. normal Euclidean distance

Euclidean distance 是衡量 D 维空间中两个点之间的距离的一种距离度量方式。

Euclidean distance 对应的 LSH hash function 为：H(V) = |V·R + b| / a，R 是一个随机向量，a 是桶宽，b 是一个在[0,a]之间均匀分布的随机变量。V·R 可以看做是将 V 向 R 上进行投影操作。其是(a/2,2a,1/2,1/3)-sensitive 的。

理解：将原始数据空间中的数据投影到一条随机的直线（random line）上，并且该直线由很多长度等于 a 的线段组成，每一个数据被投影后会落入该直线上的某一个线段上（对应的桶内），将所有数据都投影到直线上后，位于同一个线段内的数据将被认为具有很大可能是相邻的（即原始数据之间的 Euclidean distance 很小）。

四、增强 LSH（Amplifying LSH）

通过 LSH hash functions 我们能够得到一个或多个 hash table，每个桶内的数据之间是近邻的可能性很大。我们希望原本相邻的数据经过 LSH hash 后，都能够落入到相同的桶内，而不相邻的数据经过 LSH hash 后，都能够落入到不同的桶中。如果相邻的数据被投影到了不同的桶内，我们称为 false negtive；如果不相邻的数据被投影到了相同的桶内，我们称为 false positive。因此，我们在使用 LSH 中，我们希望能够尽量降低 false negtive rate 和 false positive rate。

通常，为了能够增强 LSH，即使得 false negtive rate 和/或 false positive rate 降低，我们有两个途径来实现：1）在一个 hash table 内使用更多的 LSH hash function；2）建立多个 hash table。

下面介绍一些常用的增强 LSH 的方法：

1. 使用多个独立的 hash table

每个 hash table 由 k 个 LSH hash function 创建，每次选用 k 个 LSH hash function（同属于一个 LSH function family）就得到了一个 hash table，重复多次，即可创建多个 hash table。多个 hash table 的好处在于能够降低 false positive rate。

2. AND 与操作

从同一个 LSH function family 中挑选出 k 个 LSH function，H(X) = H(Y)有且仅当这 k 个 Hi(X) = Hi(Y)都满足。也就是说只有当两个数据的这 k 个 hash 值都对应相同时，才会被投影到相同的桶内，只要有一个不满足就不会被投影到同一个桶内。

AND 与操作能够使得找到近邻数据的 p1 概率保持高概率的同时降低 p2 概率，即降低了 falsenegtiverate。

3. OR 或操作

从同一个 LSH function family 中挑选出 k 个 LSH function，H(X) = H(Y)有且仅当存在一个以上的 Hi(X) = Hi(Y)。也就是说只要两个数据的这 k 个 hash 值中有一对以上相同时，就会被投影到相同的桶内，只有当这 k 个 hash 值都不相同时才不被投影到同一个桶内。

OR 或操作能够使得找到近邻数据的 p1 概率变的更大（越接近 1）的同时保持 p2 概率较小，即降低了 false positive rate。

4. AND 和 OR 的级联

将与操作和或操作级联在一起，产生更多的 hahs table，这样的好处在于能够使得 p1 更接近 1，而 p2 更接近 0。

除了上面介绍的增强 LSH 的方法外，有时候我们希望将多个 LSH hash function 得到的 hash 值组合起来，在此基础上得到新的 hash 值，这样做的好处在于减少了存储 hash table 的空间。下面介绍一些常用方法：

1. 求模运算

new hash value = old hash value % N

2. 随机投影

假设通过 k 个 LSH hash function 得到了 k 个 hash 值：h1，h2...，hk。那么新的 hash 值采用如下公式求得：

new hash value = h1*r1 + h2*r2 + ... + hk*rk，其中 r1, r2, ..., rk 是一些随机数。

3. XOR 异或

假设通过 k 个 LSH hash function 得到了 k 个 hash 值：h1，h2...，hk。那么新的 hash 值采用如下公式求得：

new hash value = h1 XOR h2 XOR h3 ... XOR hk

五、相关参考资料

Website:

[1] http://people.csail.mit.edu/indyk/ （LSH 原作者）

[2] http://www.mit.edu/~andoni/LSH/ (E2LSH)

[3] https://blog.csdn.net/icvpr/article/details/12342159

Paper:

[1] Approximate nearest neighbor: towards removing the curse of dimensionality

[2] Similarity search in high dimensions via hashing

[3] Locality-sensitive hashing scheme based on p-stable distributions

[4] MultiProbe LSH Efficient Indexing for HighDimensional Similarity Search

[5] Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions

Tutorial:

[1] Locality-Sensitive Hashing for Finding Nearest Neighbors

[2] Approximate Proximity Problems in High Dimensions via Locality-Sensitive Hashing

[3] Similarity Search in High Dimensions

Book:

[1] Mining of Massive Datasets

[2] Nearest Neighbor Methods in Learning and Vision: Theory and Practice

Cdoe:

[1] http://sourceforge.net/projects/lshkit/?source=directory

[2]http://tarsos.0110.be/releases/TarsosLSH/TarsosLSH-0.5/TarsosLSH-0.5-Readme.html

[3] http://www.cse.ohio-state.edu/~kulis/klsh/klsh.htm

[4] http://code.google.com/p/likelike/

[5] https://github.com/yahoo/Optimal-LSH

[6] OpenCV LSH（分别位于 legacy module 和 flann module 中）