Locality Sensitive Hashing

Based on P-table Distribution

Yugang Yang　Hunan University

Abstract

建议研读我写的 Locality sensitive hashing based on Min-hashing 文档，不然我接下来的内容你都会感到很多不懂得。

对应海明距离的 LSH 称为位采样算法（bit sampling），该算法是比较得到的哈希值的海明距离，但是一般距离都是用欧式距离进行度量的，将欧式距离映射到海明空间再比较其的海明距离比较麻烦。于是，研究者提出了基于 p-稳定分布的位置敏感哈希算法，可以直接处理欧式距离，并解决(R,c)-近邻问题。

基于 p-table 分布　的　LSH 设计

最开始的时候，我们已经说过，不同的相似度判别方法，对应着不同的 LSH，那对于最常见的 Lp 范数下的欧几里得空间，应该用怎样的 LSH 呢？这就要介绍 P-stable hash 了。

在讲解 p-stable hash 之前，先简单介绍一下 p 稳定分布的概念。

Definition of p-stable distribution

一定要认真理解，这个 p-stable 很牛逼！！我了解了之后真的是露出了宠溺的微笑！！

reference: http://www.cppblog.com/humanchao/archive/2018/02/24/215521.aspx

如果一个分布 D 是 p-stable 分布，如果对于任意个实数 $v_1, v_2, v_3 ..., v_n$ 和符合 D 分布的 n 个独立同分布随机变量 $X_1, X_2, X_3 ..., X_n$，都存在一个 $p \geq 0$，使得 $\sum_{i}^{n} v_i X_i$（点乘，得到的结果是一个整数）和 $(\sum_{i}^{n} |v_i|^p)^{\frac{1}{p}} X$（先求 P 范数，X 是服从 D 分布的一个随机变量，最后整个式子求得的结果也是一个整数）有相同的分布，则称 D 为一个 p-stable 分布。比如 p=1 是柯西分布，p=2 是高斯分布。

这里有相同的分布的意思是：两个数列同分布，意味着呈线性关系的两个数列，可以理解为同比例缩放。

用通俗的语言进行解析上述定义就是：

取满足正态分布的一个数列（$v_1, v_2, v_3 \ldots, v_n$），与某个特征（$X_1, X_2, X_3 \ldots, X_n$）向量做向量点乘，得到一个数值，这个数值与这个特征向量本身的 2 阶范数，有同分布的性质。

举例：

假设有两个图像特征 F_1, F_2（均为向量），用这两个特征分别与同一个正态分布的数列做向量点乘，所得到的 2 个数值在一维上的距离与 F_1, F_2 在多维上的欧氏距离是同分布的。意思就是说，你这个 F_1, F_2 的高维距离可以通过这种乘以服从 p-table 分布的列向量转换成一维的距离，而这一维的距离是可以象征这个高维的距离（概率意义上的相近）！

用图像库中 N 个图像的 N 个特征分别与同一个正态分布的数列做向量点乘，得到的 N 个特征在一维上的点，我们用在一维上的点之间的距离度量多维空间的距离，当然这种相近是概率意义下的相近。

所以我们得出的结论是：**高维空间中的两个点的距离：$\| F_1 - F_2 \|_p$ 近到一定程度时，应该被 hash 成同一 hash 值，而向量点积性质正好保持了这种局部敏感性，因此可以用点积来设计 hash 函数族。**

局部敏感哈希：

为了简化计算，把一维上的线划分成段（段长为 r）。

为了提高算法的稳定性，向量点乘后增加一个随机的噪音 b。

于是得到了一个新的哈希函数：

$$h_{v,b}(F) = \frac{v \bullet F + b}{r}$$

其中，v 是服从 p-stable 分布的随机数列，F 是特征向量。v 和 F 进行点乘操作。b 是加入的随机噪声，r 是我们分的段的段长。 **这个哈希函数的牛逼之处在于，他的确将特征向量降维成一个小傻比，但是这个小傻比还死死的守护着自己的欧几里德距离性质！！！！**

像这样一类 hash 函数我们称之为局部敏感 hash 函数，下面给出局部敏感哈希函数的定义：

将这样的一族 hash 函数 $H = \{h : S \to U\}$ 称之为是 (d_1, d_2, p_1, p_2) 敏感的，如果对于任意 H 中的函数 h，满足以下 2 个条件：

——如果 $d(F_1, F_2) < d_1$，那么 $P_r[h(F_1) = h(F_2)] \geq p_1$

——如果 $d(F_1, F_2) > d_2$，那么 $P_r[h(F_1) = h(F_2)] \leq p_2$

其中，$F_1, F_2 \in S$，表示两个具有多维属性的数据对象，$d(F_1, F_2)$ 为 2 个对象的相异程度，

也就是相似度。

如果两个特征，在高维空间上，距离足够近（$< d_1$），那这两个特征被映射为同一值的概率越大($\geq p_1$)

概率相似度度量与调节（与构造与或构造）

我觉得这段话写的很好，很精髓!!!!!

既然是在概率意义下相似性度量，必然会存在着相近样本被 hash 到不同的 hash 值情况，同时也必然会存在不相近的样本被 hash 到相同的 hash 值情况，前一种称为伪反例，向一种称为伪正例。

伪正例（不相似的映射在了一起）通过与构造解决，即通过多个 hash 函数，计算同一个特征的多个 hash 值，只有 **L** 个 hash 函数均相同时，才认为特征相似。这有效避免了不相似的特征被判定为相似特征的情况。

伪反例（相似的没有映射在一起）通过或构造解决，即通过多个 hash 函数，计算同一个特征的多个 hash 值，只有 **k** 个 hash 函数中出现一对 hash 值相同的时候，即认为特征相似。这有效避免了相似的特征被判定为不相似特征的情况。

结合与构造与或构造 2 种方案，可以生成 $L \times k$ 个函数，每 L 个 hash 函数带表一组与构造，每 k 组 hash 函数族代表一组或构造，当满足一个或构造后特征判定为相似。设一组特征 hash 的相似概率为 s，则通过 hash 函数与/或构造后的相似概率为：$1-(1-s^r)^b$

构建 hash table 实际操作：

如果把一个函数族对向量的一组 hash 值（$h_1(F), h_2(F), \ldots, h_k(F)$）作为 hash bucket 的标识，有两个缺点：1. 空间复杂度大；2. 不易查找。为了解决这个问题，我们采用如下方法：

先设计两个 hash 函数：H_1, H_2,其中：

$H_1 : Z^k \rightarrow \{0,1,2\ldots, size-1\}$ 简单说就是把一个 k 个数 组成的整数向量映射到 hash table 的某一个位上，其中 size 是 hash table 的长度。

$H_2 : Z^k \rightarrow \{0,1,2\ldots, C\}, C = 2^{32} - 5$ 是一个大素数

者两个函数的具体算法如下，其中 r_i ， r_i’ 是两个随机整数

$$H_1(x_1, \ldots x_k) = ((\sum_{i=1}^{k} r_i x_i) \bmod C) \bmod size$$

$$H_1(x_1, \ldots x_k) = (\sum_{i=1}^{k} r_i' x_i) \bmod C$$

这 H_1, H_2 的计算中，输入是一个特征向量输出是一个整数（指纹标签）。

我们把 H_2 计算的结果成为一个数据向量的“指纹”，这也好理解，它就是由数据向量的 k 个 hash 值计算得到的。而 H1 相当于是数据向量的指纹在 hash table 中的索引，这个算法跟基本的散列表算法是一个思路。

通过这两个新建的函数，我们可以将 hash table 的构建步骤作以下详细说明：

1.从设计好的 LSH 函数族中，随机选取 L 组 hash 函数组，每组由 k 个 hash 函数构成，记为

$\{g_1(\bullet), g_2(\bullet), \ldots, g_L(\bullet)\}$，其中：$g_i(\bullet) = (h_1(\bullet), h_2(\bullet), \ldots, h_k(\bullet))$

如图 1 所示：

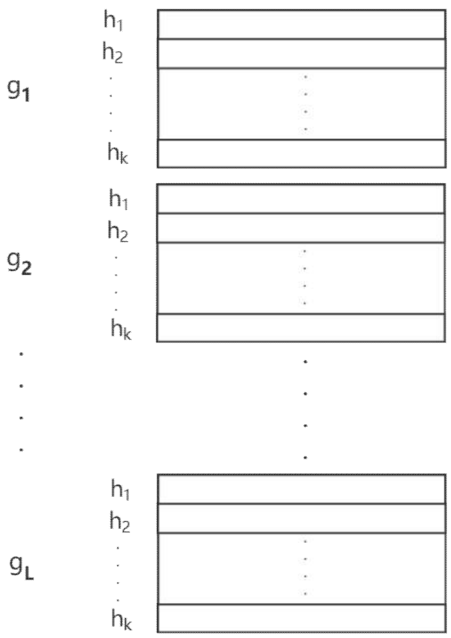

图 1

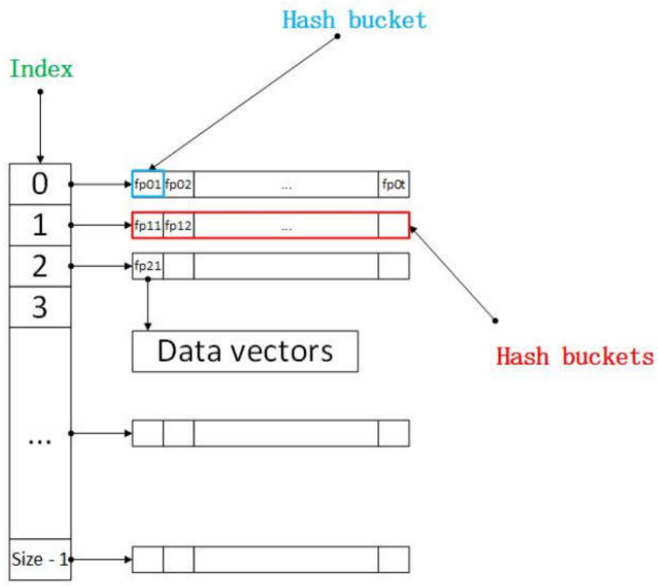

图 2

2.每个数据向量经过 $g_i(\bullet)$ 被映射成一个整型向量，记为 $(x_1, x_2 \ldots, x_k)$

3.将 2 步生成的 $(x_1, x_2 \ldots, x_k)$ 通过 H_1, H_2 计算得到两个数值：index, fp，前者是 hash table 的索引，后者是数据向量对应的指纹。这里，为了方便描述这种 hash table 的结构，我将我们用的 hash table 的结构画出，如图 2 所示。

4. 如果我们有两个特征向量，这两个特征向量在一个 g_i 中被 hash k 次。如果这两个特征向量在 k 次 hash 中，全都落入同一个 bucket 中，就说明这俩特征向量在这个 g_i 中是相似的

如图 3 所示

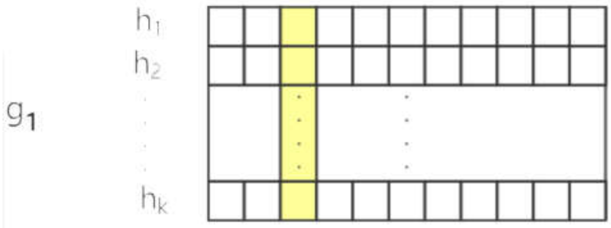

图 3

5. 如果在这 L 次中，只需要任意一个 g_i 满足第 4 步，就说明，这两个向量是相似（添加候选)，如果都在这 L 次中，没有任意一个 g_i 满足第四步，说明这俩向量相似

这样一来，通过 k 和 L 的哈希嵌套，就能得出和 min-hash 下的 LSH 方法一样的概率，即：

$$1-(1-P^k)^L$$

其中，$P=\Pr(h_{v,b}(F_1)=h_{v,b}(F_2))$。看完了 min-hash 下的 LSH 自然就知道这个 P 怎么计算，这个 P 就是经验概率！即两列向量对应位置值相同的个数除以列向量长度！

Reference

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

reference: http://www.cppblog.com/humanchao/archive/2018/02/24/215521.aspx

https://statusrank.xyz/articles/84288273.html　好啊！