Hashing for Similarity Search: A Survey

摘要：文章主要从两方面对 hashing 进行了介绍。1.局部敏感哈希：不考虑数据的分布状况，选定不同距离度量下的特定满足 LSH 条件的 hashing family 函数，并抽样产生参数。2：学习哈希，通过考虑数据的分布情况，用学习的方法（定义相应的目标函数）获得相应的 hashing 函数的参数。其他还有关于基于 hash code 进行相似度比较的一些常见搜索方法。

1.介绍

数据集大时，直接进行精确的最近邻检索是很消耗时间的。因此，近似的最近邻搜索是一种可选取得方法。典型的 ANN 的方法是 hashing。总的来说，哈希就是一种将数据转化到低维度，形成短比特组成 code 的方法。哈希的应用主要有两种方式：

用哈希表索引数据（将相似的数据以更大的概率编码到相同的 code）--局部敏感哈希。主要的研究放在了设计满足 LSH 条件的 hash function，以及针对相应的 hash table，设计有效的搜索策略。

近似距离估计（用短的 codes 进行全局的距离计算是非常高效的）。主要研究放在了设计计算短 code 的方法上，以及设计距离测量的方法上。

2.回顾

2.1 最近邻搜索

2.1.1 精确最近邻搜索

第一种方法是 KNN。涉及的距离度量有 Lp 范数，cosine similarity 等。

还有一种方法是 RNN。$\{\mathbf{x}|\,\mathrm{dist}(\mathbf{q},\mathbf{x}) \leqslant R, \mathbf{x} \in \mathcal{X}\}$。

2.1.2 近似最近邻搜索

$(1+\epsilon)$最近邻搜索：找到一个 item x : dist(q,x)<=$(1+\epsilon)$dist(q,x)*，其中，x*是真正的最近邻点。

cR 最近邻搜索：$\mathrm{dist}(\mathbf{q},\mathbf{x}) \leqslant cR$。

2.1.3 随机最近邻搜索 （ps.with probability $1-\delta$ ）

随机 *cR* 最近邻搜索以及随机 *R* 最近邻搜索。

2.2 哈希方法

将数据映射到短的，紧致的哈希码。例如，y=h(x)。其中，y 是哈希码，h(-)是哈希函数。通常，我们会使用对个哈希函数，来获得多个 code 来组成哈希码。**y**=**h**(x)，**y**=[y1 y2 … ym]'=[h1(x) h2(x) … hm(x)]'。使用哈希码进行最近邻搜索有两种方法：1 是使用哈希表查询；2 是快速的距离近似。

2.2.1 哈希表查询 （ps. Hash tables, buckets, size-L, length-K）

哈希表是由多个桶（buckets）组成的数据结构，每一个桶都通过哈希码进行索引。并且每一个 item x 都对应一个 h(x)的桶。特别的，哈希算法的哈希表构建的目标是实现邻近 items 碰撞的概率最大化。即希望，给定一个查询 q，相应的在 h(q)桶中的 items 都是 q 的实际的邻近 items。

为了改善哈希表的**召回率**，通常可以增加哈希表的数目。这样，假设共有 L 个 hash tables，我们就返回 **h**1(q),…**h**L(q)的所有 tables 的对应桶中的 items。

同时为了保证**精度**，我们希望每一个 table 中的 hash code 长度 K 都足够长。

2.2.2 快速距离近似 （ps. Exhaustive search）

另一种 hash 码处理的方式是直接进行全局的查询。即先快速的计算查询 item 和数据集样本的 hash code 的距离，获得候选的 items。然后紧接着在候选点中进行基于原始特征的 reranking，获得 KNN 或者 RNN。

这边利用了 hash code 的两个点：1.哈希码因为短，所以能很有效的进行全局计算。2.哈希码能进行有效的磁盘读写，i/o 消耗很少。

2.3 论文的组织

Section.3 介绍 LSH 的特性，以及一些在不同 distance 衡量方法下的 family instance (hashing function)。

Section 4 介绍利用 LSH codes 进行搜索的方法；Section 5.6.7 基于学习的哈希算法。

3.局部敏感哈希 （LSH）（ps. 定义，以及在不同距离度量下的 LSH 函数例子）

LSH 族$\mathcal{H}$：一族能够将在输入空间相似的 items 以更高概率映射到相同哈希码的哈希函数。

LSH 的理论研究主要在三个方面：

1. 发明针对不同类型的 distance/similarity 度量的不同的 LSH 函数族；
2. 探索不同 LSH 框架的不同性质的理论边界；
3. 改进 LSH code 的搜索策略；

LSH 的应用研究主要在：

发展更好的数据结构和搜索策略，以实现更好的搜索质量和效率；

LSH 是有限制的，因为他完全是概率的和数据无关的，因此没有考虑数据的分布。因此，也有很多机遇学习的哈希方法，他们考虑了数据的分布情况和标签信息。

从机器学习的角度来看，LSH 就是一种概率的，相似度保留的降为方法。因此，他们主要的研究重心在于发展不同的 LSH 函数（能提供在特定相似度度量方法下的无偏估计，并且伴随晓得方差，小的存储，以及更快的计算）。

3.1 族（ps.不同的距离或相似度度量对应不同的 LSH 函数族）

LSH：一族能够将在输入空间相似的 items 以更高概率映射到相同哈希码的哈希函数。其更精准的定义在右边。

Definition 1 (Locality-sensitive hashing): A family of $\mathcal{H}$ is called (R, cR, P_1, P_2)-sensitive if for any two items $\mathbf{p}$ and $\mathbf{q}$,

- if $\text{dist}(\mathbf{p},\mathbf{q}) \leqslant R$, then $\text{Prob}[h(\mathbf{p}) = h(\mathbf{q})] \geqslant P_1$,
- if $\text{dist}(\mathbf{p},\mathbf{q}) \geqslant cR$, then $\text{Prob}[h(\mathbf{p}) = h(\mathbf{q})] \leqslant P_2$.

其中，c>1, P1>P2。参数$\rho = \frac{\log(1/P_1)}{\log(1/P_2)}$用来衡量搜索的性能，$\rho$越小，性能越好。

特别的，给定上述的 LSH 敏感性族，存在一种算法，对于 cR-ANN 搜索而言，有➔

near neighbor problem which uses $O(dn + n^{1+\rho})$ space, with query time dominated by $O(n^{\rho})$ distance computations and $O(n^{\rho} \log_{1/p_2} n)$ evaluations of hash functions

LSH 策略通过级联多个 hash function, 从而扩大 P1 和 P2 的鸿沟，从而提高准确率率。特别的，对于 K 个函数 h1(x),…,hK(x), 这些都是从相应的 LSH 的族中随机独立的选择出来的（依据特定的分布，抽取 hash fucntion 的参数）。

同时，为了提高召回率，LSH 策略使用了 L 个 hash table。每一个 table 分别建立自己的 buckets 的 hash code。

因此，gl(x)实际的 hash code 会有 LxK。这就导致直接进行索引会很困难。因此，我们只保留非空的 buckets。

3.2 基于 Lp 范数距离 (ps. 去下界最大整数+code 的整数取值)

3.2.1 基于 p-稳定分布的 LSH 函数族

分布 D 具有 p-稳定：

called *p-stable*, where $p \geqslant 0$, if for any n real numbers $v_1 \cdots v_n$ and i.i.d. variables $X_1 \cdots X_n$ with distribution $\mathcal{D}$, the random variable $\sum_{i=1}^{n} v_i X_i$ has the same distribution as the variable $(\sum_{i=1}^{n} |v_i|^p)^{1/p} X$, where X is a random variable with distribution $\mathcal{D}$. The well-known Gaussian distribution $\mathcal{D}_G$, defined by the density function $g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$, is 2-stable.

如果 p=1,那么ρ=1/c+O(R/r)。

如果 p=2,那么ρ=1/c

（ps.ρ越小，性能越好）

利用 p-稳定分布，我们有哈希函数：$h_{\mathbf{w},b}(\mathbf{x}) = \lfloor \frac{\mathbf{w}^T\mathbf{x}+b}{r} \rfloor$。其中，w 和 x 一样，也是 d 维度的向量。（特别的，w 的值是从 p-稳定分布总独立抽样产生，b 是从[0,r]的均匀分布中抽样产生）。

另外，我们可以获得如下结论：

$$P(h_{\mathbf{w},b}(\mathbf{x}_1) = h_{\mathbf{w},b}(\mathbf{x}_2)) = \int_0^r \frac{1}{c} f_p(\frac{t}{c})(1 - \frac{t}{r})dt.$$

其中，$c = \|\mathbf{x}_1 - \mathbf{x}_2\|_p$。

It is claimed that uniform quantization [72] without the offset b, $h_{\mathbf{w}}(\mathbf{x}) = \lfloor \frac{\mathbf{w}^T\mathbf{x}}{r} \rfloor$ is more accurate and uses fewer bits than the scheme with the offset.

3.2.2 基于 Leech Lattice 的 LSH 函数族

1.首先将数据向之前一样映射到 t 维度（上一节 t=1）。

2.应用 Leech lattice 方法，将 t 维度的量分割到 cells。

3.2.3 基于球面的 LSH

专门面向位于单位球面上分布分数据点。

3.2.4 Beyond LSH

特别的改善了解决（c,1）-ANN 问题的能力。它建立了两个层次的 hashing 结构。

1.外部哈希表：通过一个过滤过程，分割数据到不同的桶中，同时确保每个桶中的所有点不超过一个常数。同时对剩余的点，建立一个最小化的封闭球形。

2.内部哈希表：首先计算非空 bucket 中的球的中心，然后将点按照球的中心进行有重叠区域的分割（保证同球不同点到球心的距离的差距在[-1,1]之内，保证重叠区域到球心的距离在[0,1]之内。）

查询过程：根据外部哈希表将查询样本点定位到一个桶中，然后查询数据点到之前分割子集的距离，然后对这个子集进行 LSH。

3.3 基于角度的距离 (ps. 基于符号函数 sign(-))

3.3.1 随机投影

假设在原始空间中，xi 与 xj 的角度计算为 $\theta(\mathbf{x}_i, \mathbf{x}_j) = \arccos \frac{\mathbf{x}_i^T \mathbf{x}_j}{\|\mathbf{x}_i\|_2 \|\mathbf{x}_j\|_2}$。

相应的哈希/投影函数为：$h(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x})$。（其中 w 是从标准高斯分布中抽取的参数）

那么，我们有 $P(h(\mathbf{x}_i) = h(\mathbf{x}_j)) = 1 - \frac{\theta(\mathbf{x}_i, \mathbf{x}_j)}{\pi}$。

3.3.2 超比特 LSH

将投影分成 G 组，每组 B 个随机投影。那么总过获得 GxB 个随机投影。

3.3.3 Kernel LSH （利用数据库中的样本来近似随机投影向量）

因为在希尔伯特空间进行随机投影，因此在投影前，我们有：$\theta(\mathbf{x}_i, \mathbf{x}_j) = \arccos \frac{\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)}{\|\phi(\mathbf{x}_i)\|_2 \|\phi(\mathbf{x}_j)\|_2}$

Kernel LSH 的关键在于利用高斯分布来构建一个投影向量 w。为此，我们定义 $\mathbf{z}_t = \frac{1}{t} \sum_{i \in \mathcal{S}_t} \phi(\mathbf{x}_i)$，其中 t 是一个自然数，S 是由 database 中随机抽取的 t 个 items 组成的集合。由中心极限定理我们有，当 t 足够大时，随机变量 $\tilde{\mathbf{z}}_t = \sqrt{t} \boldsymbol{\Sigma}^{-1/2} (\mathbf{z}_t - \boldsymbol{\mu})$ 满足正态分布 $N(\mathbf{0}, \mathbf{I})$。然后，如果另 w=z，我们就有哈希函数为：

$$h(\phi(\mathbf{x})) = \begin{cases} 1 & \text{if } \phi(\mathbf{x}) \boldsymbol{\Sigma}^{-1/2} \tilde{\mathbf{z}}_t \geqslant 0 \\ 0 & \text{otherwise.} \end{cases}$$

其中方差 $\boldsymbol{\Sigma}$ 和均值 $\boldsymbol{\mu}$ 是通过随机抽取 p 个数据集的 items 来进行评估的（用类此 KPCA 的技术）。

- Select p data points and form a kernel matrix K over this data.
- Form the hash table over database items: for each hash function $h(\phi(\boldsymbol{x}))$, form $\boldsymbol{e}_S$ by selecting t indices at random from $[1, ..., p]$, then form $\boldsymbol{w} = K^{-1/2} \boldsymbol{e}_S$, and assign bits according to $h(\phi(\boldsymbol{x})) = \text{sign}(\sum_i \boldsymbol{w}(i) \kappa(\boldsymbol{x}, \boldsymbol{x}_i))$.
- For each query, form its hash key using these hash functions and employ existing LSH methods to find the approximate nearest neighbors.

3.3.4 LSH with learnt metric

通过半监督的方法，学习到一个 Mahalanobis metric A，然后有：$\theta(\mathbf{x}_i, \mathbf{x}_j) = \arccos \frac{\mathbf{x}_i^T \mathbf{A} \mathbf{x}_j}{\|\mathbf{G}\mathbf{x}_i\|_2 \|\mathbf{G}\mathbf{x}_j\|_2}$

其中 G'G=A。相应的 Hash function 为：

$$h_{r,A}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } r^T G x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

其中，r 是从 d 维度的高斯分布中产生。

3.3.5 伴随(concomitant) LSH

伴随最小哈希：

假设有 2^K 个投影，{w1,w2,…,w{2^K}}，并且每一个 w 都是从 d-正态分布中抽取。然后通过两部计算哈希编码：

1.计算 2^K 个投影；

2.输出其中值最小的索引；

$$h_c(\mathbf{x}) = \arg\min_{k=1}^{2^K} \mathbf{w}_k^T \mathbf{x}$$

同时概率$\mathrm{Prob}[h_c(\mathbf{x}_1) = h_c(\mathbf{x}_2)]$与$\frac{\mathbf{x}_1^T \mathbf{x}_2}{\|\mathbf{x}_1\|_2 \|\mathbf{x}_2\|_2}$成正相关。

伴随 L 最小哈希：

记录每组 2^K 个投影中结果值最小的 L 个索引

级联版本的伴随哈希：concomitant min & max hashing

因为当 K=20 时，投影就有 1048576 个，太多了。所以，级联版本的哈希建议，我们建立两个一致性哈希函数，每个产生 10 个 code，总过 20 个 code。这样就只要 2x2^10 次投影。然后，有：

$$[\arg\min_{k=1}^{2^K} \mathbf{w}_k^T \mathbf{x}, \arg\max_{k=1}^{2^K} \mathbf{w}_k^T \mathbf{x}]$$

3.3.6 超平面哈希 (ps.搜索一个查询超平面的最近邻点)

一个点到一个超平面 n 的距离为：$d(P_\mathbf{n}, \mathbf{x}) = \|\mathbf{n}^T \mathbf{x}\|$

Basic hash function:

$$h(\mathbf{z}) = \begin{cases} h_{\mathbf{u},\mathbf{v}}(\mathbf{z}, \mathbf{z}) & \text{if } \mathbf{z} \text{ is a database vector} \\ h_{\mathbf{u},\mathbf{v}}(\mathbf{z}, -\mathbf{z}) & \text{if } \mathbf{z} \text{ is a query hyperplane normal,} \end{cases}$$

Here $h_{\mathbf{u},\mathbf{v}}(\mathbf{a}, \mathbf{b}) = [h_\mathbf{u}(\mathbf{a})\ h_\mathbf{v}(\mathbf{b})] = [\mathrm{sign}(\mathbf{u}^T\mathbf{a})\ \mathrm{sign}(\mathbf{v}^T\mathbf{b})]$, where $\mathbf{u}$ and $\mathbf{v}$ are sampled independently from a standard Gaussian distribution.

XOR 1-bit hyperplane hashing

$$h(\mathbf{z}) = \begin{cases} h_\mathbf{u}(\mathbf{z}) \oplus h_\mathbf{v}(\mathbf{z}) & \text{if } \mathbf{z} \text{ is a database vector} \\ h_\mathbf{u}(\mathbf{z}) \oplus h_\mathbf{v}(-\mathbf{z}) & \text{if } \mathbf{z} \text{ is a hyperplane normal} \end{cases}$$

Embedding hyperplane hashing

$$h(\mathbf{z}) = \begin{cases} h_\mathbf{u}(\bar{\mathbf{z}}) & \text{if } \mathbf{z} \text{ is a database vector} \\ h_\mathbf{u}(-\bar{\mathbf{z}}) & \text{if } \mathbf{z} \text{ is a query hyperplane normal} \end{cases}, \bar{\mathbf{a}} = \mathrm{vec}(\mathbf{a}\mathbf{a}^T) [a_1^2, a_1 a_2, \cdots, a_1 a_d, a_2 a_1, a_2^2, a_2 a_3, \cdots, a_d^2].$$

3.4 基于汉明距离 (ps. for binary data)

对于二值向量而言，有 h(y)=yk，其中 k={1,2,3..,d}是一个随机抽取的索引。

3.5 基于杰卡德(Jaccard)系数 （ps. By permutation）

3.5.1 min hash

对于杰卡德系数，在原始空间有相似度：$\mathrm{sim}(\mathcal{A}, \mathcal{B}) = \frac{\|\mathcal{A} \cap \mathcal{B}\|}{\|\mathcal{A} \cup \mathcal{B}\|}$。距离：$1 - \mathrm{sim}(\mathcal{A}, \mathcal{B})$。

最小哈希是针对杰卡德相似度的哈希。定义π为一个原数组的随机排列，我们有如下哈希：

$$h(\mathcal{A}) = \min_{a \in \mathcal{A}} \pi(a)$$

如果数据集 A,B 有 K 个哈希值，那么他们的杰卡德相似度就为：

$$\frac{1}{K} \sum_{k=1}^{K} \delta[h_k(\mathcal{A}) = h_k(\mathcal{B})]$$

3.5.2 K-min sketch

1.记录整个序列的最小 K 个值的位置

2.将序列分成 K 个 bin，记录每个 bin 中，最小值在这个 bin 中的位置。

3.5.3 Min-max Hash

相对于 min-hash，同时记录最小和最大值的位置。

3.5.4 B-bit minwise hashing

只是用 min-hash 的索引的最低处的 b 个比特最为 hash 值。

3.5.5 Sim-min-hash

将原有的针对序列的 min-hash 算法扩展到针对实数。

首先，将所有实数，并给每个实数建立一个 index。将 index 看做是 min-hash 中的 word，建立一些随机排序，并获得 min-hash 的索引值。

3.6 χ^2距离

原始空间两个向量的χ^2距离定义为：

$$\chi^2(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{t=1}^{d} \frac{(x_{it} - x_{jt})^2}{x_{it} - x_{jt}}}.$$

然后，χ^2-LSH 哈希函数定义为：

$$h_{\mathbf{w},b}(\mathbf{x}) = \lfloor g_r(\mathbf{w}^T\mathbf{x}) + b \rfloor$$

where $g_r(x) = \frac{1}{2}(\sqrt{\frac{8x}{r^2}+1}-1)$，并且 **w** 是从 2-stable 分布中抽样得到。

3.7 其他相似度

3.7.1 Rank similarity

Winner Take All(WTA) hash: 生成 K 个随机排列$\{\pi_k\}$。每一个排列对向量 x 进行重排，得到$\bar{\mathbf{x}}$。并且第 k 个 hash code 的值为，第 k 个重排的最大元素索引-1，即$\arg\max_{i=1}^{T} \bar{x}_i$-1 （最终为 0~T-1）。

3.7.2 Shift invariant kernels

首先，获得随机傅里叶特征（RFF）：

$$\phi_{\mathbf{w},b}(\mathbf{x}) = \sqrt{2}\cos(\mathbf{w}^T\mathbf{x} + b),$$

where $\mathbf{w} \sim P_K$ and $b \sim \mathrm{Unif}[0, 2\pi]$. For example, for the Gaussian Kernel $K(s) = e^{-\gamma\|s\|^2/2}$, $\mathbf{w} \sim \mathrm{Normal}(\mathbf{0}, \gamma\mathbf{I})$. It can be shown that $\mathrm{E}_{\mathbf{w},b}[\phi_{\mathbf{w},b}(\mathbf{x})\phi_{\mathbf{w},b}(\mathbf{y})] = K(\mathbf{x}, \mathbf{y})$.

然后，获得二值 code:

$\mathrm{sign}(\phi_{\mathbf{w},b}(\mathbf{x}) + t)$，其中阈值 t~Unif[-1,1]

3.7.3 Non-metric distance

将数据嵌入到隐性的再生核希尔伯特空间，并在 RKHS 中定义 hash function。

3.7.4 Arbitrary distance measures

首先建立线性投影函数：

$$f(\mathbf{x}; \mathbf{a}_1, \mathbf{a}_2) = \frac{1}{2\,\mathrm{dist}(\mathbf{a}_1, \mathbf{a}_2)}(\mathrm{dist}^2(\mathbf{x}, \mathbf{a}_1) + \mathrm{dist}^2(\mathbf{a}_1, \mathbf{a}_2) - \mathrm{dist}^2(\mathbf{x}, \mathbf{a}_2))$$

然后建立哈希函数：

$$h(\mathbf{x}; \mathbf{a}_1, \mathbf{a}_2) = \begin{cases} 1 & \text{if } f(\mathbf{x}; \mathbf{a}_1, \mathbf{a}_2) \in [t_1, t_2] \\ 0 & \text{otherwise.} \end{cases}$$

，其中 a1,a2 是数据集中随机选取的 items。t1,t2 是为平滑 bit 而选取的阈值。

4.局部敏感哈希：检索，模型，分析

4.1 搜索 （避免穷尽检索）

4.1.1 基于熵的检索

给定一个查询样本 q,首先产生一些 q 附近（Ball(q,r)）的随机样本点 v。然后计算得到一些桶 h(v)，然后在这些桶中，进一步搜索 cR 最近邻点。

4.1.2 基于 LSH 森林的检索

利用 tree 结构来组织 hash table。并且在检索过程中的目的是找到符合要求的 subtree：

1. 自上而下的搜索：对于每个 LSH tree，找到有最大前缀匹配的叶子节点。
2. 自下而上的搜索：从第一步找到的叶子节点出发，回溯，找到有最大前缀匹配，且满足 ANN 要求的 subtree。

4.1.3 自适应 LSH 搜索

利用 B+tree。

4.1.4 多探头 LSH (ps.利用随机扰动的哈希码)

给定查询样本 q，先产生$g(\mathbf{q}) = (h_1(\mathbf{q}), h_2(\mathbf{q}), \cdots, h_K(\mathbf{q}))$。然后获得一系列的哈希扰动向量$\{\boldsymbol{\delta}_i = \{\delta_{i1}, \delta_{i2}, \cdots, \delta_{iK}\}\}$。然后产生一系列的邻近的哈希桶$\{g(\mathbf{q}) + \boldsymbol{\delta}_{o(i)}\}$。然后计算 g(q)和这些邻近的哈希桶的距离$\sum_{j=1}^{K} x_j^2(\delta_{ij})$，以此作为第 i 个桶的 score。然后用这个 score 排序邻近的哈希桶。

后验多探头 LSH 算法：构建一个包含查询对象和数据集先验知识的后验概率模型。

4.1.5 基于动态碰撞的搜索

利用 m 各单独的哈希函数构建动态紧致的哈希函数。

4.1.6 贝叶斯 LSH

估计查询样本 q 和一个候选样本 p 的一对哈希码（g(q),g(p)）的 k 个 bit 中有 m 个匹配的情况下，所具有真实相似度为 s 的概率。如果 s 大于阈值 t 的概率小于阈值ϵ，那么就删除这个候选点。

4.1.7 Fast LSH

ACHash：先用随机对角矩阵和哈达玛矩阵预处理向量 x，使得处理后的向量变得密集。然后把所得的向量乘以一个稀疏高斯矩阵，最终得到 x 的 k 个哈希值。

DHHash：同样，用随即对角矩阵和蛤蟆大矩阵预处理向量 x，然后乘以对角矩阵 M，和高斯矩阵 G，最后在乘以另外一个哈马达矩阵。$\zeta = \left\lfloor \frac{\mathbf{HGMHDx} + b}{w} \right\rfloor$。

4.1.8 Bi-level LSH

两级哈希。首先，使用 RP-tree 将数据划分到有限纵横比的小组，用于获得良好分割的族。然后，对每个小族，单独建立哈希表。

4.2 SortingKeys LSH （外存索引，键值度量）

目的是改进检索候选点时的 I/O 消耗。

过程：给哈希键值 g(.)设计一种新的度量方法，使得其像自然数一样，把距离近的对象存在一个索引文件中。

4.3 分析和建模

4.3.1 LSH 建模

召回率与准确率

4.3.2 最近邻搜索中的难题：相关对比度

对于查询样本 q 和数据集 X，其相关对比度为：$C_r^q = \frac{\mathrm{E}_{\mathbf{x}}[d(\mathbf{q},\mathbf{x})]}{\min_{\mathbf{x}}(d(\mathbf{q},\mathbf{x}))}$。对于多个查询，有$C_r = \frac{\mathrm{E}_{\mathbf{x},\mathbf{q}}[d(\mathbf{q},\mathbf{x})]}{\mathrm{E}_{\mathbf{q}}[\min_{\mathbf{x}}(d(\mathbf{q},\mathbf{x}))]} \approx \frac{1}{[1+\phi^{-1}(\frac{1}{N}+\phi(\frac{-1}{\sigma'}))\sigma']^{\frac{1}{p}}}$。由此，我们可以看到样本维度 d，数据库尺寸 N，测量范数 p，以及向量稀疏性对相关对比度的影响。

5. 学习的哈希：基于汉明嵌入

包含 3 个方面：哈希函数，在 coding space 的相似度测量，优化准则。

哈希函数：

哈希函数的设计可以基于线性映射、球面函数、核函数、神经网络、甚至无参数函数。

其中最常见的是基于线性映射的哈希函数：$y = \text{sign}(\mathbf{w}^T\mathbf{x}) \in \{0,1\}$。

另外一种则是基于最近邻向量指派，即$y = \arg\min_{k\in\{1,\cdots,K\}} \|\mathbf{x}-\mathbf{c}_k\|_2 \in \mathbb{Z}$，ck 是聚类中心。

哈希函数类型的选择影响着哈希码的计算效率，以及哈希的灵活性和分割空间的灵活性。而哈希函数的参数优化，则依赖于其距离的测量和保留方法。

相似度测量：

在 code space 主要有两种距离测量方法，即汉明距离和欧氏距离。

汉明距离：常用在 code 是 0/1 的二值向量情况下，其距离为相应位置不同值的 bit 数。其有变体，例如加权汉明距离，距离表（一个查询样本和字典的中元素的距离表）查询等。

欧氏距离：常用在最近邻指派，以及评估哈希码的对应的向量的情况下。通常可以通过查询实现获得的距离表来进行。因此，也有一些工作就是在给定哈希码的情况下，学习这样的一个距离表。

优化准则：

如果在 hash code 空间的距离计算能够足够精确的近似在真实数据空间的距离计算，那么 ANN 就能以更大的概率，获得真正的检索结果。这就引起了*相似度指派准则*，即将在原始空间计算的距离和在 hash 空间计算的距离的差距最小化。一种相似的准则是*编码一致性哈希*，即如果原来两个数据点相似度很大，那么如果在 hash 空间中差距大的话，给予大的惩罚。

处理相似度保留，还有*编码平衡*和*比特平衡准则*。前者要求各个 buckets 中的向量能均匀分布，后者要求 0/1 比特位的数量近似。

5.1 编码一致性哈希

最下化基于相似度加权的在 hash 空间的距离，即$s_{ij}d(\mathbf{y}_i,\mathbf{y}_j)$；或者是最大化基于距离加权的在 hash 空间的距离，即$d_{ij}d(\mathbf{y}_i,\mathbf{y}_j))$。

5.1.1 谱哈希

1.将相似的 items 映射到相似的哈希码（基于汉明距离衡量）；2.哈希比特数目尽可能少（同时满足编码平衡和比特平衡）；

用$\{\mathbf{y}_n\}n=1^N$表示长度 N 的哈希码。每一个为 M 维度的二值向量。用 sij 表示在原始欧式空间的相似度，我们有：

$$\min_{\mathbf{Y}} \quad \text{Trace}(\mathbf{Y}(\mathbf{D}-\mathbf{S})\mathbf{Y}^T)$$

比特平衡：$$\text{s.t.} \quad \mathbf{Y1}=\mathbf{0}$$

比特无关：$$\mathbf{YY}^T=\mathbf{I}$$

$$y_{im}\in\{-1,1\},$$

我们用如下方法求解上面的问题：

1. 使用 PCA 算法，获得 N-d 矩阵 X 的主成分：d 个。
2. 对每一个 PCA 方向，计算最小特征的 M 各 1D 的拉普拉斯特征函数。
3. 从总共 Mxd 个 1D 特征函数中，选择最小的 M 个特征值对应的 M 个 1D 特征函数。
4. 对所有 X 计算这 M 个频率上的特征函数，并进行阈值为零的处理，已获得平衡的二值哈希。

其中，1D 的拉普拉斯特征函数，以及特征值在均匀分布$[r_l,r_r]$上为：

$$\phi_f(x) = \sin(\frac{\pi}{2}+\frac{f\pi}{r_r-r_l}x) \qquad \lambda_f = 1-\exp(-\frac{\epsilon^2}{2}|\frac{f\pi}{r_r-r_l}|^2)$$

其中 *f*=1,2，…是频率，ϵ 是一个小的数值。其中，*f* 的计算中，相应维度，max-min 越大，其频率成分越多，占得比特越多。

但是，普哈希存在一些问题，例如对均匀分布的假设。因此有扩展的主成分哈希，分割一致性普哈希等。

5.1.2 核化的谱哈希

使用核矩阵去定义哈希函数：

$$\begin{aligned} y_m &= h_m(\mathbf{x}) \\ &= \operatorname{sign}(\sum_{t=1}^{T_m} w_{mt} K(\mathbf{s}_{mt}, \mathbf{x}) - b_m) \\ &= \operatorname{sign}(\sum_{t=1}^{T_m} w_{mt} < \phi(\mathbf{s}_{mt}), \phi(\mathbf{x}) > -b_m) \\ &= \operatorname{sign}(< \mathbf{v}_m, \phi(\mathbf{x}) > -b_m). \end{aligned}$$

其中，$\{\mathbf{s}_{mt}\}_{t=1}^{T_m}$是随机从数据集中抽取的 anchor items。并且针对不同的 hash function，通常我们固定 anchor items 的数量 Tm。且$\mathbf{v}_m = [w_{m1}\phi(\mathbf{s}_{m1}) \cdots w_{mT_m}\phi(\mathbf{s}_{mT_m})]^T$。

最终的优化目标函数为：

$$\min_{\{w_{mt}\}} \quad \operatorname{Trace}(\mathbf{Y}(\mathbf{D}-\mathbf{S})\mathbf{Y}^T) + \sum_{m=1}^{M} \|\mathbf{v}_m\|_2^2$$

5.1.3 超图谱哈希

利用超图的拉普拉斯矩阵来定义目标函数。

5.1.4 稀疏谱哈希

结合稀疏主成分分析（Sparse PCA）和助推的相似度敏感哈希（Boosting SSC）到传统的谱哈希中。即对拉普拉斯图的特征向量进行阈值化，约束其特征中的非零值数量。

5.1.5 基于独立成分分析的哈希

目标，编码后的互信息$I(y_1, y_2, \cdots, y_M) = I(\mathbf{w}_1^T\mathbf{x}, \mathbf{w}_2^T\mathbf{x}), \cdots, \mathbf{w}_M^T\mathbf{x})$最小化。或者表示为最大化：

$$\begin{aligned} \max_{\mathbf{W}} \quad & \sum_{m=1}^{M} \|c - \frac{1}{N}\sum_{n=1}^{N} g(\mathbf{W}^T\mathbf{x}_n)\|_2^2 \\ s.t. \quad & \mathbf{w}_i^T \operatorname{E}(\mathbf{x}\mathbf{x}^T)\mathbf{w}_j = \delta[i=j] \\ & \operatorname{trace}(\mathbf{W}^T\mathbf{\Sigma}\mathbf{W}) \le \eta. \end{aligned}$$

,其中：

约束 1 是 constraint of whiten condition，

g(u)是非二次函数，例如 g(u)=-exp(-u2/2)。

5.1.6 半监督哈希

Sij=1 指派到同类别 items,-1 指派到不同类别 items,0 指派到其他。目标函数为最大化经验拟合：

$$\sum_{i,j\in\{1,\cdots,N\}} s_{ij} \sum_{m=1}^{M} h_m(\mathbf{x}_i)h_m(\mathbf{x}_j)$$

where $h_k(\cdot) \in \{1,-1\}$. It is easily shown that this objective function 35 is equivalent to minimizing $\sum_{i,j\in\{1,\cdots,N\}} s_{ij} \sum_{m=1}^{M} \frac{(h_m(\mathbf{x}_i)-h_m(\mathbf{x}_j))^2}{2} = \frac{1}{2}\sum_{i,j\in\{1,\cdots,N\}} s_{ij}\|\mathbf{y}_i - \mathbf{y}_j\|_2^2$.，其中

同时如果考虑到 Bit 平衡，我们需要最大化 hash bit 的方差，那么我们有：

$$\operatorname{trace}[\mathbf{W}^T\mathbf{X}_l\mathbf{S}\mathbf{X}_l^T\mathbf{W}] + \eta\operatorname{trace}[\mathbf{W}^T\mathbf{X}\mathbf{X}^T\mathbf{W}]$$

进一步，如果我们希望 W 矩阵是正交的（投影间无关），即$\mathbf{W}^T\mathbf{W} = \mathbf{I}$，我们有：

$$\begin{aligned} &\operatorname{trace}[\mathbf{W}^T\mathbf{X}_l\mathbf{S}\mathbf{X}_l^T\mathbf{W}] + \eta\operatorname{trace}[\mathbf{W}^T\mathbf{X}\mathbf{X}^T\mathbf{W}] \\ &+ \rho\|\mathbf{W}^T\mathbf{W} - \mathbf{I}\|_F^2, \end{aligned}$$

5.1.7 LDA 哈希

目的是找到满足如下最小化目标函数的二值编码：

$$\alpha\operatorname{E}\{\|\mathbf{y}_i - \mathbf{y}_j\|^2 | (i,j) \in \mathcal{P}\} - \operatorname{E}\{\|\mathbf{y}_i - \mathbf{y}_j\|^2 | (i,j) \in \mathcal{N}\}$$

其中 P 表示相似样本对，N 表示不相似样本对，where $\mathbf{y} = \operatorname{sign}(\mathbf{W}^T\mathbf{x} + \mathbf{b})$。

LDA 哈希由两个过程组成：1.找到投影矩阵；2.找到产生哈希二值的阈值；

首先，LDA 哈希松弛二值编码条件，即去掉 sign()，有：

$$\alpha\operatorname{E}\{\|\mathbf{W}^T\mathbf{x}_i - \mathbf{W}^T\mathbf{x}_j\|^2 | (i,j) \in \mathcal{P}\} - \operatorname{E}\{\|\mathbf{W}^T\mathbf{x}_i - \mathbf{W}^T\mathbf{x}_j\|^2 | (i,j) \in \mathcal{N}\}$$

可以表示为：$\alpha\operatorname{trace}\{\mathbf{W}^T\mathbf{\Sigma}_p\mathbf{W}\} - \operatorname{trace}\{\mathbf{W}^T\mathbf{\Sigma}_n\mathbf{W}\}$。

其中$\mathbf{\Sigma}_p = \operatorname{E}\{(\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T | (i,j) \in \mathcal{P}\}$，$\mathbf{\Sigma}_n = \operatorname{E}\{(\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T | (i,j) \in \mathcal{N}\}$。

然后，就是找到最小化下式的阈值：

$$\alpha\,\mathrm{E}\{\mathrm{sign}\{\mathbf{W}^T\mathbf{x}_i-\mathbf{b}\}-\mathrm{sign}\{\mathbf{W}^T\mathbf{x}_j-\mathbf{b}\}|(i,j)\in\mathcal{P}\}\ -\mathrm{E}\{\mathrm{sign}\{\mathbf{W}^T\mathbf{x}_i-\mathbf{b}\}-\mathrm{sign}\{\mathbf{W}^T\mathbf{x}_j-\mathbf{b}\}|(i,j)\in\mathcal{N}\}$$

这个可以分解为 k 个子问题，每个找到 $\mathbf{w}_k^T\mathbf{x}-b_k$对应的 bk。

5.1.8 拓扑保留哈希

保留邻域的排序以及保留数据的拓扑

首先，编码一致性要求最大化：

$$\frac{1}{2}\sum_{i,j,s,t}\mathrm{sign}(d^o_{i,j}-d^o_{s,t})\,\mathrm{sign}(d^h_{i,j}-d^h_{s,t})\approx\frac{1}{2}\sum_{i,j,s,t}(d^o_{i,j}-d^o_{s,t})(d^h_{i,j}-d^h_{s,t})$$

其中，do 和 dh 分别为在原始空间和在汉明空间的距离。

这个编码一致性，可以转化为排序一致性，即要求最大化：

$$\frac{1}{2}\sum_{i,j}d^o_{i,j}d^h_{i,j}\ =\frac{1}{2}\sum_{i,j}d^o_{i,j}\|\mathbf{y}_i-\mathbf{y}_j\|_2^2$$

$$=\ \mathrm{trace}(\mathbf{Y}\mathbf{L}_t\mathbf{Y}^T)$$

where $\mathbf{L}_t=\mathbf{D}_t-\mathbf{S}_t$, $\mathbf{D}_t=\mathrm{diag}(\mathbf{S}_t\mathbf{1})$ and $s_t(i,j)=f(d^o_{ij})$ with $f(\cdot)$ is monotonically non-decreasing.

其中

而保留数据拓扑，则要求最小化：

$$\frac{1}{2}\sum_{ij}s_{ij}\|\mathbf{y}_i-\mathbf{y}_j\|_2^2\ =\ \mathrm{trace}(\mathbf{Y}\mathbf{L}_s\mathbf{Y}^T)$$

因此综上，我们有全局的目标函数：

$$\max\ \frac{\mathrm{trace}(\mathbf{W}^T\mathbf{X}(\mathbf{L}_t+\alpha\mathbf{I})\mathbf{X}^T\mathbf{W})}{\mathrm{trace}(\mathbf{W}^T\mathbf{X}\mathbf{L}_s\mathbf{X}^T\mathbf{W})}$$

，其中 I 为正则化项。

5.1.9 基于图的哈希

利用 anchor graph（的拉普拉斯矩阵）来近似邻接图（的拉普拉斯矩阵）。

5.2 相似度指派哈希

直接比较输入空间和哈希码空间中的相似度

5.2.1 基于二值重构嵌入

最小化在输入空间的欧式距离和在 hash 码空间的汉明距离。有目标函数为：

$$\min\sum_{(i,j)\in\mathcal{N}}(\frac{1}{2}\|\mathbf{x}_i-\mathbf{x}_j\|_f^2-\frac{1}{M}\|\mathbf{y}_i-\mathbf{y}_j\|_2^2)^2$$

其中的哈希函数形式如下：

$$y_{nm}=h_m(\mathbf{x})=\mathrm{sign}(\sum_{t=1}^{T_m}w_{mt}K(\mathbf{s}_{mt},\mathbf{x}))$$

where $\{\mathbf{s}_{mt}\}_{t=1}^{T_m}$ are sampled data items forming the hashing function $h_m(\cdot)\in\{h_1(\cdot),\cdots,h_M(\cdot)\}$, $K(\cdot,\cdot)$ is a kernel function, and $\{w_{mt}\}$ are the weights to be learnt.

，

5.2.2 基于 kernel 的监督哈希

1.利用 kernel 去构建哈希函数；

$$y_{nm}=h_m(\mathbf{x}_n)=\mathrm{sign}(\sum_{t=1}^{T_m}w_{mt}K(\mathbf{s}_{mt},\mathbf{x})+b)$$

2 最小化在输入空间的欧式距离和在 hash 码空间的汉明距离。目标函数为：

$$\min\sum_{(i,j)\in\mathcal{L}}(s_{ij}-\mathrm{affinity}(\mathbf{y}_i,\mathbf{y}_j))^2$$

其中，L 是标记的样本集合，$\mathrm{affinity}(\mathbf{y}_i,\mathbf{y}_j)=M-\ \|\mathbf{y}_i-\mathbf{y}_j\|_1,\ \ \mathbf{y}\in\{1,-1\}^M$。

5.2.3 Spec Hashing

将每个 item 视为样本，将他们的相似度视为概率。目的是使输入空间和汉明空间的概率分布均衡化。

用s_{ij}^i表示输入空间归一化的相似度$(\sum_{ij} s_{ij}^i = 1)$,s_{ij}^h表示汉明空间归一化的相似度$s_{ij}^h = \frac{1}{Z}\exp\left(-\lambda \operatorname{dist}_h(i,j)\right)$,$Z = \sum_{ij}\exp\left(-\lambda \operatorname{dist}_h(i,j)\right)$。我们有如下的目标函数:

$$\begin{aligned}\min \quad & \mathrm{KL}(\{s_{ij}^i\}||\{s_{ij}^h\}) \\ = & -\sum_{ij}\lambda s_{ij}^i \log s_{ij}^h = \lambda\sum_{ij} s_{ij}^u \operatorname{dist}_h(i,j) + \log\sum_{ij}\exp\left(-\lambda \operatorname{dist}_h(i,j)\right)\end{aligned}$$

5.2.4 双线性超平面哈希

双线性超平面哈希首先将数据向量转换到成高纬度向量:

$$\bar{\mathbf{a}} = \mathrm{vec}(\mathbf{a}\mathbf{a}^T) = [a_1^2, a_1a_2, \cdots, a_1a_d, a_2a_1, a_2^2, a_2a_3, \cdots, a_d^2]$$

然后基于这个高纬度向量，产生双线性超平面 hash function:

$$h(\mathbf{z}) = \begin{cases} \mathrm{sign}(\mathbf{u}^T\mathbf{z}\mathbf{z}^T\mathbf{v}) & \text{if } \mathbf{z} \text{ is a database vector} \\ \mathrm{sign}(-\mathbf{u}^T\mathbf{z}\mathbf{z}^T\mathbf{v}) & \text{if } \mathbf{z} \text{ is a hyperplane normal} \end{cases}$$

其中，u 和 v 是从标准高斯分布中单独抽样产生。

当然，u 和 v 也可以通过利用相似度信息，学习产生：

$$\min_{\{\mathbf{u}_k,\mathbf{v}_k\}_{k=1}^K} \|\frac{1}{K}\mathbf{Y}^T\mathbf{Y} - \mathbf{S}\| \quad \text{where } \mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_N] \quad \text{and} \quad s_{ij} = \begin{cases} 1 & \text{if } \cos(\theta_{\mathbf{x}_i,\mathbf{x}_j}) \geqslant t_1 \\ -1 & \text{if } \cos(\theta_{\mathbf{x}_i,\mathbf{x}_j}) \leqslant t_2 \\ 2|\cos(\theta_{\mathbf{x}_i,\mathbf{x}_j})| - 1 & \text{otherwise,} \end{cases}$$

5.3 基于顺序保留的 hashing

目的：使得输入空间和编码空间的相关数据 items 的顺序对齐最大化。

5.3.1 最小化损失的哈希

利用一个 hinge-like 的损失函数，来构建惩罚项，来处理那些原本相似（不相似）的点却被分配差距很大（很小）的 hash code 的情况。具体目标函数如下：

$$\min \sum_{(i,j)\in\mathcal{L}} I[s_{ij}=1]\max(\|\mathbf{y}_i - \mathbf{y}_i\|_1 - \rho + 1, 0) + I[s_{ij}=0]\lambda\max(\rho - \|\mathbf{y}_i - \mathbf{y}_i\|_1 + 1, 0)$$

其中，前一项表示，如果原本相似（sij=1）的点，如果 code 差距越大（|yi-yj|），那么惩罚越大。后一项则表示，如果原本不相似（sij=0）的点，如果 code 差距越小，那么惩罚也越大。

5.3.2 基于排序损失

使得输入空间和汉明空间的相关数据 items 的顺序对齐最大化。

给定 xn,针对数据集 X，依据原始空间的欧式距离和汉明空间的汉明距离，分别成 M 类：

$$(\mathcal{C}_{n0}^e, \mathcal{C}_{n1}^e, \cdots, \mathcal{C}_{nM}^e), \quad (\mathcal{C}_{n0}^h, \mathcal{C}_{n1}^h, \cdots, \mathcal{C}_{nM}^h)$$

然偶，我们建立最大化分类对齐的目标函数（最大化每个 item 的按类的检索排序），即：

$$\begin{aligned}L(\mathbf{h}(\cdot);\mathcal{X}) &= \sum_{n=1}^N L(\mathbf{h}(\cdot);\mathbf{x}_n) = \sum_{n=1}^N\sum_{m=0}^{M-1} L(\mathbf{h}(\cdot);\mathbf{x}_n,m) \\ &= \sum_{n=1}^N\sum_{m=0}^{M-1}(|\mathcal{N}_{nm}^e - \mathcal{N}_{nm}^h| + \lambda|\mathcal{N}_{nm}^h - \mathcal{N}_{nm}^e|)\end{aligned}$$

其中 where $\mathcal{N}_{nm}^e = \cup_{j=0}^{m}\mathcal{C}_{nj}^e$ and $\mathcal{N}_{nm}^h = \cup_{j=0}^{m}\mathcal{C}_{nj}^h$。

如果有哈希函数如下：$\mathbf{h}(\mathbf{x}) = \mathrm{sign}(\mathbf{W}^T\mathbf{x} + \mathbf{b}) = [\mathrm{sign}(\mathbf{w}_1^T\mathbf{x} + b_1) \cdots \mathrm{sign}(\mathbf{w}_m^T\mathbf{x} + b_m)]^T$

我们有：$L(\mathbf{W};\mathbf{x}_n,i) = \sum_{\mathbf{x}'\in\mathcal{N}_{ni}^e}\mathrm{sign}(\|\mathbf{h}(\mathbf{x}_n) - \mathbf{h}(\mathbf{x}')\|_2^2 - i) + \lambda\sum_{\mathbf{x}'\notin\mathcal{N}_{ni}^e}\mathrm{sign}(i + 1 - \|\mathbf{h}(\mathbf{x}_n) - \mathbf{h}(\mathbf{x}')\|_2^2)$。

上式可以通过扔掉 sign function，使用 quadratic penalty algorithm 来完成。

5.3.3 三角损失哈希

保留三角顶点上的 items $(\mathbf{x}, \mathbf{x}^+, \mathbf{x}^-)$在哈希空间中的相对相似度。其中the pair $(\mathbf{x}, \mathbf{x}^+)$ is more similar than the pair $(\mathbf{x}, \mathbf{x}^-)$。为此，我们定义目标函数：

$$\ell_{\text{triplet}}(\mathbf{y}, \mathbf{y}^+, \mathbf{y}^-) = \max(\|\mathbf{y} - \mathbf{y}^+\|_1 - \|\mathbf{y} - \mathbf{y}^-\|_1 + 1, 0)$$

假设哈希函数为$\mathbf{h}(\mathbf{x}; \mathbf{W})$，我们有：

$$\sum_{(\mathbf{x},\mathbf{x}^+,\mathbf{x}^-)\in\mathcal{D}} \ell_{\text{triplet}}(\mathbf{h}(\mathbf{x}; \mathbf{W}), \mathbf{h}(\mathbf{x}^+; \mathbf{W}), \mathbf{h}(\mathbf{x}^-; \mathbf{W})) + \frac{\lambda}{2} \text{trace}(\mathbf{W}^T\mathbf{W})$$

5.3.4 Listwise 的监督哈希

该方法同样使用三角结构的 items 去近似 listwise loss。函数建立在三角张量 S 上，定义如下：

$$s(i; j, k) = \begin{cases} 1 & \text{if } \text{sim}(\mathbf{x}_i; \mathbf{j}) > \text{sim}(\mathbf{x}_i; \mathbf{k}) \\ -1 & \text{if } \text{sim}(\mathbf{x}_i; \mathbf{j}) < \text{sim}(\mathbf{x}_i; \mathbf{k}) \\ 0 & \text{if } \text{sim}(\mathbf{x}_i; \mathbf{j}) = \text{sim}(\mathbf{x}_i; \mathbf{k}) \end{cases}$$

目标是最小化下边的目标函数：

$$-\sum_{i,j,k} \mathbf{h}(\mathbf{x}_i)^T(\mathbf{h}(\mathbf{x}_j) - \mathbf{h}(\mathbf{x}_k))s_{ijk}$$

求解过程中可以通过仍掉$\mathbf{h}(\mathbf{x}; \mathbf{W}) = \text{sign}(\mathbf{W}^T\mathbf{x})$中的 sign operator 来实现。

5.3.5 相似度敏感性编码(SSC)

SSC 的目的是学习到一个加权的汉明嵌入：$\mathbf{h}(\mathbf{x}) = [\alpha_1 h(\mathbf{x}_1)\ \alpha_2 h(\mathbf{x}_2)\ \cdots\ \alpha_M h(\mathbf{x}_M)]$ 以适应特定的相似度。以 boosted SSC 为例，它通过使用 adaboost 算法(通过迭代弱分类器而产生最终的强分类器的算法)来学习处分类器。然后每个弱分类器的输出对应一个二值码，所有弱分类器的输出对应一个 hash 码。每个弱分类器进行加权，来计算加权汉明距离。其他类似的算法还有参数敏感哈希，遗忘哈希算法等。

一个 SSC 的例子，通过同时学习权重和哈希参数来进行 adaboost：

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\zeta}} \quad & \sum_{i=1}^{N} \zeta_i + C\|\boldsymbol{\alpha}\|_p \\ \text{s.t.} \quad & \boldsymbol{\alpha} \geqslant \mathbf{0}, \boldsymbol{\zeta} \geq \mathbf{0}, \\ & d_h(\mathbf{x}_i, \mathbf{x}_i^-) - d_h(\mathbf{x}_i, \mathbf{x}_i^+) \geqslant 1 - \zeta_i \forall i. \end{aligned}$$

5.4 正则化的空间分割

在不精确评估编码空间的距离的情况下，追求有效的空间分割。

5.4.1 互补的投影哈希(CPH)

CPH 根据先前计算的 m-1 个哈希，来计算第 m 个哈希。（通过检查数据点与之前 m-1 个分割空间的距离实现）。学习第 m 个哈希函数时，其对 xn 的权重为：

$$u_n^m = 1 + \sum_{j=1}^{m-1} H(\epsilon - |\mathbf{w}_m^T\mathbf{x}_n + b|), \text{ where } H(\cdot) = \tfrac{1}{2}(1 + \text{sign}(\cdot)) \text{ is a unit function.}$$

同时，如果考虑到 bit-balance 和 pair-wise bit balance（任意两个超平面将数据分割成 4 个子空间，每个子空间包含 N/4 的点）：

$$\sum_{n=1}^{N} h_1(\mathbf{x}_n) = 0, \sum_{n=1}^{N} h_2(\mathbf{x}_n) = 0, \sum_{n=1}^{N} h_1(\mathbf{x}_n)h_2(\mathbf{x}_n) = 0.$$

然后获得如下总的目标函数：

$$\min \sum_{n=1}^{N} u_n^m H(\epsilon - |\mathbf{w}_m^T\mathbf{x}_n + b|) + \alpha((\sum_{n=1}^{N} h_m(\mathbf{x}_n))^2 + \sum_{j=1}^{m-1}(\sum_{n=1}^{N} h_j(\mathbf{x}_n)h_m(\mathbf{x}_n))^2 \text{ where } h_m(\mathbf{x}) = (\mathbf{w}_\mathbf{m}^\mathrm{T}\mathbf{x} + \mathrm{b})$$

5.4.2 标签正则化的最大间隔哈希

使用边信息/辅助信息（Side Information：是指利用已有的边信息 Y 辅助对信息 X 进行编码，可以使得

信息 X 的编码长度更短）来找到能是间隔最大化的哈希函数。特别的，哈希函数使得相似的 items 获得相同的 code，不相似的 items 获得不同的 code. 用 P 表示{(i,j)}是相似的，我们下面给出目标函数：

$$\min_{\{y_i\},\mathbf{w},b,\{\xi_i\},\{\zeta\}} \|\mathbf{w}\|_2^2 + \frac{\lambda_1}{N}\sum_{n=1}^{N}\xi_n + \frac{\lambda_2}{N}\sum_{(i,j)\in\mathcal{S}}\zeta_{ij} \quad \text{s.t.} \quad \begin{array}{l} y_i(\mathbf{w}^T\mathbf{x}_i + b) + \xi_i \geqslant 1, \xi_i \geqslant 0, \forall i, \\ y_i y_j + \zeta_{ij} \geqslant 0. \forall (i,j) \in \mathcal{P}, \\ -l \leqslant \mathbf{w}^T\mathbf{x}_i + b \leqslant l. \end{array}$$

其中，约束 2 来自于边信息/辅助信息，约束 3 来自于比特约束。

与 BRE 类似，这儿的哈希函数也定义为: $h(\mathbf{x}) = \text{sign}(\sum_{t=1}^{T} v_t < \phi(\mathbf{s}_t), \phi(\mathbf{x}) > -b)$, $w = \sum_{t=1}^{T} v_t \phi(\mathbf{s}_t)$。

5.4.3 随机最大间隔哈希

随机最大间隔哈希，首先抽样 N 个 items，然后随机标定一半为-1，一半为 1.然后一个标准的 SVM 规范式形式的目标函数如下（最小间隔最大化；远离边界）：

$$\max \frac{1}{\|\mathbf{w}\|_2} \min[\min_{i=1}^{\frac{N'}{2}}(\mathbf{w}^T\mathbf{x}_i^+ + b), \min_{i=1}^{\frac{N'}{2}}(-\mathbf{w}^T\mathbf{x}_i^- - b)]$$

其中 x+是标记为 1 的样本，x—为标记为-1 的样本。

5.4.4 球面哈希

使用超球面函数去规范一个球面哈希函数:

$$h(\mathbf{x}) = \begin{cases} +1 & \text{if } d(\mathbf{p},\mathbf{x}) \leqslant t \\ 0 & \text{otherwise.} \end{cases}$$

相应的 hash code 由 K 个球面函数组成。并且给定两个哈希码 **y1**,**y2**，他们的距离计算为：

$\frac{\|\mathbf{y}_1 - \mathbf{y}_2\|_1}{\mathbf{y}_1^T\mathbf{y}_2}$，其中$\|\mathbf{y}_1 - \mathbf{y}_2\|_1$是汉明距离，$\mathbf{y}_1^T\mathbf{y}_2$是共有 1 的数量

5.4.5 密度敏感的哈希

1.在整个数据集上用 k-means 聚类算法，产生 K 个聚类族和聚类中心，i.e.,{u1,u2,…,uK}。

2.如果其中一个聚类中心 i 在聚类中心 j 的 r 个最近邻点内，那么我们就对此定义一个哈希函数：

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} - b), \text{ where } \mathbf{w} = \boldsymbol{\mu}_i - \boldsymbol{\mu}_j \text{ and } b = \tfrac{1}{2}(\boldsymbol{\mu}_i + \boldsymbol{\mu}_j)^T(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)$$

3.评估哈希函数（wm,bm）能否尽可能均匀平等的分割数据集。这个评估通过熵来进行：

$$-P_{m0}\log P_{m0} - -P_{m1}\log P_{m1}, \text{ where } P_{m0} = \frac{n_0}{n} \text{ and } P_{m1} = 1 - P_{m0}$$

其中，n 是总的样本点数目，n0 是哈希函数的超平面分割的其中一部分的数据点数

4.选出所有候选哈希函数中具有最大熵的 L 个哈希

5.5 基于加权汉明距离的哈希

用于查询相关的和查询独立的加权汉明距离的策略，计算在 code space 的距离。

5.5.1 多维度的谱哈希

目的是保证加权的汉明吸引力和原始空间的吸引力相等：

$$\min \quad \sum_{(i,j)\in\mathcal{N}} (w_{ij} - \mathbf{y}_i^T\boldsymbol{\Lambda}\mathbf{y}_j)^2 = \|\mathbf{W} - \mathbf{Y}^T\boldsymbol{\Lambda}\mathbf{Y}\|_F^2,$$ 且$\boldsymbol{\Lambda}$是对角矩阵。

其中，$\boldsymbol{\Lambda}$和 **Y** 都是需要优化的量。

如果用*(d,l)*表示主成分方向 *d* 上的第 *l* 个 1D 的特征函数，并有 *I=(d,l)*表示所有选择的特征函数。那么利用这些主成分特战术，加权的汉明吸引力计算如下：

$$\text{affinity}_d(i,j) = \sum_{(d,l)\in\mathcal{I}} \lambda_{dl}\,\text{sign}(\phi_{dl}(x_{id}))\,\text{sign}(\phi_{dl}(x_{jd})),$$

其中，x_{id}表示 x 在 d 方向上的投影，$\phi_{dl}(\cdot)$表示 *d* 方向上的第 *l* 个特征函数，λ_{dl}是相应的特征值。

然后，相应的加权汉明距离计算如下：

$$\text{affinity}(\mathbf{y}_i, \mathbf{y}_j) = \prod_d (1 + \text{affinity}_d(i,j)) - 1$$

5.5.2 加权哈希

使用加权汉明距离来评估哈希码的距离，即$\|\alpha^T(\mathbf{y}_i - \mathbf{y}_j)\|_2^2$。其优化目标函数如下：

$$\begin{aligned} \min \quad & \text{trace}(\text{diag}(\alpha)\mathbf{Y}\mathbf{L}\mathbf{Y}^T) + \lambda\|\frac{1}{n}\mathbf{Y}\mathbf{Y}^T - \mathbf{I}\|_F^2 \\ \text{s.t.} \quad & \mathbf{Y} \in \{-1,1\}^{M\times N}, \mathbf{Y}^T\mathbf{1} = \mathbf{0} \\ & \|\alpha\|_1 = 1 \\ & \frac{\alpha_1}{\text{var}(y_1)} = \frac{\alpha_2}{\text{var}(y_2)} = \cdots = \frac{\alpha_M}{\text{var}(y_M)}, \end{aligned}$$

其中 L 为拉普拉斯矩阵

上式通过去掉第一项约束，然后对 y 通过 M 个均值进行二值化求解。哈希函数$\mathrm{w}_{\mathrm{m}}^{\mathrm{T}}\mathrm{x}+\mathrm{b}$通过将数据 x 映射到 ym 来学习得到。

5.5.3 查询-自适应的比特哈希

通过查询的信息来学习权值。特别的，方法学习特定类别的 bit 权重，通过最小化同属一个类别的 hash codes 到其类中心的的加权汉明距离的均值。

5.5.4 查询-自适应哈希

目的是根据查询向量来选择哈希 bits。方法有两步组成：

1. 离线的哈希函数$\mathbf{h}(\mathbf{x}) = \text{sign}(\mathbf{W}^T\mathbf{x})\ (\{h_b(\mathbf{x}) = \text{sign}(\mathbf{w}_b^T\mathbf{x})\})$
2. 在线的哈希函数选择

5.6 其他的哈希学习算法

5.6.1 语义哈希

基于深度学习，找到那些能够重构输入数据的 hash codes。

5.6.2 样条回归哈希

找到一个全局的核形式的哈希函数：$h(\mathbf{x}) = \mathbf{v}^T\phi(\mathbf{x})$，确保这个基于全局哈希函数的哈希值和基于局部哈希函数的哈希值一致。每一个数据点的局部韩系函数是样条回归的形式：

$$h_n(\mathbf{x}) = \sum_{i=1}^t \beta_{ni}p_i(\mathbf{x})) + \sum_{i=1}^k \alpha_{ni}g_{ni}(\mathbf{x})$$

其中，$\{p_i(\mathbf{x})\}$是一系列原始的多项式，并且可以张成自由度小于 s 的多项式空间，$\{g_{ni}(\mathbf{x})\}$是 green 函数。

进而我们有全局的规范式：

$$\min_{\mathbf{v},\{h_i\},\{\mathbf{y}_n\}} \sum_{n=1}^N (\sum_{\mathbf{x}_i\in\mathcal{N}_n} \|\mathbf{h}_n(\mathbf{x}_i) - \mathbf{y}_i\|_2^2 + \gamma\psi_n(\mathbf{h}_n)) + \lambda(\sum_{n=1}^N \|\mathbf{h}(\mathbf{x}_n) - \mathbf{y}_n\|_2^2 + \gamma\|\mathbf{v}\|_2^2).$$

5.6.3 流形归纳哈希

1.聚类数据 items 到 K 类，{c1,c2,…,cK}；

2.使用现有的流形降维技术，将聚类中心嵌入到低维度，{y1,y2,…,yK}；

3.给定哈希函数：

$$\mathbf{h}(\mathbf{x}) = \text{sign}(\frac{\sum_{k=1}^K w(\mathbf{x},\mathbf{c}_k)\mathbf{y}_k}{\sum_{k=1}^K w(\mathbf{x},\mathbf{c}_k)})$$

5.6.4 非线性嵌入

一个精确的最近邻搜索算法，基于下面的核心不等式：

$$\|\mathbf{x}_1 - \mathbf{x}_2\|_2^2 \geqslant d((\mu_1 - \mu_2)^2 + (\sigma_1 - \sigma_2)^2)$$

其中$\mu = \frac{1}{d}\sum_{i=1}^{a} x_i$是所有输入的均值，$\sigma = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2$ 是标准方差。

这儿，我们首先把数据维度 *d* 划分成 *M* 个字向量，每个长度为 *dm*，然后利用上述不等式有：

$$|\mathbf{x}_1 - \mathbf{x}_2\|_2^2 \geqslant \sum_{m=1}^{M} d_m((\mu_{1m} - \mu_{2m})^2 + (\sigma_{1m} - \sigma_{2m})^2)$$

在搜索过程中，在计算精确的欧氏距离之前，我们会先上述不等式中的下界（右边），并与当前的最小欧氏距离比较，从而决定是否需要计算精确的欧式距离。

5.6.5 Anti-Sparse coding

核心：使得 hash code 中的非零元素尽可能的多。同时具有能精确的重构原始向量$\mathrm{x} \propto \mathrm{Wy}$的性能。

首先，计算下面问题：

$\mathbf{z}^* = \arg\min_{\mathbf{z}:\mathbf{W}\mathbf{z}=\mathbf{x}} \|\mathbf{z}\|_\infty$，其中$\|\mathbf{z}\|_\infty = \max_{i\in\{1,2,\cdots,K\}} |z_i|$，W为投影矩阵。

然后，计算二值码：$\mathrm{y} = \mathrm{sign}(\mathrm{z})$。

5.6.6 两步哈希

S1.学习二值码；S2.学习将输入 item 映射到 S1 获得的二值码的哈希函数；

Self-taught hashing:

S1.求解类似谱哈希的优化问题：

$$\begin{aligned} \min \quad & \mathrm{trace}(\mathbf{Y}\mathbf{L}\mathbf{Y}^T) \\ \text{s.t.} \quad & \mathbf{Y}\mathbf{D}\mathbf{Y}^T = \mathbf{I} \\ & \mathbf{Y}\mathbf{D}\mathbf{1} = 0, \end{aligned}$$

，然后通过 Y 的中值来阈值化获得哈希码；

S2.将 item 到特定 bit 的 0/1 置位看做一个分类问题，通过线性 SVM 来求解，并将该 SVM 作为哈希函数。

Sparse hashing:

S1.学习一个稀疏非负的嵌入，然后把正值置 1，零置 0。相应的学习的目标函数为：

$$\sum_{n=1}^{N} \|\mathbf{x}_n - \mathbf{P}^T\mathbf{z}_n\|_2^2 + \alpha \sum_{i=1}^{N}\sum_{j=1}^{N} s_{ij}\|\mathbf{z}_i - \mathbf{z}_j\|_2^2 + \lambda \sum_{n=1}^{N} \|\mathbf{z}_n\|_1$$

S2.针对每一个 bit 位，分别学习一个哈希函数：

$$\min_{\mathbf{w}} \sum_{n=1}^{N} \|\mathrm{y}_n - \mathbf{w}_m^t\mathbf{x}_n\|_2^2 + \lambda_1\|\mathbf{w}_m\|_1 + \lambda_2\|\mathbf{w}_m\|_2^2.$$

Local linear hashing:

S1.学习到能够保留局部线性结构的二值码：

$$\min_{\mathbf{Z},\mathbf{R},\mathbf{Y}} \quad \mathrm{trace}(\mathbf{Z}^T\mathbf{M}\mathbf{Z}) + \eta\|\mathbf{Y} - \mathbf{Z}\mathbf{R}\|_F^2 \quad \text{s.t.} \quad \mathbf{Y} \in \{1,-1\}^{N\times M}, \mathbf{R}^T\mathbf{R} = \mathbf{I}.$$

其中，Z 是非线性嵌入，M=(I-W)'(I-W),W 是线性重构的权值矩阵，通过下式的优化获得：

$$\begin{aligned} \min_{\mathbf{w}_n} \quad & \lambda\|\mathrm{s}_n^T\mathbf{w}_n\|_1 + \frac{1}{2}\|\mathbf{x}_n - \sum_{j\in\mathcal{N}(\mathbf{x}_n)} w_{ij}\mathbf{x}_n\|_2^2 \\ \text{s.t.} \quad & \mathbf{w}_n^T\mathbf{1} = 1, \end{aligned} \qquad \begin{aligned} & \text{where } \mathbf{w}_n = [w_{n1}, w_{n2}, \cdots, w_{nn}]^T, \text{ and } w_{nj} = 0 \text{ if } j \notin \\ & \mathcal{N}(\mathbf{x}_n).\ \mathrm{s}_n = [s_{n1}, s_{n2}, \cdots, s_{nn}]^T \text{ is a vector and } s_{nj} = \\ & \frac{\|\mathbf{x}_n - \mathbf{x}_j\|_2}{\sum_{t\in\mathcal{N}(\mathbf{x}_n)}\|\mathbf{x}_n - \mathbf{x}_t\|_2}. \end{aligned}$$

S2.针对 out-of sample extension，通过基于邻域的线性重构的思想，获得二值嵌入：

$\mathrm{y}_q = \mathrm{sign}(\mathbf{Y}^T\mathrm{w}_q)$，其中 wq 是一个局部的线性重构矩阵。

5.7 Beyond the Hamming Distance in the coding space

利用二值 code，设计测量方法以进行有效的距离测量。

5.7.1 曼哈顿距离

将汉明 code 有次序的转化为整数，然后利用整数的差距来代替原有的汉明距离。

5.7.2 非对称距离

假设有 K 个哈希函数：

$\{h_k(\mathbf{x}) = b_k(g_k(\mathbf{x}))\}$, where $g_k()$ is a real-valued embedding function and $b_k()$ is a binarization function.

非对称 1：基于期望：$\bar{g}_{kb} = \mathrm{E}(g_k(\mathbf{x})|h_k(\mathbf{x}) = b_k(g_k(\mathbf{x})) = b)$

其中 b=1/0。当进行在线搜索时，一个预先计算的距离查询表如下：

$$\{d_e(g_1(\mathbf{q}), \bar{g}_{10}), d_e(g_1(\mathbf{q}), \bar{g}_{11}), d_e(g_2(\mathbf{q}), \bar{g}_{20}), \ d_e(g_2(\mathbf{q}), \bar{g}_{21}), \cdots, d_e(g_K(\mathbf{q}), \bar{g}_{K0}), d_e(g_K(\mathbf{q}), \bar{g}_{K1})$$

其中$d_e(\cdot,\cdot)$ 计算欧式距离。然后，我们计算总的距离：$d_{ah}(\mathbf{q}, \mathbf{x}) = \sum_{k=1}^{K} d_e(g_k(\mathbf{q}), \bar{g}_{kh_k(\mathbf{x})})$

非对称 2：基于假设：$b_k(g_k(\mathbf{x})) = \delta[g_k(\mathbf{x}) > t_k]$，然后计算距离的下界：

$$d(g_k(\mathbf{q}), b_k(g_k(\mathbf{x}))) = \begin{cases} |g_k(\mathbf{q})| & \text{if } h_k(\mathbf{x}) \neq h_k(\mathbf{q}) \\ 0 & \text{otherwise.} \end{cases}$$

同样的，总的距离计算为$d_{ah}(\mathbf{q}, \mathbf{x}) = \sum_{k=1}^{K} d(g_k(\mathbf{q}), b_k(g_k(\mathbf{x})))$。

5.7.3 查询敏感哈希编码排序

一个对 RNN 搜索的非对称策略。它通过 PCA 的投影 W 来规范化哈希函数：$\mathrm{sign}(\mathbf{W}^T\mathbf{x}) = \mathrm{sign}(\mathbf{z})$。然后，沿着第 k 方向的投影计算为：

$$s_k(q_k, y_k, R) = \frac{P(z_k y_k > 0, |q_k - z_k| \leqslant R)}{P(|q_k - z_k| \leqslant R))}$$

直观的看，分子是那些在$|q_k - z_k| \leqslant R$，并且要映射到 yk 的点；分母是所有$|q_k - z_k| \leqslant R$的点。

然后全局的相似度为：$\prod_{k=1}^{K} s_k(q_k, y_k, R)$。同时，lookup table 也可以被用来加速距离据算。

5.7.4 Bit 重新配置

目的是从差的 hash fucntion 编码的 hash code 中学习到一个号的距离测量方法。

假设数据集维度为 M 的哈希码为 yn{n=1…N},且有$\text{similar pairs } \mathcal{M} = \{(i,j)\}$, $\text{dissimilar pairs } \mathcal{C} = \{(i,j)\}$。分别计算这些 pairs 上的差异性矩阵 Dm 和 Dc。其中每一列为$\mathbf{y}_i - \mathbf{y}_j$。然后我们建立下面的最大化问题：

$$\max_{\mathbf{W}} \ \frac{1}{n_c}\mathrm{trace}(\mathbf{W}^T\mathbf{D}_c\mathbf{D}_c^T\mathbf{W}) - \frac{1}{n_m}\mathrm{trace}(\mathbf{W}^T\mathbf{D}_m\mathbf{D}_m^T\mathbf{W}) + \frac{\eta}{n_s}\mathrm{trace}(\mathbf{W}^T\mathbf{Y}_s\mathbf{Y}_s^T\mathbf{W}) - \eta\,\mathrm{trace}(\mathbf{W}^T\boldsymbol{\mu}\boldsymbol{\mu}^T\mathbf{W})$$

其中,W 属于 bxt。第一项为最大化不相似 pairs 的差异性，第二项为最小化相似项的差异，最后两项最大化以实现 bit balance，其中 u 是 hash vector 的均值，Ys 为候选点子集 ns 的 hash codes。

6. 哈希学习：分层化

基于分层的哈希算法

6.1 1D 分层化

将 items 在投影方向上的值分割成多个部分。

6.1.1 转化编码

转换编码首先用 PCA 进行数据的转化，然后给每一个主成分方向分配一定数量的 bits。与谱哈希不同，转换编码首先使用 Bit 分配来决定哪个主成分方向被使用，并且决定要个这个方向分配多少 bits 位。

Bit 分配算法如下。同时，为了组成 hash function，每一个被选中的主成分方向 i 被聚类成 2^(mi)族，每

一个都是长度为 mi 的 0/1 编码向量。编码一个 item 是通过主成分方向投影外加一个量化组件（归类到特定的聚类中心的 hash code）形成的。其中，查询 item 和 hash code 的距离是通过累计查询和数据集 item 在每个方向上的中心点的距离实现。

Algorithm 1 Distribute M bits into the principal directions

1. Initialization: $e_i \leftarrow \log_2 \sigma_i$, $m_i \leftarrow 0$.
2. **for** $j = 1$ to b **do**
3. $\quad i \leftarrow \arg\max e_i$.
4. $\quad m_i \leftarrow m_i + 1$.
5. $\quad e_i \leftarrow e_i - 1$.
6. **end for**

6.1.2 双比特分层化

与转化编码不同，双 bit 分层化只在每个主成分方向建立 3 个聚类中心，编码分别为 00，01，11。这样，那些与聚类中心点相邻的点的汉明距离就是 1，不相邻的汉明距离就是 2。

Local digit coding 则只用 1bit 来编码数据点的每一个维度，通过一个阈值来决定 0/1。

6.2 超立方体分层

将数据 item 分层到一个超立方体的节点上，即 $\text{a vector belonging to } \{[y_1, y_2, \cdots, y_M] | y_m \in \{-1, 1\}\}$。

6.2.1 迭代分层

目的是通过将每个 bit 视为其在相应维度上的量化的值，最小化 hash codes 和 data items 的差异。具体有两步：

S1.通过 PCA 将数据降到 M 维度，$\mathbf{v} = \mathbf{P}^T \mathbf{x}$;

S2.通过求解下面的最优化问题，求解最优的 hash codes 和最优旋转 **R**

$$\min \|\mathbf{Y} - \mathbf{R}^T \mathbf{V}\|_F^2, \text{ where } \mathbf{V} = [\mathbf{v}_1 \mathbf{v}_2 \cdots \mathbf{v}_N] \text{ and } \mathbf{Y} = [\mathbf{y}_1 \mathbf{y}_2 \cdots \mathbf{y}_N].$$

其中，S2 通过两步迭代求解来完成。具体为：

首先，固定 R，有 Y=sign(R'V)；然后，固定 Y，然后就是求解一个经典的 Procrustes 正交问题，其解为：$\mathbf{R} = \hat{\mathbf{S}}\mathbf{S}^T$。这儿$\mathbf{S} \text{ and } \hat{\mathbf{S}}$通过 $\mathbf{Y}\mathbf{V}^T$的 SVD 获得，即$\mathbf{Y}\mathbf{V}^T = \mathbf{S}\mathbf{\Lambda}\hat{\mathbf{S}}^T$。

6.2.2 各向同性的哈希

目的是通过旋转空间，从而使得在每个维度的方差都是相同的。具体有三步：

S1.通过 PCA 将数据降到 M 维度，$\mathbf{v} = \mathbf{P}^T \mathbf{x}$;

S2.找到最优的旋转矩阵 **R**,从而使得$\mathbf{R}^T\mathbf{V}\mathbf{V}^T\mathbf{R} = \mathbf{\Sigma}$为一个具有相同对角值得矩阵,即$[\mathbf{\Sigma}]_{11} = \cdots = [\mathbf{\Sigma}]_{MM}$。

令$\sigma = \frac{1}{M} \text{Trace}\, \mathbf{V}\mathbf{V}^T$，我们可以用下式来寻找最优的旋转 **R**:

$\|\mathbf{R}^T\mathbf{V}\mathbf{V}^T\mathbf{R} - \mathbf{Z}\|_F = 0$，其中 Z 为对角线上值均为$\sigma$的对角矩阵。

上述问题可以通过两个算法解决：Lift and projection and gradient flow.

事实上，通过旋转空间，从而使得在每个维度的方差都是相同的，是为了 hash codes 的 bits 对后续的距离评估具有相同的贡献。

6.2.3 协调的哈希

可以看作 ITQ 和 Isotropic hashing 的结合。其目标函数为：

$$\min_{\mathbf{Y},\mathbf{R}} \|\mathbf{Y} - \mathbf{R}^T\mathbf{V}\|_F^2 \quad \text{s.t.} \quad \begin{matrix} \mathbf{Y}\mathbf{Y}^T = \sigma\mathbf{I} \\ \mathbf{R}^T\mathbf{R} = \mathbf{I}. \end{matrix}$$

特别的，与 ITQ 不同，这儿不要求 Y 是二值得。同样，可以用迭代算法求解上式：

首先，固定 **R**，有$\mathbf{R}^T\mathbf{V} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T$，进而$\mathbf{Y} = \sigma^{1/2}\mathbf{U}\mathbf{V}^T$；然后固定 **Y**,有$\mathbf{R} = \hat{\mathbf{S}}\mathbf{S}^T$，同样$\mathbf{Y}\mathbf{V}^T = \mathbf{S}\mathbf{\Lambda}\hat{\mathbf{S}}^T$。最后，我们通过零阈值，获得 **Y** 的二值编码。

6.2.4 角度分层化

针对 cosine 相似度。基本思想是利用最近的超立方体的节点$\{0,1\}^d$来近似数据向量 x：

$$\arg\max_{\mathbf{y}} \frac{\mathbf{y}^T\mathbf{x}}{\|\mathbf{b}\|_2}, \text{ subject to } \mathbf{y} \in \{0,1\}^d$$

这儿，找到二值编码的方法和迭代分层一样，目标函数如下：

$$\max_{\mathbf{R},\{\mathbf{y}_n\}} \sum_{n=1}^{N} \frac{\mathbf{y}_n^T}{\|\mathbf{y}_n\|_2} \frac{\mathbf{R}^T\mathbf{x}_n}{\|\mathbf{R}^T\mathbf{x}_n\|_2} \quad \text{s.t.} \quad \mathbf{y}_n \in \{0,1\}^M, \ \mathbf{R}^T\mathbf{R} = \mathbf{I}_M.$$

其中 R 是 dxM 的投影矩阵。通过丢掉分母$\|\mathbf{R}^T\mathbf{x}_n\|_2$，我们就可以通过求解简单的求解下式获得 code

$$\max_{\mathbf{R},\{\mathbf{y}_n\}} \sum_{n=1}^{N} \frac{\mathbf{y}_n^T}{\|\mathbf{y}_n\|_2} \mathbf{R}^T\mathbf{x}_n \quad \text{s.t.} \quad \mathbf{y}_n \in \{0,1\}^M, \ \mathbf{R}^T\mathbf{R} = \mathbf{I}_M.$$

6.3 笛卡尔分层

6.3.1 乘积分层

思想：把特征空间划分为 P 个不重叠的子空间。因此，数据集 X 可以划分为 P 个子集，每个子集包含 N 个自向量 $\{\mathbf{x}_{p1},\cdots,\mathbf{x}_{pN}\}$，并且，我们把每个子空间聚成 K 类，故有聚类中心$\{\mathbf{c}_{p1},\mathbf{c}_{p2},\cdots,\mathbf{c}_{pK}\}$。其中，每个聚类中心因此可以被编码为 log2K。

因为每个数据 item $\mathbf{x}_n$ 被分割成 P 个自向量$\{\mathbf{x}_{pn}\}$，并且每个 p 子空间的子向量都被分配到最近的一个聚类中心 $\mathbf{c}_{pk_{pn}}$。然后数据 item $\mathbf{x}_n$ 就用这 P 个自向量的集合表示：$\{\mathbf{c}_{pk_{pn}}\}_{p=1}^{P}$，获得的编码长度为$P\log_2 K$。如果用数学来表示，就是如下的最小化问题：

$$\min_{\mathbf{C},\{\mathbf{b}_n\}} \sum_{n=1}^{N} \|\mathbf{x}_n - \mathbf{C}\mathbf{b}_n\|_2^2$$

Here $\mathbf{C}$ is a matrix of $d \times PK$ in the form of

$$\text{diag}(\mathbf{C}_1,\mathbf{C}_2,\cdots,\mathbf{C}_P) = \begin{bmatrix} \mathbf{C}_1 & 0 & \cdots & 0 \\ 0 & \mathbf{C}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{C}_P \end{bmatrix} \text{where } \mathbf{C}_p = [\mathbf{c}_{p1}\mathbf{c}_{p2}\cdots\mathbf{c}_{pK}]$$

并且 *bn* 是指示向量，其中 *bnp* 是一个长度为 K 的向量，且只有其中一个被选中的聚类中心的位置值为 1.

给定查询数据 *xt*，其到 *xn* 向量$k_{1n}k_{2n}\cdots k_{Pn}$的距离可以通过对称和非对称的方式进行评估。

对称距离：首先，对查询 item xt 进行编码获得哈希码$k_{1t}k_{2t}\cdots k_{Pt}$。然后，计算一个距离表。这个距离表由 *PK* 个输入，即$\{d_{pk} = \|\mathbf{c}_{pk_{pt}} - \mathbf{c}_{pk}\|_2^2 | p = 1,\cdots,P, k = 1,\cdots,K\}$。然后通过查表，并求获得的 *P* 个距离，$\sum_{p=1}^{P} d_{pk_{pn}}$。

非对称距离：不进行输入向量编码，直接计算有 *PK* 个项的距离表。然后通过查表，求和 *P* 个距离项。

6.3.2 笛卡尔 K-means

笛卡尔 K-means 扩展了乘积分层，并且引入了旋转项 R 到目标函数中：

$$\min_{\mathbf{R},\mathbf{C},\{\mathbf{b}_n\}} \sum_{n=1}^{N} \|\mathbf{R}^T\mathbf{x}_n - \mathbf{C}\mathbf{b}_n\|_2^2$$

问题通过迭代的方法求解 $\mathbf{C}$, $\{\mathbf{b}_n\}$, and $\mathbf{R}$。首先，固定 **R**，**C** 和{**b***n*}可以通过和类此成绩分层的方法求解。然后固定 **C** 和{**b***n*}，求解 **R** 就是一个经典的正交乘积问题。

同样，数据向量 xn 也可以表示为 P 个子向量：$\{\mathbf{c}_{pk_{pn}}\}_{p=1}^{P}$，编码为$k_{1n}k_{2n}\cdots k_{Pn}$（伴随一个针对所有数据的旋转矩阵 **R**）。而对于给定查询 xt，首先计算它的旋转$\mathbf{R}^T\mathbf{x}_t$，之后的距离计算和之前的乘积分层相同。

6.3.3 复合分层

过程分为两步：S1.用有 d 维度的 P 个向量来近似向量 xn，即$\mathbf{c}_{1k_{1n}},\mathbf{c}_{1k_{2n}},\cdots,\mathbf{c}_{1k_{Pn}}$，其中每一个都是

从子空间的字典$\{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_P\}$中抽取的一个聚类中心；S2.计算所有用来近似向量 xn 的，来自不同字典的元素对内积之和，即$\sum_{i=1}^{P} \sum_{j=1, \neq i}^{P} \mathbf{c}_{ik_{in}} \mathbf{c}_{jk_{jn}}$，并要求其对所有样本 item 均为一个常数$\epsilon$。具体问题转化为：

$$\begin{aligned} \min_{\{\mathbf{C}_p\},\{\mathbf{b}_n\},\epsilon} \quad & \sum_{n=1}^{N} \|\mathbf{x}_n - [\mathbf{C}_1 \mathbf{C}_2 \cdots \mathbf{C}_P]\mathbf{b}_n\|_2^2 \\ \text{s.t.} \quad & \sum_{i=1}^{P} \sum_{j=1, j\neq i}^{P} \mathbf{b}_{ni}^T \mathbf{C}_i^T \mathbf{C}_j \mathbf{b}_{nj} = \epsilon \\ & \mathbf{b}_n = [\mathbf{b}_{n1}^T \mathbf{b}_{n2}^T \cdots \mathbf{b}_{nP}^T]^T \\ & \mathbf{b}_{np} \in \{0,1\}^K, \ \|\mathbf{b}_{np}\|_1 = 1 \\ & n = 1, 2, \cdots, N, p = 1, 2, \cdots P. \end{aligned}$$

Here, $\mathbf{C}_p$ is a matrix of size $d \times K$, and each column corresponds to an element of the pth dictionary $\mathcal{C}_p$.

为了获得最优解，上述问题可以转化为：

$$\phi(\{\mathbf{C}_p\}, \{\mathbf{b}_n\}, \epsilon) = \sum_{n=1}^{N} \|\mathbf{x}_n - \mathbf{C}\mathbf{b}_n\|_2^2 + \mu \sum_{n=1}^{N} (\sum_{i\neq j}^{P} \mathbf{b}_{ni}^T \mathbf{C}_i^T \mathbf{C}_j \mathbf{b}_{nj} - \epsilon)^2$$

其中 u 是惩罚系数，$\mathbf{C} = [\mathbf{C}_1 \mathbf{C}_2 \cdots \mathbf{C}_P]$，$\sum_{i\neq j}^{P} = \sum_{i=1}^{P} \sum_{j=1, j\neq i}^{P}$。问题可以迭代求解。

7. 哈希学习：其他问题

7.1 多表哈希

7.1.1 互补哈希

目的：学习多个 hash tables 以使得相邻的点能以大概率出现在至少一个相同的哈希桶中。

算法通过一个接一个的方式学习到多个 hash tables。其中，第一张表通过解决下面问题实现：

$$\text{trace}[\mathbf{W}^T \mathbf{X}_l \mathbf{S} \mathbf{X}_l^T \mathbf{W}] + \eta \, \text{trace}[\mathbf{W}^T \mathbf{X}\mathbf{X}^T \mathbf{W}]$$

where s_{ij} is initialized as $K(a_{ij} - \alpha)$, a_{ij} is the similarity between $\mathbf{x}_i$ and $\mathbf{x}_i$ and α is a super-constant.

为了计算第二部分的 hash functions，同样是用上面的目标函数，但是相似度矩阵 S 的定义变化为：

$$s_{ij}^t = \begin{cases} 0 & b_{ij}^a = b_{ij}^{(t-1)} \\ \min(s_{ij}, f_{ij}) & b_{ij}^a = 1, b_{ij}^{(t-1)} = -1 \\ -\min(-s_{ij}, f_{ij}) & b_{ij}^a = -1, b_{ij}^{(t-1)} = 1 \end{cases}$$

where $f_{ij} = (a_{ij} - \alpha)(\frac{1}{4} d_h^{(t-1)}(\mathbf{x}_i, \mathbf{x}_j) - \beta)$, β is a super-constant, and $b^{(t-1)ij} = 1 - 2\,\text{sign}[\frac{1}{4} d_h^{(t-1)}(\mathbf{x}_i, \mathbf{x}_j) - \beta]$.

7.1.2 互惠哈希

互惠哈希扩展了互补哈希，通过在所有哈希函数的池 B 上建立 graph；然后在搜索过程中，在这个 graph 上搜索最佳的哈希函数来建立哈希表。

7.2 主动和在线哈希

7.2.1 主动哈希

开始于一个小的，有标签信息的数据点对集合，然后主动的选择那些最有信息量的有标签数据来进行哈希函数的学习。

给定标签数据集合 L，无标签数据集合 U，候选点集合 C，算法首先学习紧致的哈希函数$\mathbf{h} = \text{sign}(\mathbf{W}^T \mathbf{x})$，然后计算候选点集合中的数据点的确定性分数$f(\mathbf{x}) = \|\mathbf{W}^T \mathbf{x}\|_2$（反映数据点到哈希函数组成的超平面的距离）。接着，其中确定性分数小的被选出来进行进一步标定。同时，要确保备选出来的点相互之间应该不能相似。总体而言，选择最优信息点的目标函数如下：

$$\min_{\mathbf{b}} \quad \mathbf{b}^T \bar{\mathbf{f}} + \frac{\lambda}{M} \mathbf{b}^T \mathbf{K} \mathbf{b} \quad \text{s.t.} \quad \begin{aligned} & \mathbf{b} \in \{0,1\}^{\|\mathcal{C}\|} \\ & \|\mathbf{b}^T\|_1 = M, \end{aligned}$$

其中 bi 是指示函数，bi=1 表示选中 xi 进行后续标定。M 是需要被选的样本点总数。$\bar{\mathbf{f}}$是候选点集合归一化的确定性分数，**K** 是候选点集合的相似度矩阵。

7.2.2 在线哈希

当相似/不相似的点对是一个接一个来，而不是一起来时，可以进行在线哈希。

7.3 绝对内积相似度的哈希

7.3.1 伴随哈希

目的是找到那些最小和最大绝对值 cosine 相似度的数据点。和伴随的 LSH 类似，伴随哈希也是用多个集合来进行二比特哈希编：$\{h_{min}(\mathbf{x}), h_{max}(\mathbf{x})\} = \{\arg\min_{k=1}^{2^K} \mathbf{w}_k^T\mathbf{x}, \arg\max_{k=1}^{2^K} \mathbf{w}_k^T\mathbf{x}\}$。同时，碰撞概率定义为：$\text{Prob}[\{h_{min}(\mathbf{x}), h_{max}(\mathbf{x})\} = \{h_{min}(\mathbf{y}), h_{max}(\mathbf{y})\}]$，它是$|\mathbf{x}_1^T\mathbf{x}_2|$的单调递增函数。也就是说，汉明距离越大，$|\mathbf{x}_1^T\mathbf{x}_2|$越小。反之亦然。

7.4 矩阵哈希

7.4.1 双线性投影

目的是将矩阵形式的特征哈希到短的 codes。其哈希函数定义为：

$$\text{vec}(\text{sign}(\mathbf{R}_l^T\mathbf{X}\mathbf{R}_r))$$

其中，X 是 dlxdr 的矩阵，Rl（dlxdl）和 Rr（drxdr）是两个随机正交矩阵。同时，很容易获得：

$$\text{vec}(\mathbf{R}_l^T\mathbf{X}\mathbf{R}_r) = (\mathbf{R}_r^T \otimes \mathbf{R}_l^T)\,\text{vec}(\mathbf{X}) = \mathbf{R}^T\,\text{vec}(\mathbf{X})$$

目标函数是去最小化旋转特征$\mathbf{R}^T\,\text{vec}(\mathbf{X})$和其二值编码$\text{sign}(\mathbf{R}^T\,\text{vec}(\mathbf{X})) = \text{vec}(\text{sign}(\mathbf{R}_l^T\mathbf{X}\mathbf{R}_r))$之间的 angle。具体函数为：

$$\max_{\mathbf{R}_l,\mathbf{R}_r,\{\mathbf{B}_n\}} \sum_{n=1}^{N} \text{trace}(\mathbf{B}_n\mathbf{R}_r^T\mathbf{X}_n^T\mathbf{R}_l) \quad \text{s.t.} \quad \mathbf{B}_n \in \{-1,+1\}^{d_l\times d_r},\ \mathbf{R}_l^T\mathbf{R}_l = \mathbf{I},\ \mathbf{R}_r^T\mathbf{R}_r = \mathbf{I},$$

其中，$\mathbf{B}_n = \text{sign}(\mathbf{R}_l^T\mathbf{X}_n\mathbf{R}_r)$。问题可以通过$\{\mathbf{B}_n\}$, $\mathbf{R}_l$ and $\mathbf{R}_r$之间的迭代求解来完成。同时，为了降低编码长度，同意用一个低维度的正交矩阵进行处理，即$\mathbf{R}_l \in \mathbb{R}^{d_l\times c_l}$ and $\mathbf{R}_r \in \mathbb{R}^{d_r\times c_r}$。

7.5 紧致稀疏编码

利用稀疏编码来代表数据的 items：非零 codes 的 atoms 被用来建立倒排索引，非零系数被用来重构数据 items，及用来近似计算查询 item 和数据集 items 的距离。

稀疏编码的目标函数为：

$$\min_{\mathbf{C},\{\mathbf{z}_n\}_{n=1}^N} \frac{1}{2}\sum_{i=1}^{N}\|\mathbf{x}_i - \sum_{j=1}^{K} z_{ij}\mathbf{c}_j\|_2^2 + \lambda\|\mathbf{z}_i\|_1 \quad \text{s.t.} \quad \|\mathbf{C}_{\sim k}^T\mathbf{c}_k\|_\infty \leqslant \gamma; k = 1,2,\cdots,n$$

其中 C 是字典，C~k 是移除第 k 个 atom 后的字典，zn 是稀疏编码，$\|\mathbf{C}_{\sim k}^T\mathbf{c}_k\|_\infty \leqslant \gamma$用来控制字典的一致性程度。

xn 的支持定义为 zn 中非零系数的索引，即$\mathbf{b}_n = \delta.[\mathbf{z}_n \neq 0]$, where $\delta.[]$ is an element-wise operation.。在这个方法里，利用$\{\mathbf{b}_n\}_{n=1}^N$来建立倒排索引，同时利用杰卡德相似度来进行搜索。最后，基于查询的对称距离和杰卡德相似度的检索结果，计算$\|\mathbf{q}-\mathbf{B}\mathbf{z}_n\|_2$来进行排序。

7.6 汉明空间中的快速搜索

7.6.1 多索引哈希

数据集的二值编码，基于 M 个不耦合的子字符串，通过 M 次索引到 M 个不同的哈希表中。给定一个查询二值编码，那些至少一次接近查询样本的子字符串一次的样本被认为是最近邻候选点。

特别地，每一个哈希码 **y** 被分成 M 个不耦合的子码{y1,…,yM}。对于每一个字码 ym，我们都相应的建

立一个哈希表。其中第 m 个表的每个输入都与一些列二值码（指那些第 m 个子码所对应的二值码）的索引相对应。

为了找到查询 q= $\{\mathbf{q}^m\}_{m=1}^M$的 RNN，算法查找那些表中汉明距离在$\lfloor \frac{R}{M} \rfloor$ of $\mathbf{q}^m$之内的，并选择出来作为候选集合，即$\mathcal{N}_m(\mathbf{q})$。最终有，$\mathcal{N} = \cup_{m=1}^M \mathcal{N}_m(\mathbf{q})$。最后，算法计算这些候选点和查询 q 之间汉明距离，并保留那些真正属于 RNN 的样本。

7.6.2 FLANN

关键在于构建多个分层的聚类树来组织二值向量，并同时在这多个树上进行最近邻搜索。

树的构建过程为：对所有点分割到 K 个聚类中心（从 items 中随机选择），并给每个点指派一个聚类中心。然后在每个聚类族中重复进行，知道每个聚类中的数目小于一个阈值。

树的搜索过程为：对每个 tree 进行一次贯穿，在这个过程中，算法会选择那些离查询样本最近的节点，并进一步挖掘这个节点下的子树。到到达叶子节点后，回溯，找到满足最近邻数量的子树。

8. 讨论、趋势、总结

8.1 大规模哈希函数的学习

很多时候对于大数据集，会抽样一部分样本来进行哈希函数的学习，但这也会在一定程度上降低精度。因此，对于大数据集，其哈希函数学习仍需进一步研究。

8.2 哈希编码的计算加速

关于加速哈希算法的编码计算的工作需要进一步研究。

8.3 距离表的计算加速

乘积分层及其变体等都需要预先计算一个查询样本和字典的中元素的距离表。但是，在实际过程中，当用分层方法来计算 code 来进行倒排索引中检索样本排序过程中，这个过程的消耗比较大。因此，距离表的计算也是一个很有意义的研究领域。

8.4 多/交叉模态的哈希

当大数据中的数据是多类型和多数据源时，利用多模态数据之间的相关性，来进行多模态的哈希学习也变得很重要。

8.5 总结

局部敏感哈希，学习哈希，基于它们的最近邻搜索。