M2LSH:基于LSH的高维数据近似最近邻查找算法

李 灿,钱江波,董一鸿,陈华辉
(宁波大学信息科学与工程学院,浙江宁波 315211)

摘 要: 在许多应用中,LSH(Locality Sensitive Hashing)以及各种变体,是解决近似最近邻问题的有效算法之一.虽然这些算法能够很好地处理分布比较均匀的高维数据,但从设计方案来看,都没有针对数据分布不均匀的情况做相应的优化.针对这一问题,本文提出了一种新的基于LSH的解决方案(M2LSH,2 Layers Merging LSH),对于数据分布不均匀的情况依然能得到一个比较好的查询效果.首先,将数据存放到具有计数功能的组合哈希向量表示的哈希桶中,然后通过二次哈希将这些桶号投影到一维空间,在此空间根据各个桶中存放的数据个数合并相邻哈希桶,使得新哈希桶中的数据量能够大致均衡.查询时仅访问有限个哈希桶,就能找到较优结果.本文给出了详细的理论分析,并通过实验验证了M2LSH的性能,不仅能减少访问时间,也可提高结果的正确率.

关键词: 近似最近邻; KNN查询; 局部敏感哈希; 高维数据
中图分类号: TP311.3 **文献标识码:** A **文章编号:** 0372-2112 (2017)06-1431-12
电子学报URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2017.06.022

M2LSH:An LSH Based Technique for Approximate Nearest Neighbor Searching on High Dimensional Data

LI Can,QIAN Jiang-bo,DONG Yi-hong,CHEN Hua-hui
(*Faculty Electrical Engineering and Computer Science,Ningbo University,Ningbo,Zhejiang* 315211,*China*)

Abstract: The LSH (Locality Sensitive Hashing) method and its variants represent the state-of-the-art indexing techniques to process ANN (Approximate Nearest Neighbor) searches in a high dimensional data space. Although the existing LSH based techniques can efficiently deal with uniform distributed data, it is noticed from the principle of their design schemes that these techniques cannot handle non-uniform distributed data well. To tackle the above problem, this paper presents a new LSH based technique, called M2LSH (2 Layers Merging LSH), to efficiently process ANN searches on non-uniform distributed data. The key idea is as follows. First, data objects are stored in a hash table with multiple buckets (i.e., the first layer), each of which corresponds to a combined hash vector used as its hash number. The count of data objects in each hash bucket is recorded. Secondly, using the hash functions for a second layer hash table, the bucket numbers from the first layer are projected into a one-dimensional space. In this space, some adjacent hash buckets from the first layer are merged so as to make the data objects uniformly distributed in the merged buckets at the second layer. Therefore, the M2LSH can not only improve the searching efficiency but also increase the accuracy of the search results. This paper also provides a detailed theoretical accuracy analysis for the M2LSH technique. Experiments using synthetic and real data show that the theoretical estimates are quite accurate, and the proposed M2LSH technique can efficiently process ANN searches with low false positive and negative rates.

Key words: approximate nearest neighbor (ANN); k-nearest neighbor (KNN) search; locality sensitive hashing (LSH); high dimensionality

---

收稿日期:2015-10-07;修回日期:2016-01-19;责任编辑:梅志强
基金项目:国家自然科学基金(No.61472194,No.61572266);浙江省自然科学基金(No.LY13F020040,No.LY16F020003);宁波市自然科学基金(No.2014A610023,No.2015A610119)

1 引言

最近邻查询问题(nearest neighbor search problem)指在给定数据集中返回与查询对象距离最近的数据对象的问题.最近邻问题在不同领域都有广泛的应用,如:人工智能、信息检索、模式识别等.形式化地,最近邻查询指在给定有 n 个 d 维数据对象的数据集 D 中找到与查询对象 $\boldsymbol{q}$ 距离最接近的数据对象 $\boldsymbol{o}^*$ 使得:$\forall \boldsymbol{o} \in D-\{o^*\}$,$\|\boldsymbol{q},\boldsymbol{o}\|_p \geqslant \|\boldsymbol{q},\boldsymbol{o}^*\|_p$,其中 $\|\cdot,\cdot\|_p$ 是给定的在 d 维空间中的距离度量 l_p 范式.常见的距离度量有 l_1 范式:$\|\boldsymbol{q},\boldsymbol{o}\|_1 = \sum_{i=1}^{d} |x_i - y_i|$;$l_2$ 范式:$\|\boldsymbol{q},\boldsymbol{o}\|_2 = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$,其中数据对象 $\boldsymbol{o}=(x_1,\cdots,x_d)$,查询对象 $\boldsymbol{q}=(y_1,\cdots,y_d)$.

最近邻查询问题目前已经有很多精确的解决方法[1].如基于空间划分的索引技术,如 R-tree[2]、KD-tree[3]、VP-tree[4]、CU-tree[5]和 SR-tree[6]等方法,对于给定的查询对象 $\boldsymbol{q}$,都能返回精确的 $\boldsymbol{q}$ 的最近邻对象.然而,随着数据对象的维度的升高,"维度灾难"会伴随着出现,使得这些基于空间划分方法的效率急剧下降.当数据维度超过 10 的时候,这些方法的效率甚至比最简单暴力的线性扫描的方法更低.换句话说,基于空间划分的索引技术并不适用于高维数据[7].而基于聚类的索引技术如:IQ-tree[8],iDistance[9]和 HashFile[10]等方法,虽然可以避免维度灾难,但在高维的数据集中仍不能快速高效地找到目标所在的聚类,并且这个过程将成为查询的性能瓶颈.

由于高维空间的最近邻查询比较困难,学术界提出 c-近似最近邻查询问题(c-approximate nearest neighbor search problem).c-近似最近邻查询问题的目标是根据查询点 $\boldsymbol{q}$ 返回数据对象 $\boldsymbol{o}^{\#}$,满足:$\|\boldsymbol{q},\boldsymbol{o}^{\#}\|_p \leqslant c\|\boldsymbol{q},\boldsymbol{o}^*\|_p$,其中 $\boldsymbol{o}^*$ 是查询对象 $\boldsymbol{q}$ 的真实最近邻.而在这方面的算法中,基于距离敏感哈希(Locality-Sensitive Hashing,LSH)的算法具有代表性.虽然 LSH 算法在解决高维空间中近邻问题上取得了非常瞩目的成就,但在处理高维数据时却很少考虑分布不均的情况.在 LSH 算法的基础上,本文针对高维数据分布不均的情况,提出了一种新的解决解决方案.

2 相关工作

ANN 问题的研究成果可以分为三类:降维、编码以及 LSH.

基于降维的方法是先将高维的数据映射到一个较低维度的空间,然后根据已有的低维解决 KNN 问题的一些方法来建立索引结构[2~8].这些索引结构往往选取空间树形结构来构建,如前文讲到的 R-tree、KD-tree、VP-tree、CU-Tree 等.

基于编码的方法在涉及计算机视觉领域的研究比较活跃[11].这些方法大多假设内存足够储存数据集和索引.将一个高维数据重新编码成一个位串,然后通过计算这些位串与查询对象之间的 AQD(Asymmetric Quantizer Distance),返回与查询对象 AQD 最小的数据作为结果.

在高维空间中,LSH 和它的变体是解决 c-近似最近邻查询问题的有效方法,具有查询的高效性及较低错误率.LSH 方法最初是由 Indyk 等人[12,13]提出,用以解决海明空间的 c-近似的最近邻问题.后来,Datar 等通过 p-稳态分布(p-stable Distribution)的性质,拓展到欧几里德空间.LSH 方案利用一系列具有"位置敏感"性质的哈希函数,把在空间中"比较接近"的数据对象以较高的概率映射到相同的哈希桶中.目前,最常见的欧几里得空间的 LSH 函数是把数据对象投影到随机的一维空间,该空间被划分成多个宽度为 w 的区间,每个区间可看作为一个哈希桶.如果两个数据对象被投影到同一区间,那么就认为这两个数据对象"碰撞"了,即认为这两个对象在原始空间中的位置也比较接近.

基本 LSH 的空间开销比较大.为解决这一问题,学者们提出许多 LSH 变体,如 E2LSH[14]、Entroy-based LSH[15]、Multi-Probe LSH[16]、LSB[17]、C2LSH[18]、SK-LSH[19]、MLBF[20]等.其中,E2LSH 将 LSH 算法成功运用到基于 p-稳定分布的高维数据中,并且提出了 AND-OR 机制来提高查询正确率,但它需要耗费较大的存储空间来存储索引文件.Entroy-based LSH 是通过熵的概念来产生与查询点满足一定条件的虚拟点,将这些虚拟点找到的近似点也作为原始查询点的候选集.Multi-Probe LSH 是在查找的过程中通过一系列的随机偏置来探测与查询桶临近的桶,将这些桶中的数据也作为查询候选集.LSB 是为了高效处理硬盘上的数据而设计的,可以减小硬盘 I/O.C2LSH 提高了 LSH 的效率和准确性,但在该算法中需要为每一个数据集中的对象维护一个碰撞计数器,对于一些比较大的数据集,这种操作对于内存消耗比较大.SK-LSH 定义了一种距离判定方式,并将组合哈希桶号建成一个树形结构,在减小硬盘 I/O 的同时也提高了查询的准确率.MLBF 则成功地将 LSH 应用到了布隆过滤器上,并首次提出多粒度查询的概念,进一步地扩充了 LSH 的应用领域.此外,还有一些基于机器学习的哈希方法,但是它们都需要较大的训练开销.

3 基础知识

令 $\boldsymbol{q} \in \mathfrak{B}(\boldsymbol{o},s)=\{\boldsymbol{q} \in R^d, \|\boldsymbol{o},\boldsymbol{q}\| \leqslant s\}$ 记作以数据对

象 $\boldsymbol{o}\in D$ 为中心,半径为 s 的超球体. 因此,LSH 函数家族的形式化定义[13]如下:

定义 1　一个 LSH 函数家族 $H=\{h:R^d\to U\}$ 定义为在给定测量范式 l_p 下 (s,cs,p_1,p_2)-敏感表示对任意数据对象 $\boldsymbol{o},\boldsymbol{q}\in R^d$ 满足:

· 如果 $\boldsymbol{o}\in\mathfrak{B}(\boldsymbol{q},s)$,即 $\|\boldsymbol{o},\boldsymbol{q}\|\leqslant s$,那么 $P[h(\boldsymbol{o})=h(\boldsymbol{q})]\geqslant p_1$;

· 如果 $o\notin\mathfrak{B}(\boldsymbol{q},cs)$,即 $\|\boldsymbol{o},\boldsymbol{q}\|\geqslant cs$,那么 $P[h(\boldsymbol{o})=h(\boldsymbol{q})]\leqslant p_2$.

其中,$P[h(\boldsymbol{o})=h(\boldsymbol{q})]$表示对象 $\boldsymbol{o}$ 和 $\boldsymbol{q}$ 落入相同桶中的概率,$c>1$ 和 $p_1>p_2$. 此外,组合哈希函数表示成 $g=(h_1,\cdots,h_k)$,其中 $h_1,\cdots,h_k$ 是从 LSH 哈希函数族中随机抽取的 k 个距离敏感函数. 根据文献[18],一个用以构造适用于 l_2 距离度量的 LSH 函数家族的形式如下:

$$h_{\boldsymbol{a},b}(\boldsymbol{o})=\left\lfloor\frac{\boldsymbol{a}\cdot\boldsymbol{o}+b}{w}\right\rfloor \tag{1}$$

其中,$\boldsymbol{o}$ 记作数据对象 $\boldsymbol{o}\in R^d$ 的向量表示,$\boldsymbol{a}$ 是一个每个坐标元素都是独立随机地从标准正态分布 $N(0,1)$ 中抽取的 d 维向量,w 是一个用户指定的区间宽度常数,b 是一个随机地从均匀分布$[0,w)$中抽取的实数.

对于任意两个数据对象 $\boldsymbol{o}_1$ 和 $\boldsymbol{o}_2$,令 $s=\|\boldsymbol{o}_1,\boldsymbol{o}_2\|$,则 $\boldsymbol{o}_1$ 和 $\boldsymbol{o}_2$ 在随机从 LSH 函数家族中抽取的函数 $h_{\boldsymbol{a},b}$ 作用下产生碰撞的概率 $p(s)$ 可以通过以下计算得到[14]:

$$\begin{aligned}p(s)&=P[h_{\boldsymbol{a},b}(\boldsymbol{o}_1)=h_{\boldsymbol{a},b}(\boldsymbol{o}_2)]\\&=\int_0^w\frac{1}{s}f\left(\frac{t}{s}\right)\left(1-\frac{t}{w}\right)\mathrm{d}t\end{aligned} \tag{2}$$

其中 $f(x)=\dfrac{2}{\sqrt{2\pi}}\mathrm{e}^{\frac{-x^2}{2}}$.

4　M2LSH 算法

现实应用中需处理的数据的分布往往不是均匀分布的,这使得现有 LSH 算法在查询稳定性上会有一些折扣. 因为数据分布不均将导致索引文件中有的哈希桶保存的数据多,而有的哈希桶很少. 查询的时候,有的查询就会返回过多的候选结果,而有的查询返回的候选结果又过少. 这样就不能够保证查询的稳定性. 对于这种情况一个行之有效的方法是在构造索引的时候使每个哈希桶中的数据量能大致均衡,这样即使对于非均匀分布的数据集,LSH 算法也能有一个很好的稳定性.

为此,我们提出一种新的 LSH 构建方式,在保证查询正确率的前提下尽可能地减少候选集,不仅可以提高查询准确率也可以提高查询速度,提升整体性能.

在图 1 展示了在单个哈希表中建立索引的过程. 首先,需要将原始数据存放到以组合哈希向量表示的哈希桶中. 如对象 $\boldsymbol{o}$,经过组合哈希后会被存放到 $g_i(\boldsymbol{o})=\langle h_1(\boldsymbol{o}),h_2(\boldsymbol{o}),\cdots,h_{k_1}(\boldsymbol{o})\rangle$ 对应的哈希桶中. 算法采用的桶宽 w 比较小,并且对每个组合哈希桶都有一个计数器 Counter,用以记录落在该桶中的对象数. 下面说明索引结构的构建过程和查询方式:

(1)建立索引阶段:

① 将数据都投影到以组合哈希向量表示的桶中. 每个数据对象 $\boldsymbol{o}$ 经过一个 $g_i(\cdot)$ 都会生成$(h_1(\boldsymbol{o}),h_2(\boldsymbol{o}),\cdots,h_{k_1}(\boldsymbol{o}),1)$的形式,其中$(h_1(\boldsymbol{o}),h_2(\boldsymbol{o}),\cdots,h_{k_1}(\boldsymbol{o}))$对应为对象 $\boldsymbol{o}$ 经过 AND 过程后应该放入的组合哈希桶号,用 $\boldsymbol{o}'$来表示. 1 用于对该桶号中数据数目进行计数.

② 将第一步得到的组合哈希桶号进行二次哈希,将其投影到一条线上,并准备合并. 通过对组合哈希桶号再次哈希,使这些桶号在一维空间上重新排序,并使得相邻的哈希桶号被哈希到相邻的位置.

③ 对二次哈希后组合哈希向量表示的桶号进行桶合并. 这一步通过将需合并的桶打上相同的标记来实现. 在合并哈希桶的过程中应该考虑一些因素,包括:桶中存放数据的平均个数 average count (用 AC 表示)、被哈希到相邻位置的哈希桶号之间的距离、以及当前哈希桶中保存数据的个数. 例如 $\boldsymbol{o}'_1(\langle1,2,4,5\rangle,10)$,$\boldsymbol{o}'_2(\langle1,1,4,5\rangle,5)$,而 AC 为 50,如果这两个桶对象,被哈希到了相邻的位置,会被打上相同的标记. 如果两个相邻的桶,两个桶中的数据都很多(远大于 AC)就没有合并的必要. 即使两个桶中的数据都很少(小于 AC),而这两个相邻的桶的真实距离却比较远,大于设定的一个阈值 ρ,则不会将它们打上相同的标记.

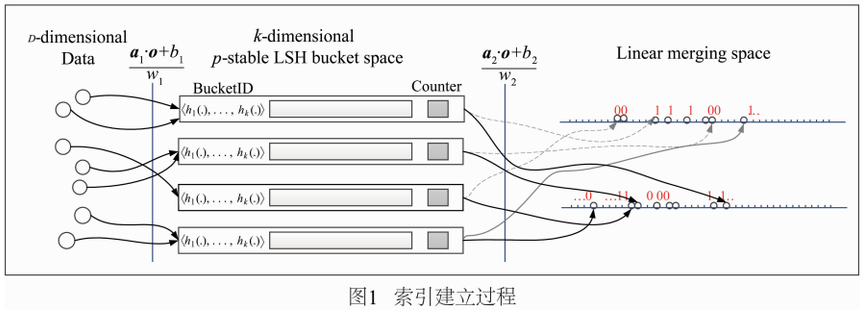


图1　索引建立过程

建立索引算法的实现见算法 1

算法 1 BuildIndex(L,k_1,k_2,w)

```
1 Input D (set of object o)
2 L  (number of hash tables)
3 k1  (number of first layer hashes)
4 k2  (number of Second layer hashes)
5 w  (width of bucket)
6 Output Hash Index (stored information of each table gi,i=1,…,L)
7 Foreach i=1,…,L
8     Initialize hash function g1 by generating k1 random hash functions
9     Generates k2 random hash function as the second layer.
10 Foreach i=1,…,L
11   Foreach j=1,…,n1(the number of the dataset D)
12       Store object oj in bucket gi(·)
13       and counter_{gi(oi)}
14   Foreach j=1,…,n2(the number of the dataset of (gi(·),
     counter_{gi(.)}))
15       Projected oj onto k2 lines
16   Foreach j=1,…,k2
17       Call Bucket merger(line_j) by marking bucket with 0 or 1
```

该算法描述了建立索引算法实现的过程. 从伪代码中第 7 行开始分别构建 $i=1,\cdots,L$ 个 $g(\cdot)$,每一个 $g(\cdot)$对应为一个索引文件. 从第 11 行开始,首先,将数据存放到以组合哈希向量表示的哈希桶中,并且对当前哈希桶中已经存放的数据个数进行计数. 接下来从第 17 行开始,对这些哈希桶号进行二次哈希以及桶合并,具体实现过程如算法 2 所示.

算法 2 Bucket Merger

```
1 Input Line (the second hashes)
2 compute the AC
3 Set flag = 0
4 Foreach i=1,…,k2
5   For j = 1,…,length of Sortedline_i
6   If((count + = getCount(bucketID of line_i))/
     (average count) <ρ&& dist(bucketID of current bucketID) < cs)
7     Flag_i[bucketID] = flag
8   Else
9    flag = ¬ flag
```

该算法描述了桶合并过程,其中需要考虑三个非常重要的数据,一是每个桶中存放数据个数的 AC,如算法中第 2 行所示. 这个数据比较容易获取,只需用总数据个数除以组合哈希桶的个数即可;第二就是当前已经标记为相同标记的桶的总数据个数与 AC 的比值 ρ 的选取,在实验过程中这个参数设定为 1.5,还有就是应该合并的哈希桶的距离限定 cs,用于控制距离比较远的桶被合并的可能,实现过程如算法 2 中第 6 行所示. 算法 2 中第 7 行 Flag_i 记录了第二层哈希中每条线上的哈希桶号的标记,第 9 行 flag = ¬ flag 则代表了 flag 在 0,1 之间变化的方式.

(2)查询阶段:

图 2 中 $\boldsymbol{q}'$代表了查询对象 $\boldsymbol{q}$ 经过第一次哈希后生成的组合哈希向量,对其进行二次哈希后,可以知道它在直线上所映射的位置,这样只需要在直线上左右探测与其标记相同的其它桶号,那么这些桶中的数据对象也将作为 $\boldsymbol{q}$ 的查询候选集,图中只是展示了一维向量上的情况. 为降低假阴性,可设置多个这样的映射. 举例如下:

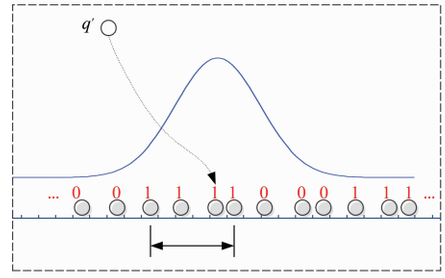


图2 查询示意图

①如果一个查询点 $\boldsymbol{q}$ 经过第一次组合哈希得到了 $\boldsymbol{q}'(\langle 1,1,4,5\rangle,5)$,认为可能还有很多与 $\boldsymbol{q}$ 相近的点被哈希到相邻的桶号中去了. 为了得到这些相邻的桶号,需对 $\boldsymbol{q}'(\langle 1,1,4,5\rangle,5)$进行一次再哈希,就会得到在建立索引的阶段所进行的桶合并中与它相邻的桶号,如发现 $\boldsymbol{o}'_1(\langle 1,2,4,5\rangle,10)$与 $\boldsymbol{q}'$的标记相同,因此,$\boldsymbol{o}'_1$ 中的数据也会被当做 $\boldsymbol{q}$ 的查询候选集.

②如果一个查询点 $\boldsymbol{q}$ 经过 AND 过程得到了这样一个桶号 $\boldsymbol{o}'(\langle 5,7,3,1\rangle,100)$,此时对象数量足够(100),$\boldsymbol{q}$ 就没有再进行二次哈希的需要了. 直接将 $\boldsymbol{o}'$ 中的数据返回作为候选集即可.

③也会存在一种情况,查询点经过后得到的桶号中所包含的数据很少,而经过二次哈希后发现没有与它的桶号打相同标记的其它桶,那么极有可能是因为哈希函数的原因,对于这种情况在 OR 的过程[14]中是可以得到弥补.

在线查找算法的实现见算法 3.

算法 3 Approximate Nearest Neighbor Query

```
1 Input A query object q
2 Access Index file
3  (get the information of table gi,i=1,…,L and Marking line_j,j=1,
   …,k2)
4 Output K (or less) approximate nearest neighbors
5 S←∅
6 Foreach i=1,…,L
```

7　$S \leftarrow S \cup \{$objects found in $g_i(\boldsymbol{q})$ bucket of g_i and with the same mark on k_2 lines of second hashes$\}$

8 Return the K nearest neighbors of $\boldsymbol{q}$ found in set

总结来说,在查找过程中查询对象 $\boldsymbol{q}$,首先,经过第一次哈希得到一个组合哈希向量 $\boldsymbol{q}'$,然后对 $\boldsymbol{q}'$ 进行二次哈希,进而找到那些与 $\boldsymbol{q}'$ 拥有相同标记的组合哈希桶号,将这些哈希桶中的数据也作为 $\boldsymbol{q}$ 的查询候选集,如算法 3 中的第 7 行所示,这样对 L 个哈希表 g_i 都进行同样操作就能得到一个比较高的查询准确率.

5　理论分析

本节对 M2LSH 进行理论分析,分别讨论:(1) M2LSH 中两个距离为 s 的对象经过第一次组合哈希后所得到的组合哈希桶号的距离期望;(2)证明通过合并哈希桶方式可增大正确率;(3)根据所得距离期望,分析假阴性和假阳性;(4)讨论 M2LSH 算法中的参数选择问题.

5.1　组合哈希桶号的距离期望

定理 1　对于距离为 s 的两个对象,经过第一次哈希后得到 k_1 维的组合哈希向量(即桶号)的距离期望的平方为 $\frac{s^2 k_1}{w^2}$.

证明:令 $s = \|\boldsymbol{o}_1, \boldsymbol{o}_2\|$ 表示两个对象之间的距离,从 H 中独立随机选取的函数 $h_{\boldsymbol{a},b}(\boldsymbol{o}) = \left\lfloor \frac{\boldsymbol{a} \cdot \boldsymbol{o} + b}{w} \right\rfloor$ 的作用下,$\boldsymbol{o}_1$ 和 $\boldsymbol{o}_2$ 投影到一维空间. 因此,

$$h_{\boldsymbol{a},b}(\boldsymbol{o}_1) - h_{\boldsymbol{a},b}(\boldsymbol{o}_2) = \left\lfloor \frac{\boldsymbol{a} \cdot \boldsymbol{o}_1 + b}{w} \right\rfloor \left\lfloor \frac{\boldsymbol{a} \cdot \boldsymbol{o}_2 + b}{w} \right\rfloor$$

由于 $\frac{b}{w}$ 为 $(-1,1)$ 之间的小数,因此,

$$h_{\boldsymbol{a},b}(\boldsymbol{o}_1) - h_{\boldsymbol{a},b}(\boldsymbol{o}_2) \approx \frac{1}{w}(\boldsymbol{o}_1 \cdot \boldsymbol{a} - \boldsymbol{o}_2 \cdot \boldsymbol{a})$$

由 p 稳定分布原理可知,$\boldsymbol{o}_1 \cdot \boldsymbol{a} - \boldsymbol{o}_2 \cdot \boldsymbol{a}$ 与 sX 同分布,其中 X 为标准正态分布. 因此

$h_{\boldsymbol{a},b}(\boldsymbol{o}_1) - h_{\boldsymbol{a},b}(\boldsymbol{o}_2)$ 与 $\frac{s}{w}X$ 近似同分布,$(h_{\boldsymbol{a},b}(\boldsymbol{o}_1) - h_{\boldsymbol{a},b}(\boldsymbol{o}_2))^2$ 近似服从 $\frac{s^2}{w^2}X^2$ 分布.

对于组合哈希向量 $g_i(\boldsymbol{o}) = \langle h_1(\boldsymbol{o}), h_2(\boldsymbol{o}), \cdots, h_{k_1}(\boldsymbol{o}) \rangle$,
$\boldsymbol{o}_1$ 和 $\boldsymbol{o}_2$ 的组合哈希向量距离的平方为
$(h_{\boldsymbol{a}_1,b}(\boldsymbol{o}_1) - h_{\boldsymbol{a}_1,b}(\boldsymbol{o}_2))^2 + (h_{\boldsymbol{a}_2,b}(\boldsymbol{o}_1) - h_{\boldsymbol{a}_2,b}(\boldsymbol{o}_2))^2 + \cdots + (h_{\boldsymbol{a}_{k_1},b}(\boldsymbol{o}_1) - h_{\boldsymbol{a}_{k_1},b}(\boldsymbol{o}_2))^2$

式中任意 $(h_{\boldsymbol{a}_i,b}(\boldsymbol{o}_1) - h_{\boldsymbol{a}_i,b}(\boldsymbol{o}_2))^2$ 服从 $\frac{s^2}{w^2}X^2$ 分布. 根据卡方分布的定义,即 n 个独立标准正态分布的随机变量的平方和服从自由度为 n 的卡方分布,因此上式服从 $\frac{s^2}{w^2}X^2_{k_1}$ 分布. 即,对于距离为 s 的两个对象,经过第一次哈希后得到 k_1 维的组合哈希向量(即桶号)的距离期望的平方为 $\frac{s^2}{w^2} * (E(X^2_{k_1})) = \frac{s^2 k_1}{w^2}$. 证毕.

图 3 实验验证了定理 1 的正确性. 我们分别验证了在不同距离下的两个对象,通过不同的哈希桶宽,以及组合哈希函数的个数的情况下,组合哈希向量的距离与理论值的比较. 其中实验值是 100 万次实验的平均值(如图 3 中用 EX 代表). 结果显示理论值与实验值非常接近.

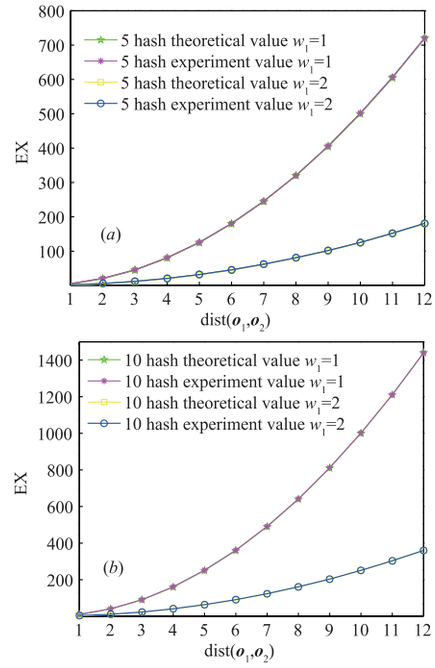


图3　组合哈希向量距离的理论值与实验值的比较

5.2　通过合并哈希桶提高准确性

下面分析究竟能有多大的概率找到距离为 S 的两个对象 $\boldsymbol{o}_1$ 与 $\boldsymbol{o}_2$,即通过组合哈希后在二次哈希桶合并后被找到的概率. 我们知道,对象经过第一次哈希后所落入的桶是如图 4 所示的一种概率关系的.

那么与 $\boldsymbol{q}$ 相邻的一个对象 $\boldsymbol{o}$ 有可能与 $\boldsymbol{q}$ 落入相同的桶中,也有可能会落入与 $\boldsymbol{q}$ 相邻的桶中,事实上 $\boldsymbol{o}$ 落

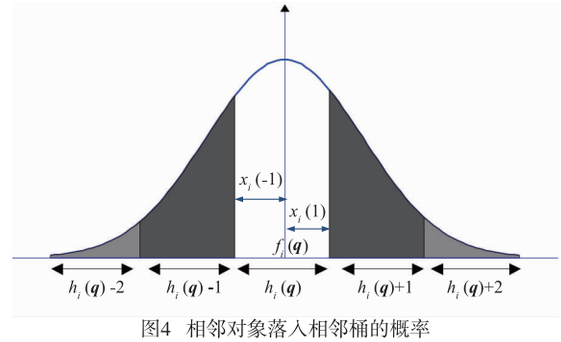


图4　相邻对象落入相邻桶的概率

入 $\boldsymbol{q}$ 的左边还是右边的概率完全依赖于 $\boldsymbol{o}$ 落入的位置与 $\boldsymbol{q}$ 左边界或者右边界有关. 那么与 $\boldsymbol{q}$ 相邻近的点被找到的概率可以通过如下的分析得到.

定义 2 $f_i(\boldsymbol{q})=\boldsymbol{a}_i\cdot\boldsymbol{q}+b_i$ 代表 $\boldsymbol{q}$ 在投影线上的位置(如图 4 所示),则 $h_i(\boldsymbol{q})=\lfloor\frac{\boldsymbol{a}_i\cdot\boldsymbol{q}+b_i}{w}\rfloor$ 代表 $\boldsymbol{q}$ 在线上的桶号,其中, $\boldsymbol{a}_i$ 是一个 d 维的向量,它的每一维选自于服从于 $N(0,1)$ 的高斯分布. $b_i\in\mathbb{R}$ 选自于 $[0,w)$ 的均匀分布.

定义 3 令 $\Delta=(\cdots\ -3\ -2\ -1\ \ 0\ +1\ +2\ +3\ \cdots)$, $x_i(\delta),\delta\in\Delta$ 代表 $\boldsymbol{q}$ 在线上的位置 $h_i(\boldsymbol{q})+\delta$ 的距离,那么可以得到如下的关系:

$$\begin{cases}x_i(-1)=f_i(\boldsymbol{q})-h_i(\boldsymbol{q})*w, x_i(+1)=w-x_i(-1);\\ x_i(-2)=f_i(\boldsymbol{q})-(h_i(\boldsymbol{q})-1)*w, x_i(+1)=w-x_i(-2);\\ \cdots\end{cases}$$

而 $P[h_i(\boldsymbol{o})=h_i(\boldsymbol{q})+\delta]\approx e^{-sx_i(\delta)^2}$,其中 s 是依赖于 $\|\boldsymbol{o}-\boldsymbol{q}\|_2$ 的一个距离[14],在这里 s 依赖于第一次哈希后组合哈希向量的距离 $\sqrt{\frac{s^2k_1}{w^2}}$. 那么可以得到:

$$P[h_i(\boldsymbol{o})=h_i(\boldsymbol{q})+\Delta]=\sum_{i=1}^{M}P[h_i(\boldsymbol{o})=h_i(\boldsymbol{q})+\delta_i]$$

定理 2 通过二次哈希合并相邻组合哈希桶号的方式,可增大查询的准确率.

5.3 M2LSH 假阳性及假阴性

接下来讨论在组合哈希的情况下采用单次哈希和二次哈希以及哈希桶合并情况下的假阳性及假阴性的问题.

首先,根据本文第 3 节中 LSH 的定义,假定这里的 $s=1$, $c=10$,那么 LSH 就是 $(1,\ 10,\ p_1,\ p_2)$ 敏感的,因此根据定义,希望当两个对象的距离 $s=\|\boldsymbol{o}_1,\boldsymbol{o}_2\|_2\leqslant 1$ 的时候,这两个对象被哈希到一起的概率越大越好,也即 p_1 越大越好,而两个对象的距离 $s\geqslant 10$ 的时候,这两个对象被哈希到一起的概率越小越好,也即 p_2 越小越好.

那么,如果从假阴性和假阳性上考虑,当两个对象的距离 $s\leqslant 1$ 时,这两个对象落入相同的哈希桶中的概率应为 p_1,而未落入相同桶中的概率为 $1-p_1$,也就是假阴性. 如果两个对象的距离 $s\geqslant 10$ 的时候,按照设想这两个对象的距离已经比较远了,它们不应该落入相同的哈希桶. 如果它们碰巧以一定的概率落入了相同的哈希桶,那么这个概率就是我们所说的假阳性概率,也就是 LSH 定义中 p_2.

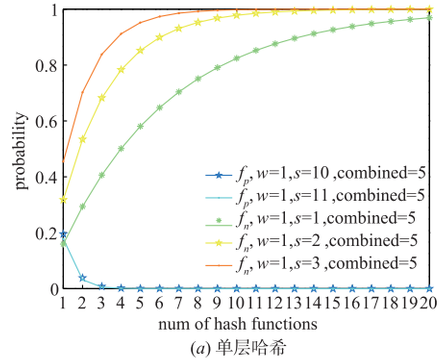
假定组合哈希函数为 $g_i(\boldsymbol{o})=\langle h_1(\boldsymbol{o}),h_2(\boldsymbol{o}),\cdots,h_{k_1}(\boldsymbol{o})\rangle$,表示该组合哈希中包含 k_1 个哈希函数. 这时,当两个对象的距离 $s\leqslant 1$ 时,它们在组合哈希的情况下仍然能够哈希到一起的概率为 $p_1^{k_1}$,不能哈希到一起的概率为 $1-p_1^{k_1}$,为在组合哈希函数的情况下的假阴性. 当两个对象的距离 $s\geqslant 10$,这两个对象在组合哈希的情况仍被哈希到一起的概率为 $p_2^{k_1}$,也即在组合哈希的情况下产生的假阳性.

已知两个对象的距离,可以根据如下的公式得到这两个对象被哈希到一起的概率,进而能够转化成我们所要讨论的假阴性和假阳性.

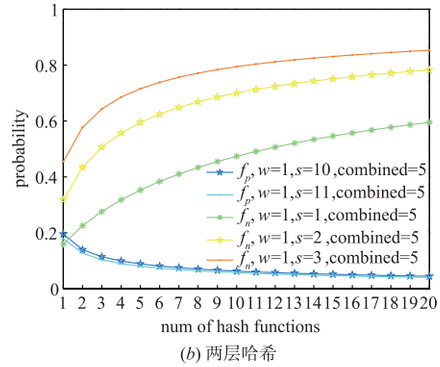
$$\begin{aligned}p(s)&=P[h_{\boldsymbol{a},b}(\boldsymbol{o}_1)=h_{\boldsymbol{a},b}(\boldsymbol{o}_2)]\\&=\int_0^w\frac{1}{s}f\left(\frac{t}{s}\right)\left(1-\frac{t}{w}\right)dt,\end{aligned}$$

接下来对两个对象的距离在 $s=1,2,3$ 时,采用单次组合哈希和采用 M2LSH 方法的情况下,两种方法在合并哈希桶数为 5 时的假阴性上的对比,以及在 $s=10,11$ 时的对比. 在 M2LSH 的方法中,两个对象的距离 s 经过组合哈希后,根据前面的分析,可以求出这两个对象经过组合哈希后所得到的组合哈希桶号的欧式距离的期望为 $s'=\sqrt{\frac{s^2k_1}{w^2}}$,在第二次哈希的时候,用这个新的距离来进行哈希得到 M2LSH 算法在 s 为不同值时的假阴性及假阳性. 通过以上的理论分析,可以得到如图 5 的分析结果:

图 5(a)表示在一个哈希表中只采用一次组合哈希,以及哈希桶宽 $w=1$,统计合并 5 个哈希桶的假阳性及假阴性随着组合哈希函数个数 k_1 增长的结果. 图 5(b)为



(a) 单层哈希



(b) 两层哈希

图5 假阳性及假阴性

M2LSH 算法在第一次组合哈希桶宽为 $w_1=1$,第二次单个哈希桶宽 $w_2=1$,合并哈希桶个数为 5 时随着组合哈希函数个数 k_1 增长的结果. 从图 5 的结果图中可以得出在单个哈希表中采用两次哈希的处理方式虽然相对于只采用单次组合哈希的方式增大了差不多 10% 的假阳性,但在图 5(b)中的假阴性降低了超过 20%.

5.4　参数选择

假设当两个对象的距离 $s\leqslant 1$ 时,这两个对象是相近的;当两个对象的距离 $s\geqslant 10$ 时,这两个对象比较远. 下面以此为例讨论参数选择.

因为当两个对象的距离 $s=1$ 时,在第一次组合哈希桶宽 $w_1=1$,经过第一次组合哈希后距离变为 $s'_1=\sqrt{\frac{s^2}{w^2}*k_1}=\sqrt{k_1}$. 当两个对象的距离 $s=10$ 时,在第一次组合哈希桶宽 $w_1=1$,经过第一次组合哈希后距离变为 $s'_2=\sqrt{\frac{s^2}{w^2}*k_1}=10\sqrt{k_1}$ 因此,如果希望达到在二次哈希的过程中每个哈希函数都能有假阴性 $\leqslant 0.5$,假阳性 $\leqslant 0.05$,并且二次哈希的桶宽设定为 $w_2=1$,首先讨论假阴性,即

$$1-p(s'_1)=P[h_{a,b}(\boldsymbol{o}_1)=h_{a,b}(\boldsymbol{o}_2)+\Delta]\leqslant 0.5$$

$$\text{得},\int_0^{n*w_2}\frac{1}{s'_1}f\left(\frac{t}{s'_1}\right)\left(1-\frac{t}{w_2}\right)\mathrm{d}t\leqslant 0.5$$

其中,$\Delta=(-2\quad -1\quad 0\quad +1\quad +2)$,$n$ 为在二次哈希的过程中合并的哈希桶的个数,可以推断出当 n 取 5 的时候 $k_1\leqslant 16.2659$,可以推断出在单个哈希表中要满足一定的假阴性时,组合哈希函数个数的上界也即在本文中 k_1 的上界. 接着讨论假阳性,即

$$p(s'_2)=P[h_{a,b}(\boldsymbol{o}_1)=h_{a,b}(\boldsymbol{o}_2)+\Delta]\leqslant 0.05$$

$$\text{得},\int_0^{n*w_2}\frac{1}{s'_2}f\left(\frac{t}{s'_2}\right)\left(1-\frac{t}{w_2}\right)\mathrm{d}t\leqslant 0.05$$

则可以推断出 $k_1\geqslant 15.64$. 因此,我们可以设定在第一次组合哈希的过程中组合哈希函数的个数 k_1 为 16,并且在二次哈希合并哈希桶的个数大约为 5 的情况下就可以得到这样的一组假阳性假阴性的要求了.

讨论过程中假阴性取值为 0.5,事实上,这是一个很大的值,对于近邻查询来说不可取,而这是在第二次哈希过程中只有一个哈希函数的结果,通过增加二次哈希函数的个数,可以降低这个值. 如,在 M2LSH 中,取第二次哈希的哈希函数个数为 3,那么,通过这三个哈希函数的 OR 之后假阴性变为:$1-(1-0.5^3)=0.125$. 通过增大哈希表的个数可以进一步降低假阴性.

6　实验

实验平台 CPU 为 A8 7100、8GB DDR3 内存. 通过合成数据集和真实数据集来评估 M2LSH 的性能,并和 C2LSH 以及 Multi-probe LSH 算法进行比较.

6.1　数据集

利用 3 个常用于评估现存 LSH 算法性能的真实数据集:Mnist, Color 和 Audio. 此外,还生成了一个符合 ZIPF[21] 非均匀分布的合成数据集,称之为 Zipf.

Mnist:Mnist 数据集包含了 60,000 个 784 维的数据对象,每一个数据对象都是一个大小为 28×28 的图像. 由于大多数的像素都并不显著,所以只保留了 80 维方差最大的坐标值,从而每个数据对象变成 80 维. Mnist 数据集还包含了一个大小为 10,000 的测试集(Test Set). 随机从测试集中抽取 50 个数据对象作为查询集(Query Set),并用数据集提取的 50 个对应的维度生成查询对象.

Color:Color 数据集包含了 68,040 个 32 维的数据点,其中每一个数据点都是来自 Corel collection 中的图片的颜色直方图(Color Histogram). 随机地从数据集中抽取 50 个数据点作为查询集,并把这 50 个点从数据集中删除. 从而得到一个大小为 67,990 的数据集和一个大小为 50 的查询集.

Audio:Audio 数据集包含了 54,384 个 192 维的数据点,从中随机抽取了 50 个数据点作为查询集并把它们从数据集中除掉. 从而得到一个大小为 54,334 的数据集.

Zipf:首先随机生成了 100 个 100 维彼此距离比较远的数据对象,用 $\boldsymbol{q}_{\text{center}}$ 表示. 然后再分别以这 100 个数据对象为中心,随机生成的 50~150 个与这些中心点不同距离上占不同比例的数据点,最终形成以 100 个数据为中心的分布的不均匀数据集,其大小均为 85,300 个 100 维的数据对象,查询集我们采用的是从这 100 个中心点中随机抽取的 50 个作为查询对象.

6.2　性能评估函数

采用正确率、平均查询时间两个方面来对 M2LSH 算法的性能进行评价.

正确率　用来评价返回的查询结果的准确性. 已知一个查询点 $\boldsymbol{q}$,而 $\boldsymbol{o}_1^*,\boldsymbol{o}_2^*,\cdots,\boldsymbol{o}_K^*$ 为 $\boldsymbol{q}$ 的 KNN 结果,作为 ANN 查询的结果,最终也会返回 K 个与 $\boldsymbol{q}$ 相近的点,用 $\boldsymbol{o}_1,\boldsymbol{o}_2,\cdots,\boldsymbol{o}_K$ 来表示. 这两种结果均会按照与查询点 $\boldsymbol{q}$ 的真实距离按照升序的方式进行排序. 因此,点 $\boldsymbol{q}$ 的 ANN 查询结果的正确率可以用如下公式来得出.

$$\text{Ratio}(\boldsymbol{q})=\frac{1}{K}\sum_{i=1}^{K}\frac{\|\boldsymbol{o}_i,\boldsymbol{q}\|}{\|\boldsymbol{o}_i^*,\boldsymbol{q}\|},$$

在接下来的实验中采用查询结果集中 Ratio($\boldsymbol{q}$)的平均值来表示 M2LSH 算法的正确性.

平均查询时间　查询时间主要由两部分组成,一部分是在多个表中查找候选集的时间,一部分是从候选集中找到并返回最终查询结果的时间. 事实上,在整个查询时间中,花费在第二部分的时间占据了大部分. 因此,如果能够减小在第二部分中的时间开销,那么算法的总的

查询时间可以得到进一步的提升. 而 M2LSH 算法与 C2LSH 以及 Multi-probe LSH 算法相比,会在保证假阴性以及假阳性的前提下返回更小的候选集,这样在从候选集中找到真正候选结果的时候就比较节省查询时间. 以 t_i 代表查询集中第 i 个查询的时间开销,N_q 代表查询集的大小,M2LSH 的平均查询时间用如下形式来表示:

$$\mathrm{ART}=\frac{1}{N_q}\sum_{i=1}^{N_q}t_i.$$

6.3 算法中各个参数的讨论

在 M2LSH 算法中,有几个参数会直接影响算法的性能,包括第一层哈希桶宽 w 的设置,组合哈希函数中哈希函数个数 k_1 的选取以及哈希表的个数. 在接下来的实验中,我们将通过调试这些参数,来对 M2LSH 算法的性能进行评价.

首先,来分析桶宽 w 和哈希函数个数 k_1 对实验结果的影响. w 和 k_1 是构成 LSH 索引文件的两个基本参数,通过在一个哈希表中来对它们进行调试. 对于不同的 w 和 k_1 进行了一组实验,通过对查询集(每个实验数据的查询集大小均为 50)返回的查询结果的 Ratio 进行评价.

一般地,从下图 6 中可以得出,桶宽 w 和第一次组合哈希函数的个数 k_1 对实验结果的影响是共同起作用的. 当 w 很小的时候,返回结果的 Ratio 也并不是很高,这是因为哈希函数是随机选取的. 在 Zipf(图 6(d))数据实验中,可以发现,当 w 取最小值 0.5 的时候,Ratio 仅仅在 1.3 左右. 在其它的几个实验数据集中,也可以发现这个现象. 这是因为当 w 很小的时候,较少的哈希函数已经可以达到一个比较完美地识别能力. 因此,即使增加更多的哈希函数的个数也并不会使查询结果有一个明显的提升. 反而因为哈希函数的增加从而影响查询时间. 当第一层哈希函数的个数达到一定程度的时候,继续增加哈希函数的个数,对查询结果的正确性的提高也并不是太大,因此在试验过程中,取 $k_1=16$ 就可以得到一个比较好的实验结果,与 5.4 节讨论一致.

通过对比试验结果中不同 w 所对应的 Ratio 比较平稳的部分,得出当 k_1 很小的时候 Ratio 会很大. 例如,在 Color(图 6(b))实验结果中,当 $w=0.5$ 的时候,稳定部分 Ratio 的值明显比其它 w 所对应的 Ratio 要高. 当 w 很小的时候,在哈希的过程中,极有可能将原本非常相近的数据哈希到相邻的哈希桶中. 这就会导致丢候选结果的可能. 而 M2LSH 算法中在进行二次哈希的过程中可以很好的降低此类问题的可能性. 相反,如果 w 太大,就会使很多本来相距较远的数据哈希到相同的桶中. 为了减小这种问题的可能性,又不得不增加哈希函数的个数,这就使得后续的处理增加了时间的开销. 在 M2LSH 中,用较小的 w 以及较少的哈希函数就能得到比较好的效果.

接下来,再对哈希表的个数对实验的结果的影响进行实验. 哈希表的个数主要影响的是查询返回结果的假阴性以及空间开销. 直觉上,哈希表的个数越多,就可以从这些哈希表中获取更多的信息. 返回的查询结果就会更加精确,但耗费的查询时间就会更多. 为了测试哈希表的个数对 Ratio 和 ART 的影响,分别对哈希表的个数为 2、3、4、5、6 进行了试验. 每个实验数据的查询集仍然设定为 50.

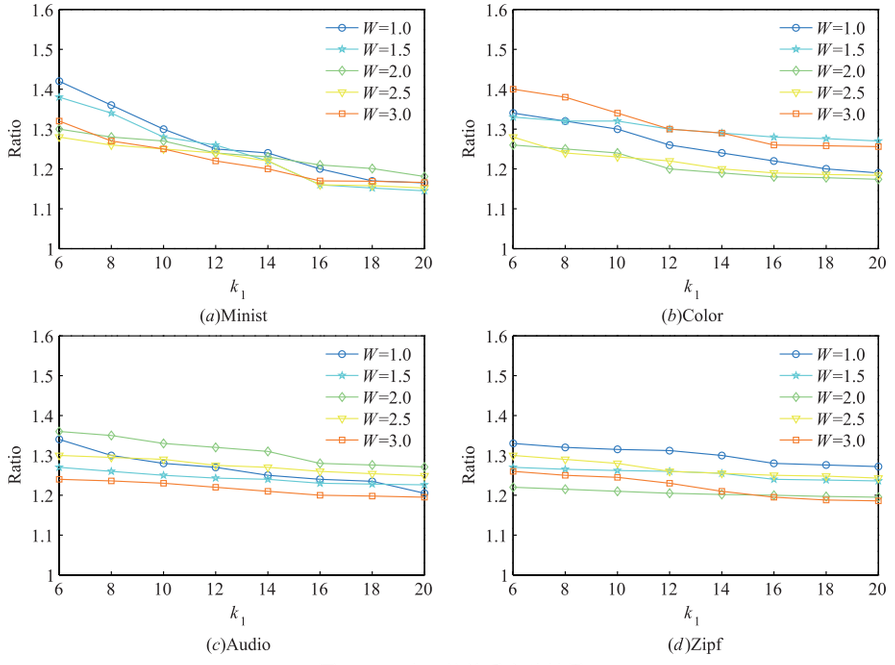
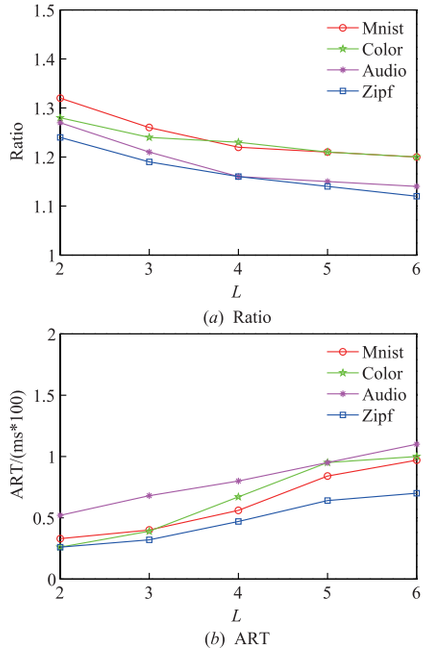


图6 Ratio与k_1的关系实验结果

正如所预期的一样. 当哈希表的个数 L 从 2 增长到 6 的过程中,Ratio 呈下降趋势,也即哈希表个数的增加会提高查询正确率,如图 7 所示. 这与我们的直觉是一致的,因为随着哈希表个数的增加,查询结果的质量也在提高. 同样它也证明了在第 4 部分所提出的 M2LSH 算法是有效的. 更重要的是,随着 L 个数的增长,Ratio 下降的趋势呈现一种减缓的趋势. 这表明当哈希表的个数达到一定程度后,继续增加哈希表的个数,对返回结果的提升促进作用并不大. 尤其是当 L 超过 4 的时候,就更加明显了.

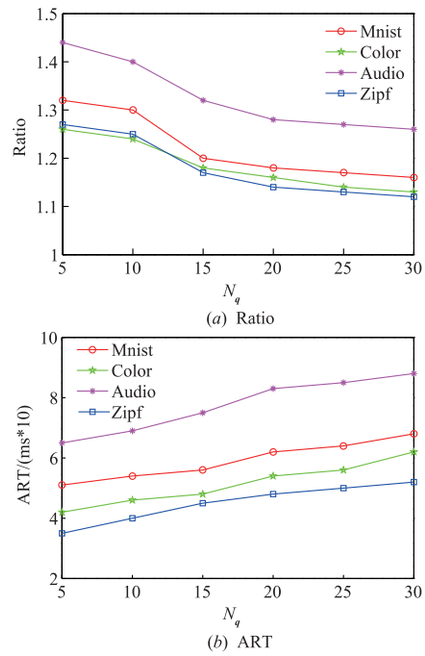


(a) Ratio　(b) ART

图7　Ratio、ART与table个数的关系实验结果图

同样查询集的大小 N_q,也会直接影响实验的结果. 理论上查询集 N_q 越大,所得到的实验结果就会更加能够代表算法的真实性能. 在实验中,我们分别对当查询集为: N_q =5、10、15、20、25、30 的情况单独进行了测试. 并且结果如图 8 所示. 正如所预期的一样当 N_q 增大的时候,Ratio 在所有的实验数据集中均呈现下降的趋势,如图 8(a). 至于 ART,它会随着 N_q 的增长而增长(如图 8(b)),最后趋于平缓,这也印证了算法的可靠性.

探测率代表的是每查询一个数据对象所得到的候选集与数据集大小的比值. 这也从一个方面反映了算法在查询时的性能,同时也是 M2LSH 算法在提高算法性能(高速、准确)上的一个比较关键的因素. 探测率越小代表找到最终返回结果需要遍历的数据量就越少,那么在相同返回结果要求下,所需要的时间就越少. 如果在减小候选集的情况下又保证了一定的正确率,这样算法不仅能提高查询速度,而且可以提高正确率. M2LSH 在探测率上的实验结果如图 9 所示:从中可以得出,随着哈希表



(a) Ratio　(b) ART

图8　Ratio、ART与查询N_q的关系

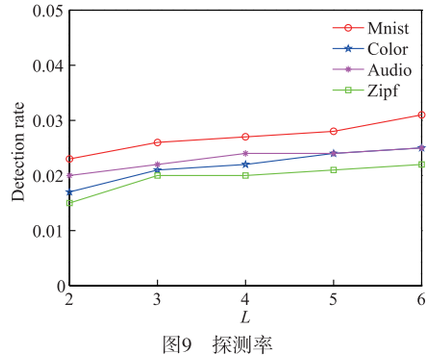


图9　探测率

个数的增长,M2LSH 在探测率上的增长是十分平缓的. 尤其对于一些数据分布不是很均衡的数据集如 Zipf,M2LSH 的探测率比其它三个比较均匀的数据集要更高一些. 这也符合了 M2LSH 算法的设计初衷.

接下来,将在 Mnist、Color、Audio 以及合成数据 Zipf 四个数据集上,与 Multi-probe LSH 和 C2LSH 就 Ratio、ART 上的情况进行对比,在试验的过程中我们设置第一层组合哈希函数个数 k_1 =16,二层哈希函数个数 k_2 =3,第一层哈希桶宽 w_1 =1,二层哈希桶宽 w_2 =1,哈希表个数 L =4.

为了做一个比较全面的对比,将 K (K 代表最终要返回的近邻的个数)设定为{1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100}. 然后分别统计三个算法在 Ratio 以及 ART 上的平均情况. 其中,在 Ratio 上的情况如图(10)所示.

非常明显,M2LSH 在 Ratio 上的值均小于 Multi-probe LSH 和 C2LSH 的. 而 Ratio 代表了返回的近邻与真实近邻之间的接近程度,Ratio值越小就代表越接近,

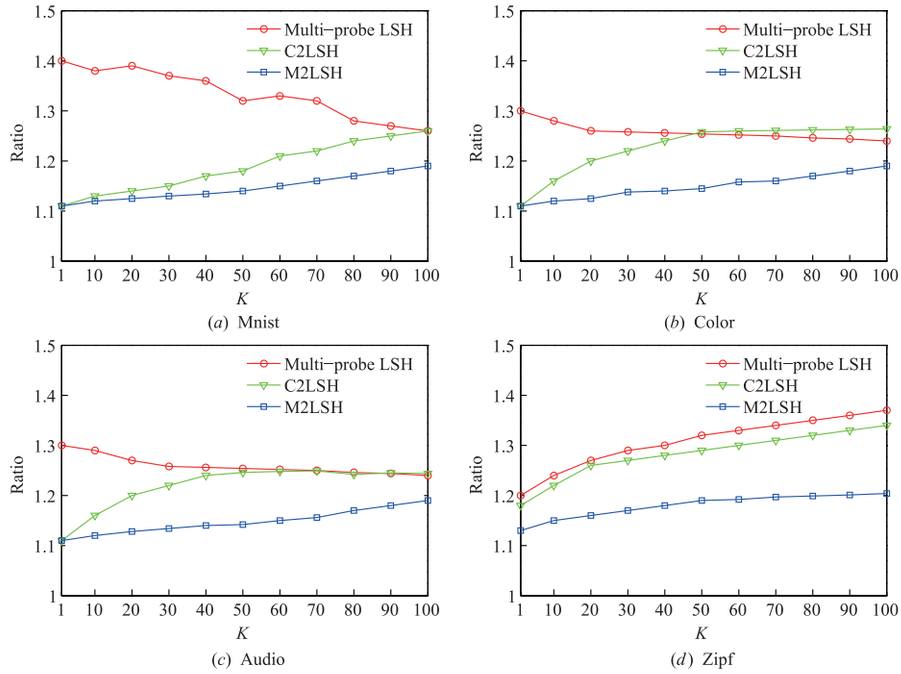


图10 三个算法在Ratio上的比较

从而也表明返回的值越精确. 特别是在 Zipf 数据集中，可以发现 M2LSH 在 Ratio 上一直处于较小值得水平，与真实值非常的接近. 此外，也不难发现，随着 K 的增长，M2LSH 与 C2LSH 在 Ratio 上均呈现出增长的趋势，但 M2LSH 增长的要更加缓慢一些. 而 Multi-probe LSH 则相对来说比较平稳甚至有下降的趋势，但是即使这样它的 Ratio 值依然是三个算法中最高的.

在 ART 上的实验结果如图 11 所示，与其它两个算法相比，M2LSH 要好一些. 特别是在 Zipf 数据集上，这也表明了 M2LSH 算法在查询时间上的开销不仅与数据维度相关，还与数据的分布情况相关. 从实验结果图中可以得出，三个算法在在这四个数据集上的平均每

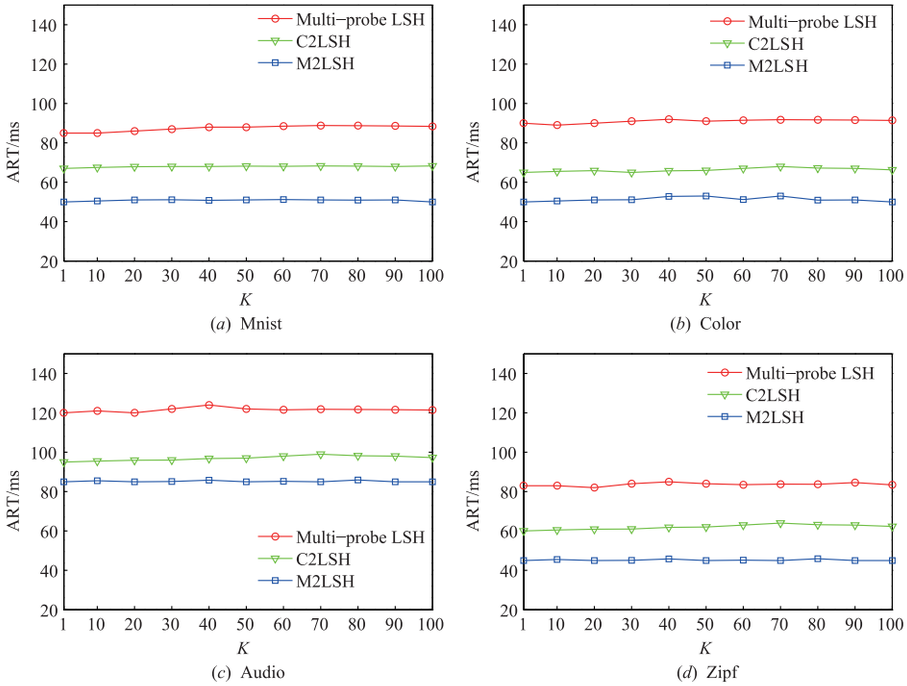


图11 三个算法在ART上的比较

次查询的时间开销随着 K 的增长是比较稳定的. 但是,M2LSH 算法的时间开销要小于其它两者,而 C2LSH 又比 Multi-probe LSH 在时间开销上小一些. 特别当在数据分布不是很均匀的时候,M2LSH 算法在查询时间上的优势就更加的明显,如图 11(d)所示.

6.4　算法在非均匀数据集上的性能

为了进一步验证 M2LSH 在非均匀高维数据集上的性能,我们合成了 4 个不同均匀度的 ZIPF 实验数据集,Zipf1 ~ Zipf4. 在生成这些数据集的过程中,按照如下的原则进行[21]:

$$P(s)=\frac{c}{s^{\alpha}}$$

这里 s 表示随机生成的数据对象与中心数据对象 $\boldsymbol{q}_{\text{center}}$的距离,$P(s)$表示距离为 s 的数据对象的出现频率. 通过数据频率分布中对 C 设定不同的值来生成不同分布的测试数据集 Zipf1-Zipf4,在实验的过程中 α 取值为 1. 四个测试数据集的分布情况如图 12(a)所示. 图 12(b)是 M2LSH 算法的处理这些数据集的效果. 从图中可以看到,尽管数据分布不同, M2LSH 无论在查询正确率和查询时间都是比较稳定的.

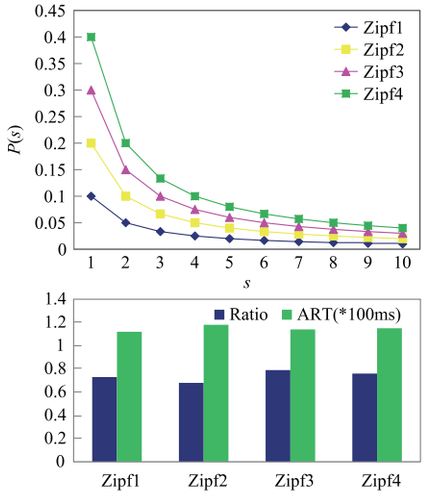


图12　非均匀数据集及其Ratio、ART

7　总结

在基于流行的 LSH 技术上,论文提出了一种新的索引结构 M2LSH,来解决高维空间下的 ANN 问题. 为了加快查询速度,采用了对第一次组合哈希后的组合哈希桶号,进行二次哈希来找与该桶相近的其它桶的策略. 并且在基于桶计数以及在实际的查询中需要返回的查询结果的数量提出了哈希桶合并的方式. 使得最终索引文件中每个桶中的数据个数比较均衡. 论文对 M2LSH 的算法的性能进行了评估,通过大量比较实验,证明了 M2LSH 算法的有效性.

M2LSH 算法对第一次哈希过程的桶宽有较高的要求,一般希望通过合并能使合并桶中数据尽量均匀,但是对于特别不均匀数据集,即无法通过合并来均匀,算法会退化到一般 LSH 性能. 同时由于涉及的参数较多,如何使算法的性能达到最优,这些都是今后研究工作需要解决的问题.

参考文献

[1] Samet Hanan, J Gray. Foundations of Multidimensional And Metric Data Structures[M]. Elsevier:Morgan Kaufmann,2006. 150 – 360.

[2] Guttman A. R-trees:A dynamic index structure for spatial searching[A]. 1st ACM SIGMOD International Conference on Management Of Data[C]. New York,USA:SIGMOD,1984. 47 – 57.

[3] 过洁,徐晓旸,潘金贵. 虚拟场景的一种快速优化 Kd-Tree 构造方法[J]. 电子学报,2011,39(8):1811 – 1817.
GUO Jie, XU Xiao-yang, PAN Jin-gui. Build Kd-tree for virtual scenes in a fast and optimal way[J]. Acta Electronica Sinica,2011,39(8):1811 – 1817. (in Chinese)

[4] Burkhard W A, Keller R M. Some approaches to best-match file searching[J]. Communications of the ACM, 1973,(16):230 – 236.

[5] 庄毅,胡海洋,胡华. 基于质心片的不确定高维索引研究[J]. 电子学报,2011,39(5):1136 – 1142.
ZHUANG Yi,HU Hai-yang,HU Hua. Centroid-slice-based uncertain high-dimensional indexing structure[J]. Acta Electronica Sinica, 2011, 39(5): 1136 – 1142. (in Chinese)

[6] Katayama,Norio,Satoh,et al. The SR-tree:An index structure for high-dimensional nearest neighbor queries[J]. Acm Sigmod Record,1997,26(2):369 – 380.

[7] Kevin S Beyer,Jonathan Goldstein,Raghu Ramakrishnan, et al. When is "nearest neighbor" meaningful? [J]. Lecture Notes in Computer Science,1999,16(18):217 – 235.

[8] Berchtold S, Bohm C, Jagadish H V, et al. Independent quantization:an index compression technique for high-dimensional data spaces[A]. IEEE International Conference on Data Engineering[C]. San Diego,CA:IEEE,2000. 577 – 588.

[9] Beygelzimer A,Kakade S,Langford J. Cover trees for nearest neighbor[A]. 23th International Conference on Machine Learning[C]. Montreal, Canada: ACM, 2006. 97 – 104.

[10] 许新征,丁世飞,史忠植,贾伟宽. 图像分割的新理论和新方法[J]. 电子学报,2010,38(2A):76 – 82.
XU Xin-zheng, DING Shi-fei, SHI Zhong-zhi, JIA Wei-kuan. New theories and methods of image segmentation

[J]. Acta Electronica Sinica,2010,38(2A):76 – 82. (in Chinese)
[11] 陆哲明,潘正祥,孙圣和. 一种矢量量化码书搜索的快速算法[J]. 电子学报,2000,28(2):133 – 135.
LU Zhe-ming, PAN Jeng-shyang, SUN Sheng-he. A fast codebook search algorithm for vector quantization[J]. Acta Electronica Sinica,2000,28(2):133 – 135.
[12] Indyk P, Motwani R. Approximate nearest neighbors: towards removing the curse of dimensionality[A]. Proc of the 30th Annual ACM Symposium on Theory of Computing[C]. Dallas, TX, USA: ACM, 1998. 604 – 613
[13] Gionis A, Indyk P, Motwani R. Similarity search in high dimensions via hashing[A]. Proc of the 25th International Conference on Very Large Data Bases[C]. Edinburgh, Scotland, UK: Morgan Kaufmann Publishers Inc VLDB, 1999. 518 – 529.
[14] Datar M, Indyk P, Immorlica N, et al. Locality-sensitive hashing scheme based on p-stable distributions[J]. SoCG,2004,34(2):253 – 262.
[15] Panigrahy, R. Entropy-based nearest neighbor algorithm in high dimensions[A]. ACM-SIAM Symposium on Discrete Algorithms[C]. Florida, USA: SODA, 2006. 155 – 163.
[16] Qin Lv, William Josephson, Wang Zhe, et al. Multi-probe lsh: Efficient indexing for high-dimensional similarity search[A]. 33th International Conference on Very Large Data Bases[C]. Vienna, Austria: VLDB, 2007. 950 – 961.
[17] Tao Yufei, Yi Ke, Sheng Cheng, et al. Quality and efficiency in high dimensional nearest neighbor search[A]. 35th SIGMOD International Conference on Management of Data[C]. New York, USA: SIGMOD, 2009. 563 – 575.
[18] Gan Junhao, Feng Jianlin, Fang Qiong, et al. Locality sensitive hashing scheme based on dynamic collision counting[A]. 38th International Conference on Management of data[C]. Scottsdale, Arizona, USA: SIGMOD, 2012. 541 – 552.
[19] Liu Yingfan, Cui Jiangtao, Huang Zi, et al. SKLSH: An efficient index structure for approximatenearest neighbor search[A]. 40th International Conference on Very Large Data Bases[C]. Hangzhou, China: VLDB, 2014. 745 – 756.
[20] Qian Jiangbo, Zhu Qiang, Chen Huahui. Multi-granularity locality-sensitive bloom filter[J]. IEEE Transactions on Computers, 2015, 64(12): 3500 – 3514.
[21] Newman M E J. Power laws, Pareto distributions and zipf´s law[J]. Contemporary Physics, 2005, 46(5): 323 – 351

作者简介

李 灿 男,1989 年出生于湖北荆门,宁波大学信息科学与工程学院硕士研究生,主要研究方向为大数据处理,数据挖掘.
E-mail:nblcan@163.com

钱江波(通信作者) 男,博士,1974 年出生于浙江宁波. 宁波大学教授,主要研究方向为数据库技术、流数据处理、多维数据索引技术等.
E-mail:qianjiangbo@nbu.edu.cn

董一鸿 男,博士,1969 年出生于浙江宁波. 宁波大学教授,主要研究方向为大数据处理、数据挖掘和人工智能等.
E-mail:dongyihong@nbu.edu.cn

陈华辉 男,博士,1964 年出身于浙江鄞州. 宁波大学教授,主要研究方向为数据库技术、流数据处理等.
E-mail:chenhuahui@nbu.edu.cn