锥树最大内积搜索

薛明淇

2017 年 4 月 28 日

摘要

针对欧氏距离和余弦相似度查找最佳匹配的问题已经被广泛地研究了。然而就我们所知，另一个与之相关的问题，通常意义上的对于最大内积高效地查找最佳匹配却从来没有被探索过。首先，我们在这里提出一个通常意义上基于单个树形数据结构的“branch-and-bound”算法。随后，我们提出了一个双树来应对有多个查询的情况。最后我们会给出一个新的数据结构，锥树，来增加双树的效率。我们用了各种应用中产生的的大量数据集对这些算法进行了评估，在某些情况下，与暴力搜索的算法相比，效率可以提高多达 5 个数量级。

1 介绍

在这篇论文中，我们讨论的是面对针对内积相似度的查询，在一个给定的点集内如何高效查找最佳匹配的问题。搜索的效率是我们主要的关注点。正式地，我们考虑的是下面这个问题：

最大内积查询 给定一个包含 N 个点的集合 $S \subset R^D$ ，高效地找到一个点 $p \in S$ 使得

$$\langle q, p \rangle = \max_{r \in S} \langle q, r \rangle \tag{1}$$

乍看之下，这个问题与很多已有文献做的工作十分类似。针对欧氏距离（或者更普遍地 L_p）的最佳匹配是度量空间中被广泛研究的最近邻搜索的问题。对于余弦相似性的最佳匹配的高效检索已经在文本挖掘和信息检索领域中被研究过。不过正如我们会在下一节中解释的那样，最大内积搜索不仅与上述问题不同，而且更加困难

1.1 应用

最大内积搜索的一个显然的应用来自于已经大获成功的推荐系统中所用的矩阵分解框架，如“Netflix Prize”中所用的。矩阵分解使得就“用户”向量和“商品”向量而言，可用的数据可以被精确地表示出来（“商品”的例子比如音乐或者电影）。在这个背景下，用户对于商品的偏好就是相对应的“用户”向量和“商品”向量的内积[1]。如果把用户当做查询，商品当做参考集合的话，对于用户推荐的检索就等同于最大内积搜索的问题。通常会使用线性扫描来查找最佳的推荐。一个高效的搜索算法会使得在矩阵分解框架中的推荐检索可以向更大的系统中拓展。

常用的文档检索任务会使用余弦相似性来匹

[1] 更好的方法其实是用户和商品向量的内积加上一个偏移项，但是如果把用户向量尾部添加一个 1，商品向量添加一个商品偏移量，这个问题就退化成最大内积问题了

配文档。然而在特定的环境下，文档会被表示成（不一定是标准化的）向量，向量间的内积就代表它们的相似度。在这样的情况下，除非向量被标准化成同样的长度，用余弦相似度来做文档匹配只能在返回不准确的结果的代价下使这个算法可拓展，因为内积和余弦相似性不一样的（我们会在第二节中进一步讨论这个问题）。

还有一个类似的问题就是最大核操作：对于给定的一个点集 S，一个查询 q 和一个核函数 $\mathcal{K}(\cdot,\cdot)$，任务就是在集合 S 上找到使得 $\mathcal{K}(p,q)$ 最大的点 $p \in S$ 这个问题在机器学习中的最大后验推理，计算机视觉中的图像匹配中都被广泛地应用。如果核函数可以被显式地表达成 $\varphi(\cdot)$ 使得 $\mathcal{K}(p,q) = \langle\varphi(q),\varphi(r)\rangle$ 的形式，并且在所有 S 中的点和查询 q 都被转换到 φ 空间，那么这个问题就退化最大内积搜索问题了。

1.2 这篇论文

在这篇论文中，为我们提出了两个基于树的“branch-and-bound”算法以及一个新的数据结构来解决这个问题。在第二节中，我们把这个问题和与度量空间中一些更为常见的问题，如最近邻搜索和对于余弦相似性的最佳匹配问题进行对比。在第三节中，我们使用已有的球树数据结构和一个新颖的约束，提出了一个简单的“branch-and-bound”算法。在下一节（第四节）中，我们讨论了在同一个点集上进行多次查询的情况，并且提出了基于双树的“branch-and-bound”算法。在第五节中，我们将展示一个新的数据结构，锥树，来对双树的算法进行索引。这些数据结构利用了一个新颖的内积约束，这比传统的球树所能达到的更好。第六节中，所提出的这个算法会针对不同的数据集进行评估。而第七节证明了我们提出的这个算法可以被应用于最大核操作，并且使用一个通用的核函数而不需要任何显式的 φ-空间的点的表示。在最后一节中，我们会给出我们的结论以及未来一些可能的在这个问题上的工作方向。

2 最大内积搜索

在欧几里德度量空间中存在许多最近邻搜索的技术（见一些调查，如 [9]）。大量针对余弦相似度测度的最佳匹配算法也已经被开发出来，其中很多都在关注文本类型的数据。最近邻搜索的问题（在度量空间中）已经被广泛流行的局部敏感哈希（LSH）大致地解决了。LSH 已经被推广到其他形式的相似性函数（与把距离作为不相似函数相反）如余弦相似性

[2]。

在核函数满足特定条件的情况下，也可以通过 LSH 高效地解决近似的最大核操作。“dimension reduction”和双树算法也被用来高效地解决近似的最大核操作。

2.1 最大内积搜索和已有问题的不同

这里我们将解释为什么最大内积搜索和其它已有的搜索问题都不太一样。因此用于解决那些问题的技术（如 LSH）不能直接应用于这个问题。

[2]这里需要注意的是，Charikar 等人用的相似性函数不完全是余弦相似性。两个点 p 和 q 的距离用 θ/π 来计算，其中 θ 是两个点在原点处的夹角，使得相似性函数变为 $\left(1-\frac{\theta}{\pi}\right)$。这个相似性函数可以跟余弦相似度是直接对应的关系

欧式空间内的最近邻搜索 这涉及到对于一个查询 q 找到一个点 $p \in S$ 使得:

$$p = \arg\min_{r \in S} \|q - r\|_2^2 = \arg\max_{r \in S} \left(\langle q, r \rangle - \frac{\|r\|_2^2}{2} \right)$$
$$\neq \arg\max_{r \in S} \langle q, r \rangle (\text{unless } \|r\|_2^2 = k \forall r \in S).$$

因此，如果 S 中所有点的范数被标准化微相同的长度，那么最大内积搜索等价于欧式度量空间中的最近邻搜索。然而没有这个限制的话，这两个问题就可能会有非常不同的解。

余弦相似度的最佳匹配 这构成了对于一个查询 q 找到一个点 $p \in S$ 使得:

$$p = \arg\max_{r \in S} \frac{\langle q, r \rangle}{\|q\| \|r\|} = \arg\max_{r \in S} \frac{\langle q, r \rangle}{\|r\|}$$
$$\neq \arg\max_{r \in S} \langle q, r \rangle (\text{unless } \|r\|_2^2 = k \forall r \in S).$$

仅当集合 S 中的所有点被标准化成相同的长度时，对于余弦相似性的最佳匹配可以给出最大内积

局部敏感哈希 LSH 已经被应用于各种相似性函数。LSH 参与构造哈希函数使得对于每个哈希函数 h，以及任何一对点 $r, p \in S$ 满足下面这个条件:

$$\Pr[h(r) = h(p)] = \text{sim}(r, p) \tag{2}$$

其中 $\text{sim}(r, p) \in [0, 1]$ [3] 是我们所感兴趣的相似度函数。对于我们的情况，我们可以标准化我们的数据集使得 $\forall r \in S, \|r\| \leq 1$[3]，并且假设所有的数据都在第一象限（这样没有内积会比零要小）。在这样的情况下时，$\text{sim}(r, p) = \langle r, p \rangle \in [0, 1]$ 就是一个我们所关心的一个有效的相似度函数

[3]这里的标准化和前面提到的标准化是不同的，在前面所有的点被标准化具有相同的长度。在这里点的长度被标准化为小于等于 1，但互相并不一定相等

我们知道对于任意一个相似形函数来接受一个局部敏感哈希函数族，距离函数 $\mathbf{d}(r, p) = 1 - \text{sim}(r, p)$ 必须满足三角不等式（[7] 中的引理 1）。然而，距离函数 $\mathbf{d}(r, p) = 1 - \langle r, p \rangle$ 并不满足三角不等式（甚至当所有点都被限制在第一象限时）[4]。所以就算所有的点都被限制在第一象限时（这是一个非常严格的条件），LSH 还是无法应用在内积相似性函数上。

高效最大核操作 各种技术都已经被提出用于高效地解决这个问题。对于具有非常高（可能为无穷）维显式表示的核函数，Rahimi 等人提出了一个把这些高维表示变换成较低维的技术，同时仍然近似地保留了内积来提高可拓展性。然而，最终的研究仍然使用了对于点集的线性扫描来查找最大内积，或者在找到最近邻等价于找到最大内积的假设下，使用最近邻搜索。对于一些与翻译无关的核函数 [5]，一些基于树的算法已经被拓展使用到更大的数据集合中。然而，这个算法是否可以推广到一般的核函数仍然不清楚。LSH 在计算机视觉中被广泛应用于图像匹配，然而仅适用于承认局部敏感哈希函数的核函数。因此，现有的技术中没有一个可以在不损失结果的准确性，或者不做一些限制性假设的情况下，直接应用于最大内积搜索。

2.2 为什么最大内积搜索有可能会更困难

最大内积缺少一种普遍运用的相似函数的一个基本的性质 - 巧合。比如，一个点到它自己的

[4]反例: 令点 $x, y, z \in S$ 使得 $\|x\| = \|y\| = \|z\| = 1$，$x$ 和 y，y 和 z，z 和 x 之间的夹角分别为 $\left(\frac{\pi}{4} - 0.1\right)$，$\frac{\pi}{4}$ 和 $\frac{\pi}{2} - 0.3$。三角不等式 $\mathbf{d}(x, y) + \mathbf{d}(y, z) \geq \mathbf{d}(z, x)$，当 $\mathbf{d}(x, y) = 0.23, \mathbf{d}(y, z) = 0.29, \mathbf{d}(z, x) = 0.70$ 时，对于 $\mathbf{d}(\cdot, \cdot) = 1 - \langle \cdot, \cdot \rangle$ 不成立

[5]当核函数 $\mathcal{K}(p, q)$ 只与点 p 和 q 之间的（欧几里得）距离相关时被认为是翻译无关的。高斯的 RBF 核函数就是这样的一个翻译无关的核函数

欧氏距离为 0；一个点对于它自己的余弦相似度为 1。一个点 $x \in S$ 对于它自己的内积为 $\|x\|^2$，根据 $\|x\|$ 值的不同这个值可高可低。另外，可能存在很多其它的点 $y \in \S$ 使得 $\langle y, x\rangle < \|x\|^2$

高效地最近邻搜索方法就在很大程度上依赖这些特性（三角不等式和上述的巧合）来实现它的效率。因此，在没有任何外加假设的情况下，最大内积检索问题显然要比之前解决的类似问题更难。这也很可能就是为什么在我们已知范围内，没有任何已有工作来解决这个问题的一般形式的原因。

2.3 为什么使用树形结构

树状结构曾被广泛应用于最近邻搜索问题。因为这是个在最近邻的情况下被广泛运用的方法，我们相信在考虑最大内积搜索这个问题之前，回顾一下它们是很具有启发性的。

对于精确的最近邻搜索来说，只要存在低维有固维数，树状结构可以对于从低到高维的所有数据的效率产生巨大的提升。树状结构也可以很容易的被使用在一些非精确的情况下，且对于误差的范围有各种的保证。这就包括了一些在*排名*概念上的近似，也就是说，如果无法得到一个真正的最佳匹配，树状结构可以保证搜索的结果，好比说，是 10 个最佳匹配其中之一 – 而不是提供一个对于更加抽象的量，比如说距离（也就是 LSH 所提供的那样；如何拓展 LSH 算法来提供一个排名上的保证还不清楚），的保证。这看上去对于很多应用，比如说推荐算法，就有一些特定的意义。树形结构也可以在一些重要的实用应用的场景下被用作另一种近似的搜索，其中需要在用户定义的一个时限内找到可能的最佳匹配。这如果用像 LSH 那样的算法是不可能实现的 – LSH 提供了理论上的误差约束，然而*在搜索中*没有任何办法来限制这些误差。树的另一个重要的优势在于，树只需要一次建立 – “branch-and-bound”算法可以被适配进不同级别的近似以及时间限制中。然而基于哈希的方法对于不同的限制需要多次不同的哈希。通常的范式是对于很多近似的值进行预哈希。而且可以使用机器学习的方式，从数据中学习出一些模式来建树，来提供更好的准确性和效率。

对于最近邻的场景下，基于树状结构的方法体现出的显著优势，引发了一个问题，也就是是否可以将它们带入最大内积搜索的问题中去。

3 基于树的搜索

球树是一个“空间分割”的二叉树，它被十分广泛地应用于各种对数据集进行索引的任务中。这个树中的所有节点代表一个点的集合，每个节点用包围着这个点集的球的球心进行索引。一个节点上的点集被分成两个不相交的集合来形成两个子节点，把空间分割为（可能重叠的）两个超球。树是分层构建的，如果一个节点包含的点的大小小于一个阈值 N_0，那么这个节点就被作为叶子节点。

3.1 树的构建

我们用一个简单的启发式的建树方法，它先近似地选择两个相距最远的主元点，然后用把点分配给距离它们最近的主元的方式来分割数据。这种启发式背后的直觉在于这些两个点可能位于主方向上。这个基于分割和递归的完整方法在算法 1 和算法 2 中给出。

3.2 “Branch-and-Bound”算法

球树在最近邻搜索中被广泛地运用，而且它是已知的相对容易被拓展到高维的数据中的数据结

构。树的搜索通常会使用深度优先的“branch-and-bound”算法 – 对于一个查询，需要深度优先地遍历这棵树，首先遍历和查询点更接近的点的节点，然后用三角不等式限制住和别的分支可能的最小的距离的界。如果这个到这个分支的最小距离大于到目前的最近邻的距离，那么这个分支就会从计算中被移除。一个类似的贪心深度优先搜索可以被用与最大内积搜索中，不过对遍历的节点选择会最大化节点中可能存在的点和查询点最大内积，而不是选择遍历和查询点相距更近的节点。递归的“branch-and-bound”算法在算法 4 中给出。对于一个查询（q），算法从树的根部开始。在每一步中，算法都处在一个当前的节点（T），它会检查这个节点中所有的点和查询点的可能的最大内积，$\mathbf{MIP}(q, T)$，是否比当前对于查询的最佳匹配（$q.bm$）更好。如果这个检查失败，就不会再去遍历树的这个分支。否则，算法将递归地便利这棵树，深度优先地查看那些可能会有更好匹配的那些分支。如果这个节点是一个叶子节点，那么算法就只需要在这个节点中线性查找最佳的匹配。这个算法保证在结束时会给出一个精确的解（也就是最大内积）。

3.2.1 用球来限制最大内积

我们在上面提出了一个新颖方法，来计算一个给定点和球中的点的可能最大内积的解析上界（这个情况下也就是查询 q）。重要的是要注意，对于一个球的信息被完全限制在它的中心和半径上。在本节剩下的内容中，我们将用 $\|\cdot\|$ 的记号来表示 $\|\cdot\|_2$。

定理 3.1. *给定一个以 p_0 为中心，R_p 为半径的球 $\mathcal{B}_{p_0}^{R_p}$ 以及其之中的点，与一个（查询）点 q，q 和球 $\mathcal{B}_{p_0}^{R_p}$ 中点的可能的最大内积的上界为：*

$$\max_{p \in \mathcal{B}_{p_0}^{R_p}} \langle q, p \rangle \leq \langle q, p_0 \rangle + R_p \|q\| \tag{3}$$

证明. 假设对于球 $\mathcal{B}_{p_0}^{R_p}$ 中的查询 q，p^* 是可能的最佳匹配点，其中 r_p 为球心 p_0 与 p^* 之间的欧氏距离（根据定义，$r_p \leq R_p$）。θ_p 为向量 $\vec{p_0}$ 与向量 $\vec{p_0 p^*}$ 之间的夹角，ϕ 和 ω_p 分别为向量 $\vec{p_0}$ 与向量 $\vec{q}$，向量 $\vec{p^*}$ 之间的夹角（见图 5）。那么 p^* 的长度用 p_0 与 θ_p 表示也就是：

$$\|p^*\| = \sqrt{(\|p_0\| + r_p \cos\theta_p)^2 + (r_p \sin\theta_p)^2} \tag{4}$$

角 ω_p 可以用 p_0 与 θ_p 表示成：

$$\cos\omega_p = \frac{\|p_0\| + r_p \cos\theta_p}{\|p^*\|}, \sin\omega_p = \frac{r_p \sin\theta_p}{\|p^*\|} \tag{5}$$

令 θ_{q,p^*} 为向量 $\vec{q}$ 和 $\vec{p*}$ 间的娇娇，根据角之间的三角不等式，我们有：

$$|\theta_{q,p^*}| \geq |\phi - \omega_p|.$$

假定所有角都在 $[-\pi, \pi]$ 的范围内（而不是通常的 $[0, 2\pi]$），我们有：

$$\cos\theta_{q,p^*} \leq \cos(\phi - \omega_p) \tag{6}$$

用这个不等式我们可以得到下面这个 q 和任意 $p \in \mathcal{B}_{p_0}^{R_p}$ 可能的最大内积的上界

$$\max_{p \in \mathcal{B}_{p_0}^{R_p}} \langle q, p \rangle = \langle q, p^* \rangle (\text{by assumption}) = \|q\| \|p^*\| \cos\theta_{q,p^*}$$

根据等式 4,5,6，我们有：

$$\begin{aligned}
\max_{p \in \mathcal{B}_{p_0}^{R_p}} \langle q, p \rangle &\leq \|q\| \|p^*\| \cos(\phi - \omega_p) \\
&= \|q\| (\cos\phi(\|p_0\| + r_p \cos\theta_p) + \sin\phi(r_p \sin\theta_p)) \\
&\leq \|q\| \max_{\theta_p} (\cos\phi(\|p_0\| + r_p \cos\theta_p) + \sin\phi(r_p \sin\theta_p)) \\
&= \|q\| (\cos\phi(\|p_0\| + r_p \cos\phi) + \sin\phi(r_p \sin\phi)) \\
&\leq \|q\| (\cos\phi(\|p_0\| + r_p \cos\phi) + \sin\phi(R_p \sin\phi))
\end{aligned}$$

其中，第三个不等式来自最大值的定义。下面的等式来自对 θ_p 的放大，这就给了我们一个最优的值 $\theta_p = \phi$。最后的不等式是因为 $r_p \leq R_p$ 的事实。化简最后一个不等式就可以得到不等式 3 □

对于树的搜索算法（算法 4），我们就可以用下面这个式子作为 q 和节点 T 的最大可能内积：

$$\mathbf{MIP}(q, T) = \langle q, T.\mu \rangle + T.R\|q\|.$$

这个上界几乎可以在要求单个内积的同时计算出来（因为查询的范数可以在做搜索之前就计算出来）

4 基于双树的搜索

对于一组查询，树可以对每个查询分别做一次遍历。然而如果查询点的集合非常的大，那么一个提高查询效率的常见技巧就是把查询也用书来进行索引。就可以通过用双树算法对两棵树同时进行遍历来进行搜索。一个基本的想法就是把树遍历的代价均摊到每组类似的查询上。双树算法已经被应用于各种基于树的算法，如最近邻搜索，以及核密度估计，并且在理论上具有一定的运行效率保证。

4.1 双树的“branch-and-bound”算法

一个一般的双树算法在算法 6 中给出。和算法 4 类似，这个算法向下遍历参考集 S（$RTree$）。而且这个算法也同时向下遍历查询的集合 V（$QTree$），这就形成了一个四路的递归。在每一步，算法在 $QTree$ 的一个节点 Q 以及 $RTree$ 的节点 T，对于每个 Q，值 $Q.\lambda$ 代表 Q 中任意一个查询与它当前候选的最佳匹配的内积的最小值。如果这个值比任意一个 Q 中的查询和任意一个 T 中的参考点形成可能的最大内积，$\mathbf{MIP}(Q, T)$ 更大，那么这段递归就不再会继续进行。当这个算法同时处在两个树的叶节点时，它就可以通过对 $RTree$ 叶节点线性扫描来得到 $QTree$ 中每一个查询的最佳匹配。

我们将探索两种对查询进行索引的方式 – (1) 用球树来对查询进行索引（算法 7 中 $MakeQueryTree$ 是算法 2）(2) 用一个新颖的数据结构锥树来对查询进行索引（算法 7 中的 $MakeQueryTree$ 是算法 9）。在下面的小节中，我们会推导出适用于球树的表达式 $\mathbf{MIP}(Q, T)$。适用于锥树的表达式将在第五节中给出。

4.2 使用球树

在这个小节中，我们会以下面这个定理给出两个球之间内积的界。

定理 4.1. *给定两个分别以 p_0 与 q_0 为球心，R_p 与 R_q 为半径的球 $\mathcal{B}_{p_0}^{R_p}$ 以及 $\mathcal{B}_{q_0}^{R_q}$，一对点 $p \in \mathcal{B}_{p_0}^{R_p}$ 和 $q \in \mathcal{B}_{q_0}^{R_q}$ 之间的最大可能内积的上界为：*

$$\langle p_0, q_0 \rangle + R_p R_q + \|q_0\| R_p + \|q_0\| R_q. \qquad (7)$$

证明. 考虑一对点 $(p^*, q^*), p^* \in \mathcal{B}_{p_0}^{R_p}, q^* \in \mathcal{B}_{q_0}^{R_q}$ 使得：

$$\langle q^*, p^* \rangle = \max_{p^* \in \mathcal{B}_{p_0}^{R_p}, q^* \in \mathcal{B}_{q_0}^{R_q}} \langle q, p \rangle. \qquad (8)$$

令 θ_p 为 $\vec{p_0}$ 与 $p_0\vec{p}^*$ 之间的夹角，θ_q 为查询球中所对应的角。令 ω_p 为 $\vec{p_0}$ 和 $\vec{p^*}$ 之间的夹角，ω_q 为 $\vec{q_0}$ 和 $\vec{q^*}$ 之间的夹角。令 r_p 为 p_0 与 p^* 之间的距离，r_q 为 q_0 与 q^* 之间的距离。令 ϕ 为 p_0 与 q_0 在原点处的夹角。

对于球 $\mathcal{B}_{p_0}^{R_p}$ 有以下事实（对于球 $\mathcal{B}_{q_0}^{R_q}$ 也类

似)：

$$\|p^*\| = \sqrt{\|p_0\|^2 + r_p^2 + 2\|p_0\| r_p \cos\theta_p},$$
$$\cos\omega_p = \frac{\|p_0\| + r_p\cos\theta_p}{\|p^*\|}, \sin\omega_p = \frac{r_p \sin\theta_p}{\|p^*\|}$$

根据角之间的三角不等式，我们知道：

$$|\theta_{q^*,p^*}| \geq |\phi - (\omega_p + \omega_q)|,$$

我们就能得到下面这个等式：

$$\langle q^*, p^*\rangle = \|p^*\|\|q^*\|\cos(\phi - (\omega_p + \omega_q)). \quad (9)$$

用 θ_p 和 θ_q 来替换上面那个等式中的 ω_p 和 ω_q（和定理 3.1 的证明类似），我们有：

$$\begin{aligned}
\langle q^*, p^*\rangle =& \langle q_0, p_0\rangle + r_p r_q \cos(\phi - (\omega_p + \omega_q)) \\
&+ r_p\|q_0\|\cos(\phi - \theta_p) + r_q\|p_0\|\cos(\phi - \theta_q) \\
\leq& \max_{\theta_p,\theta_q} \langle q_0, p_0\rangle + r_p r_q \cos(\phi - (\omega_p + \omega_q)) \\
&+ r_p\|q_0\|\cos(\phi - \theta_p) + r_q\|p_0\|\cos(\phi - \theta_q) \\
\leq& \max_{\theta_p,\theta_q} \langle q_0, p_0\rangle + r_p r_q + r_p\|q_0\| + r_q\|p_0\| \\
\leq& \langle q_0, p_0\rangle + R_p R_q + R_p\|q_0\| + R_q\|p_0\|
\end{aligned}$$

第一个不等式来自对最大值的定义，第二个不等式来自 $\cos(\cdot) \leq 1$，最后一个不等式来自于 $r_p \leq R_p, r_q \leq R_q$ 的事实 □

对于双树的搜索算法（算法 6），两棵树的节点 Q 与 T 的最大可能内积为：

$$\mathbf{MIP}(Q,T) = \langle q_0, p_0\rangle + R_p R_q + R_p\|q_0\| + R_q\|p_0\|$$

有趣的是，当包含查询点的球中只有一个点时，也就是 $R_q = 0$ 时，这个上界就退化成定理 3.1 的上界了。

5 锥树

在等式 1 中，取得内积最大值的点 p，与查询点 q 的范数 $\|q\|$ 无关。令 $\theta_{q,r}$ 为 q 和 r 在原点处的夹角，那么搜索最大内积的任务也就是找到一个点 $p \in S$ 使得：

$$p = \arg\max_{r\in S} \|r\|\cos\theta_{q,r} \quad (10)$$

这也就是说只有查询点的方向会影响结果。每个球提供了内积的一个界是因为它们同时限制了向量的范数和方向。然而因为范数跟查询没有关系，就不需要在球中索引它了（也就是限制它的范数），只有它们的方向的范围需要被限制。基于这个原因，我们提出基于查询点的方向（从原点出发）来对查询进行索引，来形成一个锥树。这些查询点用一些（可能重叠）的开放的锥形来进行分级索引。每个点都表示成向量，对应它们的中轴，以及一个角度，对应它们的孔径 [6]。

5.1 锥树的建立

锥树的建立和球树的建立十分类似。它们之间唯一的区别就是在分割点时使用了余弦相似度而不是欧氏距离。（详见图 8 的伪代码）

5.2 锥形球的界

因为查询的范数不会影响等式 10 中的结果，我们假定所有的查询都具有单位范数。

定理 5.1. *给定一个以 p_0 为球心，R_p 为半径的球 $\mathcal{B}_{p_0}^{R_p}$ 与其中的点，已经一个以 q_0 为中轴，孔径 $2\omega_q \geq 0$ 的锥 $\mathcal{C}_{q_0}^{\omega_q}$，任意一组点 $p \in \mathcal{B}_{p_0}^{R_p}, q \in \mathcal{C}_{q_0}^{\omega_q}$*

[6] 一个锥的孔径是中轴和边缘夹角的两倍

之间可能的最大内积的上界为：

$$\|p_0\|\cos(\{|\phi|-\omega_q\}_+)+R_p \tag{11}$$

其中 ϕ 是 p_0 与 q_0 在原点处的夹角，函数 $\{x\}_+=\max\{x,0\}$。

证明. 这里有两种需要考虑的情况：

i $|\phi|<\omega_q$

ii $|\phi|\geq\omega_q$

对于情况（i），球 $\mathcal{B}_{p_0}^{R^p}$ 的球心 p_0 在锥 $\mathcal{C}_{q_0}^{\omega_q}$ 之内，也就是说：

$$\max_{q\in\mathcal{C}_{q_0}^{\omega_q},p\in\mathcal{B}_{p_0}^{R_p}}\|p\|\cos\theta_{q,p}\leq\|p_0\|+R_p \tag{12}$$

因为有一些查询 $q^*\in\mathcal{C}_{q_0}^{\omega_q}$ 和 p_0 的方向相同，就直接给出最大的可能内积。

对于情况（ii），我们假定 $\phi\geq 0$ 不会丧失这个问题的一般性。那么 $\phi\geq\omega_q$。继续使用定理 3.1 和 4.1 中表达最佳的点对 (q^*,p^*)，以及图 9 中的记号，我们可以说：

$$|\theta_{p^*,q^*}|\geq|\phi-\omega_q-\omega_p| \tag{13}$$

因为 ω_q 是固定的，我们就可以继续说：

$$\begin{aligned}\max_{q\in\mathcal{C}_{q_0}^{\omega_q},p\in\mathcal{B}_{p_0}^{R_p}}\|p\|\cos\theta_{q,p}&\leq\|p^*\|\cos\theta_{q^*,p^*}(\text{by def})\\&\leq\|p^*\|\cos(\phi-\omega_q-\omega_p).\end{aligned} \tag{14}$$

用 $\|p_0\|$，r_p 和 θ_p 来表示 $\|p^*\|$ 和 ω_p，接下来最大化 θ_p，用 $r_p\leq R_p$ 这些事实，我们可以得到：

$$\max_{q\in\mathcal{C}_{q_0}^{\omega_q},p\in\mathcal{B}_{p_0}^{R_p}}\|p\|\cos\theta_{q,p}\leq\|p_0\|\cos(\phi-\omega_q)+R_p \tag{15}$$

把情况（i）和（ii）结合起来，我们就得到了式 11。 □

6 实验与结果

在这一节，我们将对算法 5（SB - Single Ball Tree）和算法 7 的效率进行评估。对于双树的算法，我们用了它的两个变形 – i 用球树来索引查询点（DBB – Dual ball-ball）ii 用锥树来索引查询点（DBC – Dual ball-cone）。我们把我们提出的算法和在算法 3 中给出的线性搜索（LS – linear search）进行了比较。我们将报告我们的算法相对于线性搜索的加速比 [7]。对于树来说，叶子中含有点的数量 N_0 可以通过交叉验证的方式来选择，不过我们对所有的数据集专门选了一个值，$N_0=20$，来说明用我们的算法得到的效率的提升不需要任何复杂的交叉验证。

数据集 我们使用了各种来自不同的数据挖掘的领域的数据集。我们用了以下合作公司的数据集：MovieLens，Netflix 和 Yahoo! Music 数据集。对于文本数据，我们使用了 LiveJournal Blog moods 数据集。我们同样使用了 MNIST digits 数据集来做评估。我们同样考虑了三个天文方面的数据集，LCDM，PSF 和 SJ2。我们还用了一个用 20 维随机分布的点合成数据集（U-Rand）。其他数据集都是在机器学习中被广泛运用的，来自 UCI 机器学习仓库。数据集的细节由表 1 给出 [8]。

建树时间 建树的效率是非常高的。我们在表 6 中给出了建树时间并把他们和线性搜索的运行时间进行了对比。在最后一列中，我们展示了建树时间和算法 3 中运行时间的对比。对于单个球和两

[7] 加速比被定义为是线性搜索使用的时间和被评估的算法所使用的时间的比值

[8] 对于合作筛选的数据集，参考点（项目）和查询（用户）是被明确定义的，我们随机地从数据集中选出查询集（V），剩下的作为参考集（S）

个球的树算法，建树中分别包括了构造一个和两个球树。对于“dual ball-cone”的算法，为了方便起见，所有查询都被标准化为单位长度 [9]。在标准化查询之后，会建立两个树。标准化查询的时间被要求包括在建树的时间中，这就使“dual ball-ball”和“dual ball-cone”算法的建树时间有显著的差别。

表 6（R）最后一列的数字告诉我们建树时间相对于真正线性查找的运行时间是有多小。最高的比例只有 OptDigits 数据集的 0.15。这就代表查找时只要比线性查找快 1.18 倍就可以弥补建树的代价。对于更多的数据集，这个比例还要更小。而且，建树的代价只有一次。当一棵树已经建好之后，它就可以多次地被用作查询。

查找效率 相对于线性查找，查找效率的提升在表 6 中给出。总体来说，加速比从低至 OptDigits 的 1.13 到 LCDM 和 PSF 数据集的 10^5（4 个数量级）那么多。重要的是，需要注意所有加速比很低的（比一个数量级还小）使用单个球的算法，三个算法的加速比都十分低，相对还是具有可比性的。然而，甚至加速比达到 2 了之后对于绝对的速度提升也是十分显著的。比如对于 Yahoo!Music 数据集，2 的加速比加上 120 秒的建树时间节省了总共 19 小时的计算时间。对于绝大多数对于单个球的算法具有很高加速比的数据集，双树算法的加速比也非常高。

还有三件非常重要的事需要注意。首先，如单个树的算法（算法 5）加速比不高的话，双树算法（算法 7）也不会有特别好的表现。这很大程度上是因为树无法找到非常严格的界，所以需要遍历非常多的分支。双树的算法放松了界限来把遍历的代价均摊到每个查询上。但是如果这个界对于算法 5 来说比较差，对于双树的界就会更差。因此，双树算法没有表现出任何显著的速度提升。第二，当查询的数据集数量非常大的时候，双树算法（尤其是“dual ball-cone”算法）就开始显著地单个树算法更优秀。这对于双树算法来说是一个很普遍的状况。查询集一定要足够大，才能使遍历参考树的效率均摊到查询上之后，超过遍历查询树本身的开销。最后，在一般情况下，使用球树的双树算法会显著地比使用锥树来做查询要更慢。这很可能是因为这两个原因 – i 锥形对于查询提供了一个更紧密的索引，单个锥就可以索引很多个方向相近但是范数不同的球。ii 等式 10 中的 $\mathbf{MIP}(Q,T)$ 是相当松的一个上界。。

我们同样考虑了一个更一般的问题，也就是对于查询 q 检索在集合 S 中第 k 大的内积的点。这和第 k 近邻搜索问题十分类似，我们同样对我们的算法在图 11 中给出了当 $k = 1$，2，5 和 10 时，相对于线性查找的加速比。

7 最大核操作和一般的核函数

在这一节中，我们提供了一些讨论，关于如何在不显式地把点表示成内积空间中的条件下，把前面提出的那些算法应用到一个内积空间中。内积的定义是一个核函数 $\mathcal{K}(q,p) = \langle\varphi(q),\varphi(p)\rangle$.

要在内积空间中做这件事，我们就要修改一下建树的过程。对于一个节点 T 和它的点集 $T.S$，在 φ-空间中的均值被定义为 $\mu = \frac{1}{|T.S|}\sum_{p\in T.S}\varphi(p)$。然而，$\mu$ 不一定有一个显式的表示形式，但是有可能用下面的方法计算出与 μ 的内积：

$$\langle\mu,\varphi(q)\rangle = \frac{1}{|T.S|}\sum_{p\in T.S}\mathcal{K}(q,p).$$

然而计算这个式子的开销在搜索过程中可能就会

[9] 这是因为查询点的范数不会影响结果

显得非常大。因此我们认为，选择一个在 φ-空间中，最接近均值 μ 的那个点当做新的球心。所以新的球心 p_c 可以这样给出：

$$p_c = \arg\min_{r\in T.S} \mathcal{K}(r,r) - \frac{2}{T.S}\sum_{r'\in T.S}\mathcal{K}(r',r). \quad (16)$$

这个操作在计算上是二次的，不过可以在预处理阶段做这件事来获得查询阶段效率的提升。对于一个球心 p_c，包围住 φ-空间中的点集 $T.S$ 的球的半径 R_p 可以由下式给出：

$$R_p^2 = \max_{r\in T.S} \mathcal{K}(p_c,p_c) + \mathcal{K}(r,r) - 2\mathcal{K}(r,p_c) \quad (17)$$

由这些对于球心和半径的定义，一个球树用算法 2 在任意的 φ-空间中建立。给定一个 φ 空间中的球，定理 3.1 中的等式 3 就变为：

$$\mathbf{MIP}(q,T) = \mathcal{K}(q,p_c) + R_p\sqrt{\mathcal{K}(q,q)}. \quad (18)$$

这个上界的计算等价于单个核函数的求值（$\mathcal{K}(q,q)$ 可以在搜索之前提前计算出来）。用这个上界，树的搜索算法（算法 5）就可以在任意的 φ 空间进行。我们在这篇论文一个较长的版本中对这个算法进行评估。

用同样的原则，双树的算法（算法 7）也可以被应用到任意的 φ 空间中。对于使用球树来进行查询的双树，定理 4.1 中查询节点 Q 和节点 T 的最大可能内积的上界可以被修改成：

$$\mathcal{K}(q_c,p_c) + R_pR_q + R_q\sqrt{\mathcal{K}(q_c,q_c)} + R_q\sqrt{\mathcal{K}(p_c,p_c)} \quad (19)$$

其中 p_c 和 q_c 是在 φ-空间中的球心，它们的半径分别为 R_p 和 R_q

对于使用锥树进行索引的查询，在 φ-空间中，锥的中轴点可以被选为和 φ-空间的均值夹角最小的点。因为在 φ-空间中的查询也应当被标准化，对于一个查询树的节点 Q，集合 $Q.S$ 的均值应当为 $\mu = \frac{1}{Q.S}\sum_{q\in Q.S}\frac{\varphi(q)}{\|\varphi(q)\|}$，所以新的锥中轴点 q_c 为：

$$q_c = \arg\max_{q\in Q.S} \frac{\sum_{q'\in Q.S}\frac{\mathcal{K}(q',q)}{\sqrt{\mathcal{K}(q',q')}}}{\mathcal{K}(r,r)} \quad (20)$$

再次，这个计算时间对于数据集的大小来说是二次的，但是可以给搜索时间提高效率。半孔径的余弦值为：

$$\cos\omega_q = \min_{q\in Q.S} \frac{\mathcal{K}(q_c,q)}{\sqrt{\mathcal{K}(q_c,q_c)\mathcal{K}(q,q)}} \quad (21)$$

定理 5.1 对于锥树查询节点 Q 和球树参考节点 T 的上界就变为：

$$\sqrt{\mathcal{K}(p_c,p_c)}\cos(\{|\phi| - \omega_q\}_+) + R_p \quad (22)$$

其中 ϕ 的定义为：

$$\cos\phi = \frac{\mathcal{K}(p_c,q_c)}{\sqrt{\mathcal{K}(q_c,q_c)\mathcal{K}(p_c,p_c)}}$$

这个界的计算效率是很高的，是因为他只需要一个核函数的求职（$\mathcal{K}(q_c,q_c)$ 和 $\mathcal{K}(p_c,p_c)$ 两项可以被提前计算出来并且存在树中）。

8 结论

我们考虑了一般的最大内积搜索的问题，而且用几个新颖的基于树的方法高效地解决了这个问题。我们用树形结构，并且用了“branch-and-bound”算法来进行最大内积搜索。我们还展示了一个双树算法来应对多个请求的状况。我们用了各种的数据集，对我们的算法进行了评估，并且展现出算法在计算上的效率。

对于我们的算法一个理论上的分析会让我们更好地理解这些算法的效率。对我们算法运行时间进行严格的分析是我们未来在这个方向工作的一部分。

9　引用

略