# Efficient similarity search and classification via rank aggregation

## 摘要：

我们推出了一种新的可以在高维数据中进行高效的相似性查询和分类的方法。在这个框架中，数据库的元素是表现为欧几里得空间中的向量。在同一个空间中，给定一个查询向量，我们的目标是找到数据库中和和该查询相似的元素。在我们的方法中，少量相互独立的“voters”会基于数据库元素和查询的相似性，给数据库中的元素进行排名。这些排名是通过高效的聚集（aggreation）算法来进行组合的。我们的方法论引出了近似最相邻元素的计算技巧和多种可供替换的计算最相邻元素方法。

我们的方法使用的一个例子如下。每个 voter 会将所有的向量（数据库中的元素和查询）投影到一条随机的线上（每个 voter 对应的 line 都不同），然后计算基于数据库元素在线上的投影和查询在线上的投影的距离来进行排名。而之后的聚集规则会挑选出 median rank 得分最高的元素。这些聚集有如下的几个吸引人的特性。在理论上，我们证明了，在很高的概率上我们方法选出来的元素会是一个带（1+Ɛ）因子 的在欧氏距离下的最近相邻。在实用方面，这方法很高效，通过探查不多于 5%的数据库数据获得一个很高质量的结果。同时这个方法是数据库友好的，它访问主要通过预定好的顺序进行非随机访问。而且，它不同于其它的寻找近似的最相邻的方法，我们的方法不用额外的存储空间。还有我们扩展了我们的方法使得可以解决 k-最相邻问题。

我们进行了两个实验样例来检验我们方法的有效性。我们的方法包含了两个最相邻问题的应用场景-----相似性查询和分类问题。在这两实验中，我们研究了我们方法的性能，它们在查询的质量和查询的效率上都表现优异。

## 1.介绍：

最邻近问题是广泛应用在很多计算机科学额领域。不正式的讲，在一个度量空间上给定一个包含了 n 个点（元素）的数据库 D，还有一个查询。找出 1(或者 k)个最相邻于查询的点。一些有名的最相邻应用有信息检索，模式识别，数据分析等等。最相邻问题之所以受欢迎的原因是由于把现实世界中的物体映射到度量空间的向量中是较为简单和自然的。所以像相似性查询和分类的问题就变成了最相邻问题。由于映射物体到特征向量中经常是一个启发式的步骤，在许多的应用中满足于在数据中查找一个点的最相邻点。这些问题促成了一些很精妙的计算问题。这里有大量的相关文献关于如何高效地解决最相邻问题。(。。。)

在这篇论文中，我们推出了一个创新性的方法用来进行相似性查询和分类问题，或者其他基于最相邻问题的应用。我们的方法是建立在两个方式上的：1.排名聚集（rank aggregation）2.个例最优（instance optimal）。我们的方法满足下面的两个需求，即使他们可能是相矛盾的。1.鲁棒性.2.保证算法的搞笑和数据库友好。

我们工作是在下面这个简单的 idea 下开始的，假设我们正在一个包含 n 个元素的数据库 D 中对一个查询 q 进行最相似检索。d 维的空间记为 $X^d$（X 是二值集合{0，1}…），所以每个元素都映射为该空间上的一个点，而 d 个坐标就对应我们之前提到的 voters，n 个点就是候选人

（candidates）。Voter j (1 <= j <= d)为 n 个点进行基于在坐标轴 j 下和查询的距离进行排名。然后我们就会得到 d 条候选人的排名链表。我们的目标就是综合这些排名，生成一个单一的排名。我们对最终得排名的靠前的元素感兴趣。

排名聚集问题就是一个如何聚集 d 条由 d 个坐标生成的连。这个问题的历史可以回溯到至少两个世纪之前，但是在数学上的深入理解是发生上 60 年前，其底层的计算方法现在任然是一个很活跃的研究。排名聚集的最重要的数学问题是确立一个鲁棒的聚集方法。在这方面做得最好的是 Young 和 Levenglick。他们做出了一个的聚集机制。例如：它满足 Condorcet criterion，Condorcet criterion 说的是如果存在一个候选人 c，对于其他的候选人，如果大多数 vators 都更加喜爱 c，那么 c 就是这次选举的获胜者。满足 Condorcet criterion 的聚集机制是避免了候选人会由于被一些糟糕的 vators 的差评而否决掉，提高鲁棒性。

Kemeny 的提议如下:给定 n 个候选人，和 d 个这些候选人的排列 f1，f2… $f_d$ 产生排列可以最小化 $\sum_{i=1}^{d} K(\tau_i,\sigma),$ 的 σ 期中 $K(\tau,\sigma)$ 表示 Kendall tau 距离， 也就是 f 和 σ 对 $(c,\tilde{c})$ 排名不相同的数目.我们把这个叫做 Kendall-optimal aggregation。但是不幸的是单单计算 4 条链 Kendall-optimal aggregation 就是一个 NP 完全的问题了。所以一定要找近似的算法或启发式的算法。

让我们来阐述一下最相邻 entire 和排名聚集的联系。作为一个简单又有力的一个例子，注意到如果我们的空间是{0，1}$^d$ 赋值以汉明度量，那么每个 voter 会产生一个偏序，给定一个查询 q，第 i 个 voter 切分整个数据库为两份

$D_i^+ = \{x \in D \mid x_i = q_i\}$ and $D_i^- = \{x \in D \mid x_i \neq q_i\},$ 排名会让 $D_i^+$ 高于 $D_i^-$

这不难看出 Kendall-optimal aggregation 会把上面 voter 长生的排名聚集成了他们对查询 q 的汉明距离了。还可以考虑这样的一个事实，最相邻的问题在一些有趣的度量下会退化成汉明度量，我们可以看到，排名聚集至少有最相邻一样的能力。(我们会在后面提供一些更有说服力的例子)

在另外一方面，我们可以把最相邻问题通过直接的算法在 O(nd)时间内解决，再重新让它回变成一个 NP 问题。即使有了一些好的近似算法和启发式可以用于聚集问题。但它们至少 $\Omega(nd + n^2)$ 的复杂度。幸好，有两个关键的因素将我们从这里解救出来。1.我们只是仅仅对排名靠卡的 k 个元素感兴趣而不是全部排名。2.找到 k 个在聚集获胜的获胜者中，一个启发式的 median ranks 有高效的实现。我们下面探讨。

# 1.1Median rank aggregation

虽然计算 Kendall-optimalaggregations 没希望获得高效的算法，但是有一个可以在多项式时间内进行排序产生 factor-2 近似于 Kendall-optimal 的排序结果的方法叫 footrule。而且 footrule-optimal aggregation 有以下的好的启发性，我们称之为 median rank aggregation: 基于在 voters 中投票产生数据库中的点的中位数，对所有的点进行排序。这是一个合理的启发，因为如果 median ranks 是全部不相同的那么它产生的结果事实上是就是完成了了 footrule-optimal-aggregation。应此我们让我们的问题简化为了找出数据库中有最好 median rank 的点（或者前 k 个）。

我们不把 median rank aggregation 仅仅视为 Kendall-optimal aggregation 的启发式近似，我们认为它是一个自然的排名聚集方法。我们将会在 2.1.1 节中展示 median rank aggregation 给出了一个对应距离相似于 footrule 距离的最优解。另外，median rank aggregation 有如下的两个令人满意的性质，我们将会在下面详细讲解。

**数据库友好和个例最优化算法。**一个强烈推荐使用 median rank aggregation 的理由是它对数据库的友好性。特别的，我们希望推出一个有适合于数据库的性质的最相邻问题的（近似）解。理想地说，我们不希望出现复杂的数据结构，很大的储存空间的需求和大量的随机访问。例如：这些考虑马上就排除和很多理论上很好的方法，即使是最近的数据库的论文，都被这些考虑拖累着。相反，median rank aggregation 仅仅使用排序这个预处理步骤。不需额外的存储空间，没有随机访问。通过避免了随机访问我们的方法可以不需要”指针”来定位元素的坐标值。

我们现在讨论 median rank aggregation 的高效性。让我们根据 d 个坐标轴分别预先排序好 n 个数据库中的点。给定一个查询 q=（$q_1$ ….$q_d$ ）我们可以很简单地在预先排序好的（1<=i<=d）第 I 条链上定位 $q_i$ 的值，同时放置两个游标。一旦 2d 个游标放置好了，也就是对每个 I 有对应的 2 个游标，通过分别上下移动两个游标，我们可以产生一个可以依次产生一个在第 i 个 voter 下选出的候选人的流。我们像下面这样考虑 d 个 voter 在做下面的在线操作: 当第 i 个 voter 在第一次被调用的时候，会产生在 i 轴上与查询 q 在 i 轴上距离最近的元素，第二次被调用则产生距离次近的元素，如此类推。因此我们有一个在线版本的聚集问题要解决。

事实上，我们可以通过上面简单的调用产生第 I 个 voter 产生的流和最高的 median rank 一起来产生最终的获胜者，这暗示着有可能不用扫描全部 d 个排名链表来确定优胜者。确实，通过一个顺序或随机访问的”最优“步数来计算得分链表的聚集操作吸引了很多研究人员。我们设计了一个基于 NRA(无随机访问)的算法。这个算法应用在 online median-rank-winner 问题，它可以被干脆地一句话描述。*访问 d 个 voters 生成的排名链表，每链表一次一个元素，当一个元素出现在多于一半的链表中时，它就是获胜者。*我们把这个算法称之为 MEDRANK 算法。我们会表面 MEDRANK 不仅仅是一个好的算法，同时它是一个单例最优的算法。也就是说相对与算法最好的情形，每个情形的复杂度都只是常量于最好情形的倍数。我们泛化 MEDRANK 算法去寻找最高的 k 个元素。例如：在一个获胜者找出来以后，我就接着找下一个出现在一半的 voter 链表中的元素。

**近似最相邻借。**Median rank aggregation 可以和另外一个有力量的 idea 结合在一起。这个 idea 就是沿着随机的线投影。我们会在 Section 2 中通过一个简单的几何引理，表明如果 我们投影 n 个数据库的点（也包括查询）到 m 维($m = O(\epsilon^{-2} \log n)$) 然后运行我们的 MEDRANK，选出来的获胜者会在很大的概率上是在欧氏度量上相对于查询 q 的 ε 近似最相邻解。（我们说查询 q 的 ε 近似最相邻解意思是对于每个 $\bar{c} \in D$, we have $d(c, q) \leq (1 + \epsilon)d(\bar{c}, q)$）

# 1.2 Rank aggregation vs. nearest neighbors

我们感受到排名聚集是一个对于相似性检索，分类问题是个新的鲁棒的范式。就像我们之前提到的，这是一个可以至少和最相邻一样强大，同时它具有非常高效的实现。现在我们点出另外一个在数据库上 rank aggregation 优于 nearest neighbors 的点。考虑相似性检索问题某个元素不能很自然地被某种度量空间所表达，例如目录应用，那里的向量的 features 可能是按类别的（例

如颜色）。或者是数值问题，不同的坐标的数值单位不匹配。在这种情况下，将元素映射到度量空间上的点就很成问题了。而 rank aggregation 范式则是很好地解决了这个问题，当看某一个元素和查询的相似性的时候，我们就仅仅针对一个 feature 来进行排名。分类的搜索是数据库的常见操作，我们的算法 MEDRANK 很合适地实现了，并且是一个高效率和有效的解法。

# 1.3 Organization

论文剩下的部分是按下面的结构来安排的。Section 2 展示了 MEDRANK 和相关算法的正式描述，Section 3 是描述了我们的实验和实验的分析，我们的实验包含了最相邻问题的主要应用—相似性检索和分类。在两个实验中我们展示了聚集的方法在检索质量和效率上有很好的效果。在 Section 4 中我们会总结整篇论文。

# 2.FRAMEWRK AND ALGORITHMS

在这节的第一部分，我们会描绘整个框架，包括必要的关于排名聚集的准备知识和单例优化算法。这里主要有两个技术方面在这节中：(1) 转化欧氏 ε 近似最相邻问题到通过最好的 median rank 来选取候选人(n 个候选人，$O(\epsilon^{-2}\log n)$ 个 voter)。(2) 证明只使用顺序访问的 MEDRANK 算法会最多比使用顺序和随机访问的方法多耗时常数倍的时间。因此 MEDRANK 是一个在数据库模型上计算 median 获胜者的单例最优算法，同时可以产生一个可证明是近似最相邻的结果。

# 2.1 Rank aggregation, nearest neighbors, and instance optimal algorithms

## 2.1.1 Preliminaries

让 σ 和 τ 表示 n 个元素的排列；σ(i)表示元素 i 在 σ 中的排名(小的值代表排名越高)，我们会表示元素 i 比 j 排名高成 σ(i)< σ(j)，τ 和 σ 的 Kendall tau 距离记为 K(σ，τ)，它是被定义为满足下面其中任意一种情况的(i，j)对的数目：(1) σ(i)< σ(j), τ(i)> τ(j)  (2) σ(i)> σ(j), τ(i)< τ(j)

而 τ 和 σ 的 footrule 距离则表示为 F(τ,σ)，定义是$\sum_i |\sigma(i) - \tau(i)|$

让 $\tau_1$ ，$\tau_2$ …$\tau_m$ 表示对于 n 个元素 m 个排列。而 $\tau_1$ ，$\tau_2$ …$\tau_m$ 的 Kendall optimal 聚集是任何这样一个排列 σ 使得$\sum_i K(\sigma, \tau_i)$ 最小，相似的 $\tau_1$ ，$\tau_2$ …$\tau_m$ 的 footrule-optimal 聚集是使得 $\sum_i F(\sigma, \tau_i)$ 最小的任意一个排列 σ，

这已经知道　K(σ，τ) <= F(σ，τ)<= 2K(σ，τ).而计算 Kendall optimal 聚集是 NP-hard 的而计算 footrule-optimal 聚集是可以在多项式时间内完成的给定一组 $\tau_1$ ，$\tau_2$ …$\tau_m$ ，我们定义 $\mathrm{medrank}(i) = \mathrm{median}(\tau_1(i), \ldots, \tau_m(i))$ 。因此，medrank 对每个元素赋值以它们在各

排列中的中值。下面是一个简单的数学命题，它指出了在很多的情况下，median rank aggregation 给出了 footrule-optimal aggregation

PROPOSITION 1.让 $\tau_1$ ， $\tau_2$ ...$\tau_m$ 表示对于 n 个元素 m 个排列，如果中值 medrank(i)是全部不同的，那么 medrank 是一个和和 $\tau_1$ ， $\tau_2$ ...$\tau_m$ 的 footrule-optimal aggregation 一样的排列

即使 median ranks 不是相互不同的，下面的一条命题说明了 median rank aggregation 给出了一个类似于 footrule 距离的最优解。让 f 表示一个为每个元素赋值的函数，令 τ 代表一个排列，令 τ 表示一个排列，它们都是相对于同一个元素集合。定义 M(f, τ)= $\sum_i |f(i) - \tau(i)|$，和是对于每一个元素 i 而言的。因此 M 是相似于 footrule 距离的，除了 f 是一个赋值的函数，而不是一个排列。

PROPOSITION 2. .让 $\tau_1$ ， $\tau_2$ ...$\tau_m$ 表示对于 n 个元素 m 个排列，那么 medrank 是一个函数 f 使得 $\sum_{j=1}^{m} M(f, \tau_j)$ 最小

PROOF.我们希望最小化 $\sum_{j=1}^{m} M(f, \tau_j)$ = $\sum_{j=1}^{m} \sum_i |f(i) - \tau_j(i)| = \sum_i \sum_{j=1}^{m} |f(i) - \tau_j(i)|$

这是很清晰最后一个连加式是只与一个 i 有关的，所以我们可以求出每个 i 对应的 f(i)值来达到全局最小，很显然，当 f(i)取 median($\tau_1$ ， $\tau_2$ ...$\tau_m$)的值时可以达到这样的效果。

令 D 代表在 $R^d$ 具有 n 个点的数据库，对于向量 q. 一个在数据库 D 中 q 的欧氏最相邻点是这样的一个点 x，对于所有属于数据库 D 的点 y 有 d(x,q) <= d(y,q)，d 表示欧氏距离。下面阐述 ε-approximate 欧氏最相邻点的定义。一个在数据库 D 中 q 的欧氏 ε 近似最相邻点是这样的一个点 x，对于所有属于数据库 D 的点 y 有 d(x,q) <= (1+ε)d(y,q).

## 2.1.2 An algorithm for near neighbors

将数据投影到随机选取得线上的思想是被 Kleinberg 在最邻近检索中推出来的。更详细地说，考虑一个点 $q \in \mathbf{R}^d$ 和 $u, v \in \mathbf{R}^d$ 满足 d(v,q)>(1+ε)d(u,q)。假设我们随机地挑选一个 d 维的单位向量 r。一个高效的做法是对 d 个坐标 $r_1$.....$r_d$ 通过独立同分布的高斯分布 N(0,1)来产生随机数，之后再将其规范化。我们然后就将 u，v，q 投影到 r 上。令<. , .>记为普通的内积。直观上地来说，我们希望投影<u,r>会比<v,r>离<q,r>更加近，也就是<u-q,r>小于<v-q,r>。下面的引理给出了正式的表达。

LEMMA 3 假设 $x, y \in \mathbf{R}^d$，且令 ε > 0 满足 $||y|| > (1+\epsilon)||x||$，如果 r 是一个随机的单位向量(通过上面的方式来产生)，那么 $\Pr[\langle y, r\rangle \leq \langle x, r\rangle] \leq 1/2 - \epsilon/3$

通过令 x = u-q 和 y=v-q 这会推导出至少在 $1/2 + \epsilon/3$ 的概率下 <u-1,r>小于<v-q,r>

现在令 q 为查询点，令 $w \in D$ 是 q 在最相邻点，令 $B = \{x \in D \mid d(x,q) > (1+\epsilon)d(w,q)\}$ 考虑一个固定的值 x 属于 B。如果我们随机地选取一个向量 r，然后通过 D 中的点在 r 上投影和 q 在 r 上投影的距离来给所有的点排名，那么 w 是至少以概率 $1/2+\epsilon/3$ 比 x 排名靠前，假设我们随机地选取多个向量 $r_1.....r_m$ 然后按如下的方法创建 m 个在 D 上的排名链表:第 j 个排名链表是通过在 $r_j$ 上的投影和 q 的的投影得到的。那么 w 在 $r_1.....r_m$ 中排名超过其它 B 上的 x 点的期望至少超过 $m(1/2+\epsilon/3)$，确实通过标准的 Chernoff bounds，如果 $m = \alpha\epsilon^{-2}\log n$ （其中 α 恰当地选取的话）那么至少有（1-1/$n^2$ ）的概率 w 排名在多于 $m(1/2+\epsilon/3)$ 条链上比每个 B 上的 x 大，特别地，w 在 m 条链上的 median rank 是比其他 x 的 median rank 要高。因此如果我们计算的点 $z \in D$ 有在 m 条链上最高的 median 排名，那么我们就会以至少（1-1/n）的概率得到不在 B 上的 z，所以这满足 $d(z,q) \le (1+\epsilon)d(w,q)$，我们将上面的讨论写成下面的总结。

定理 4. 令 D 代表一个在 $R^d$ 上的 n 个点的集合。让 $r_1.....r_m$ 表示 $R^d$ 上的随机单位向量，$m = \alpha\epsilon^{-2}\log n$ (a 通过恰当地选取)令 q 属于 $R^d$ 的任意一个点，定义 m 条链表 $L_i$ (0<=i<=m)，$L_i$ 上的排序是根据 D 中的点到 $r_i$ 上的投影距离 q 到 $r_i$ 上的投影的大小以升序来排名。对每一个 D 上的 x，令 medrank(x) – median($L_1$ (x),…, $L_m$ (x)),令 z 成为在 D 中 medrank(z)最小的值，那么对所有 D 上的 x $d(z,q) \le (1+\epsilon)d(w,q)$ 会以(1-1/n)的概率成立。

事实上，上面的论述表示的更多的东西，令 q 代表一个查询，令 $w \in D$ 是 q 在最相邻点，令 $w_2 \in D$ 代表第二个 q 的最相邻点，类似于 B，我们定义 $B_2$ $B_2 = \{x \in D \mid d(x,q) > (1+\epsilon)d(w_2,q)\}$ (注意到 $B_2 \subseteq B$)通过相似的论证我们可以推导出，有很高的概率 $w_2$ 比其他的 $B_2$ 上的元素之 median rank 要高，这隐含着元素 $z_2$ 如果是 median rank 的排名第二高的元素，肯定会满足 $d(z_2,q) \le (1+\epsilon)d(w_2,q)$ 类似地对于任意的常数 k，我们都会得到 $z_3$ …$z_k$ 通过 3 到 k 个 median 排名最高元素满足 $d(z_j,q) \le (1+\epsilon)d(w_j,q)$ $w_j$ 代表了 q 的 第 j 个最相邻点。

对于实现的目的，我们不能给每个查询都给数据库中的 n 个点排序 m 次，所以我们会在预处理阶段创建 m 个排好序的链表。排序是按照元素到 r 上的投影的模。 第 i 个排序好的链表的形式 $(c_1^i, v_1^i), (c_2^i, v_2^i), \ldots, (c_n^i, v_n^i)$，其中 $v_t^i = \langle c_t^i, r_i \rangle$。对于每个 t，有 $v_1^i \le v_2^i \le \ldots v_n^i$，而 $c_1^i, \ldots, c_n^i$ 是 1，2，..n 的一个排列，给定一个 $q \in \mathbf{R}^d$，我们首先计算 q 在 m 个向量中的投影，对于每个 i，我们在第 i 个排好序的链表中定位< $r_i$ ，q>，也就是找到这样的一个 t 使得 $v_t^i \le \langle r_i, q \rangle \le v_{t+1}^i$ 成立，同时初始化两个游标 $v_t^i$ , $v_{t+1}^i$

现在$c_t^i$ and $c_{t+1}^i$ 中的其中一个点就是投影下距离 q 投影的最近的点了(这也是唯一一步用到随机访问的步骤)通过合适地移动两个游标，我们就能隐式地产生一个距离 q 投影的升序。这就造成了下面的，顺序查找 m 条链的方法: 这个 routine 是拿到一个查询 q，初始化 2m 个游标，然后通过游标来进行产生元素。

当有更多的存储空间和预处理可以花费的时候我们实现了对排序好的链表进行随机访问，然后给一个 D 中的 x，这个 routine 会返回在第 i 条链中 x 的排名。我们的 MEDRANK 不需要这样的随机访问。

### 2.1.3 Instance optimal aggregation

我们现在把计算 ε 近似最相邻问题，简化为如下的我们马上要描绘的场景。这里有 m 条排好序的链表，每条长度为 n，第 i 条链的元素的形式是(x,$v_i$ ) $v_i$ 是第 i 个排名高的值，第 i 条链式根据 $v_i$ 以降序排列的(我们的例子里面是升序)，在我们的例子中 $v_i$ 就是在链中第 i 的排名。更深入地来讲聚集函数就是拿 m 个分数链表，然后聚集成一个综合的分数链表。目标就是能够确定聚集后得分最高的 k 个元素。

这里有两个访问数据的方式，顺序访问或随机访问。在顺序访问中聚集算法从链表的头部开始扫描获取一个元素的值，因此如果元素 x 在链 i 中排名第 l，那么就需要 l 次顺序访问，而第二种模式是随机访问，在这种情况下就只需要一次随机访问。

在这个场景下，我们的算法 MEDRANK 可以这样来描述，x 的 $v_i$ 值代表的是 x 在链 i 上的排名，MEDRANK 是对 m 条链进行并行的顺序访。第一个出现在多于一半的链上的元素就是最相邻元素，而下一个就是第 2 个最相邻元素，一直下去就能找到 k 个最相邻元素。注意我们没有随经访问（除了我们的对游标的初始化过程用到了随机访问）事实上当聚集函数是 median 的时候，这很容易看出这是一个 NRA(无随机访问)算法。

我们将展示 MEDRANK 是一个单例最优(instace optimal)的算法。直观地来讲，NEDRANK 对每个数据库来说算法是最优解的常数倍。更严谨地说，instace optimality 是如下定义的。令$\mathcal{A}$代表算法的集合，$\mathcal{D}$代表数据库的集合，令 cost(A，D)代表算法 A 在数据库 D 中(顺序和随机)访问的总次数。一个算法 B 说是在$\mathcal{A}$和$\mathcal{D}$上 instance optimal 的话意味着对于每个$\mathcal{A}$中的 A 和$\mathcal{D}$中的 D 有 $cost(B,D) = O(cost(A,D))$，这个方程表示了存在这样的常数 c 和 c′ > 0 使得 cost(B,D) <= c*cost(A,D)+ c′ 常数 c 代表了最优比率(optimality ratio)

在我们的情形中，$\mathcal{D}$就是一系列的包含 m 条排名链的数据库，$\mathcal{A}$是正确的可以找出前 k 个 median rank 的(只允许顺序或者随机访问)算法

定理 5 让$\mathcal{A}$，$\mathcal{D}$代表上面所描述的一样，算法 MEDRANK 是在$\mathcal{A}$，$\mathcal{D}$上单例最优的。

证明，假设算法 MEDRANK 在数据库 D 上跑的时候在对每条链做完了 l 次访问后得出了，得到了 top-k 的结果，那么第 k 个最小的 median rank 是 l。

让 A 代表$\mathcal{A}$中的任意的算法，让我们定义一个空位(vacancy)在链 i 上的有值 j，表示在 level j 在链 i 是没被上是没被 A 所(随机或者顺序)访问到的。让 U 代表这样的集合：该集合的链在

小于 l 层的地方有空位，我们下面证明 U 的大小是最多$\lfloor m/2 \rfloor$。首先我们假设 U 的大小是最多$\lfloor m/2 \rfloor$这个结论不成立，那么我们定义这样一个新的数据库 D' 它是通过如下修改 D 中 U 的每条链得来的: 令 x 代表一个新的不在 D 中的元素，对于 U 中的每条链，x 的排名放到每条链中的第一个空位，而对应第一个空位 level 值得原先的元素挪到链的底部。对于不再 U 中的每条链，我们把 x 放到链中底部。直观上来讲 x 填充了每条在 U 中的链的第一个空位。根据我们的假设 x 在 D 中所有链的排名小于 l 的数目超过了半数，那么它的 median rank 将会肯定小于 l。现在算法 A 在 D 和 D' 上跑，这样话 A 算法应该是给出了两个一样的结果，但是 A 在 D' 上作出了错误的判断，因为 x 这个新加的元素有比 l 要小的 median rank 却不再 k-top 元素结果里面。所以我们的假设不成立。证得 U 的大小是最多$\lfloor m/2 \rfloor$。

让 Q 代表算法 A 的访问次数，那么从我们刚才的证明中我们可以发现有至少$\lceil m/2 \rceil$条链式没有在小于 l 层的 vacancy 的，这隐含着$Q \geq \lceil m/2 \rceil (\ell - 1) \geq (m/2)(\ell - 1)$，因此 ml <=2Q+m ，而 ml 就是我们 REDRANK 的访问次数，因此我们的 MEDRANK 是 instance optimal 的，期中 optimality ratio 最多是 2

在定理 5 的证明当中，我们看到了算法 MEDRANK 有 optimality ratio 最多是 2，还附带了个加性的常数 m，如果我们希望摆脱这个加性的常数，那么我们可以用 Q > m/2 这个事实，来获取 optimality ratios 等于 4，从而摆脱加性常数 m，非常有趣的是 optimality raios 在文献[11]中给出的都是 m 的线性或者平方，而我们的算法是第一个有意义的可以令 optimality ratio 独立于 m 的算法。

会有 MEDRANK 算法访问链表的长度超过一半的情况出现，但是文献[9]指出了当链表是随机独立的时候，MEDRANK 算法访问链表的长度是$O(n^{1-2/(m+2)})$。当排名链表间是正相关的时候，我们希望终止会在更早地结束。事实上，当排名链表是通过数据库的点再 r 上的投影和 q 在 r 上的投影的距离来排名的时候，会发现链表是非常相关的，这里我们不去更深入地探讨了。

我们比较下 MEDRANK 算法和文献[9]Fagin's 的 FA 算法。FA 仅当在所有的链上看到 k 个元素才会停止，而 MEDRANK 在一半的链上看到 k 个元素就会停止了。还有 FA 中 k 的个元素在 m 条链中不一定是 top-k 元素，所以随机访问还是要的，然而我们的 MEDRANK 中 k 个元素是在 m 条(隐式)链的头部的所以不需要随机访问。

## 2.2 Summary of algorithm

我们现在给出 MEDRANK 和两个相关的算法 OMEDRANK 和 L2TA 的正式描述，OMEDRANK 是一个启发式的提升，致力于更好地减少运行时间，而 L2TA 算法则是一个“threshold algorithm”的实现，它是一个计算欧氏最相邻问题的单例最优算法(模型限制在数据的坐标是通过顺序或随机访问到的)，我们把直接通过线性扫描数据库元素来找到最相邻元素的朴素算法记为 L2NN。

算法的描述是标准的伪代码风格，我们倾向于清晰明了，而不是详细的细节和和边界条件。我们会描述寻找优胜者的过程，还有找出 top-k 元素的扩展方法很直接。

算法 MEDRANK 是聚集算法家族中的一员，我们可以通过修改 median 这个中位数的形成变种算法，我们给出了 MINFREQ 这个参数来控制中位数这个值得替代，即使这个值表面上不会和最相邻问题有直接的联系，但是 MINFREQ 这个参数是直接关系到每个选取出来的元素要出现在各个链表中的至少次数，而 median rank 算法中的 MINFREQ 的值就等于 0.5. 这里改变这个值得思想在于，我们期望通过提升这个值来让算法要探寻 m 条链的更深的深度从而提升结果的质量，因此可以有时间和检索质量的权衡。

---

**Algorithm** MEDRANK

*Pre-processing*

Create $m$ lists $L_1, \ldots, L_m$, where $L_i$ consists of the pairs $(c, c_i)$ for all $c \in D$.

For $1 \leq i \leq m$, sort $L_i$ in ascending order of the second component. Now each $L_i$ has the form $(c_{i,1}, v_{i,1}), \cdots, (c_{i,n}, v_{i,n})$, where the $c_{i,t}$'s are the $n$ distinct objects in the database, and $v_{i,1} \leq v_{i,2} \leq \ldots \leq v_{i,n}$.

*Query-processing*

Given $q \in \mathbf{R}^m$, for each $i$, initialize two pointers $h_i$ and $l_i$ into $L_i$ so that $v_{i,h_i} \leq q_i \leq v_{i,l_i}$.

$S$ will be a set of "seen elements" $c \in D$ and their frequencies $f_c$; initialize $S$ to $\emptyset$.

**while** $S$ has no element $c$ s.t. $f_c >$ MINFREQ $* m$ **do:**

for $1 \leq i \leq m$ do:

if $|v_{i,h_i} - q_i| < |v_{i,l_i} - q_i|$ then

set $c = c_{i,h_i}$ and decrement $h_i$

else

set $c = c_{i,l_i}$ and increment $l_i$

if $c \notin S$, then

add $c$ to $S$ and set $f_c = 1$

else

increment $f_c$

end-for

**end-while**

Output the element $c \in S$ with the largest $f_c$.

第二个我们要描述的算法似乎 OMEDRANK，这是一个基于下面对 MEDRANK 算法的观察得到的算法。我们会直接考虑 $c_{i,h_i}$ 和 $c_{i,l_i}$ 这两个元素，因为我们不会使用随机访问(例如：查找 $c_{i,h_i}$ 在其他链表 $L_j$ 的排名)，这会增加我们在 S 中要考虑的元素，但是我们可以节约很多的比较次数。

---

**Algorithm** OMEDRANK

*Pre-processing*

Identical to MEDRANK.

*Query-processing*

Given $q \in \mathbf{R}^m$, for each $i$, initialize two pointers $h_i$ and $l_i$ into $L_i$ so that $v_{i,h_i} \leq q_i \leq v_{i,l_i}$.

$S$ will be a set of "seen elements" $c \in D$ and their frequencies $f_c$; initialize $S$ to $\emptyset$.

**while** $S$ has no element $c$ s.t. $f_c >$ MINFREQ $* m$ **do:**

for $1 \leq i \leq m$ do:

for $c \in \{c_{i,h_i}, c_{i,l_i}\}$ do:

if $c \notin S$, then

add $c$ to $S$ and set $f_c = 1$

else

increment $f_c$

end-for

decrement $h_i$ and increment $l_i$

end-for

**end-while**

Output the element $c \in S$ with the largest $f_c$.

最后我们会描述一下一个用来计算欧氏最相邻元素的单例最优算法，这个算法是一个“阈值函数(threshold algorithm)”的作用或者 TA，来自于文献[11]，它是为了计算欧氏距离(L2 的来源)的最相邻问题而存在的。这个算法我们叫作 L2TA 算法，可以用来代替原生朴素的最相邻问题的解法，而这个算法会更快些。

**Algorithm** L2TA

*Pre-processing*

Create $m$ lists $L_1, \ldots, L_m$, where $L_i$ consists of the pairs $(c, c_i)$ for all $c \in D$.

For $1 \leq i \leq m$, sort $L_i$ in ascending order of the second component. Now each $L_i$ has the form $(c_{i,1}, v_{i,1}), \ldots, (c_{i,n}, v_{i,n})$, where the $c_{i,t}$'s are the $n$ distinct objects in the database, and $v_{i,1} \leq v_{i,2} \leq \cdots \leq v_{i,n}$.

Create the index $P$ such that $P(i, c)$ equals that value of $j$ where $c_{i,j} = c$, for each $c \in D$ and $1 \leq i \leq m$. That is, $P(i, c)$ is the position (or rank) of $c$ in the sorted list $L_i$.

*Query-processing*

Given $q \in \mathbf{R}^m$, for each $i$, initialize two pointers $h_i$ and $l_i$ into $L_i$ so that $v_{i,h_i} \leq q_i \leq v_{i,l_i}$.

$S$ will be a set of "seen elements" $c \in D$ and their distances $d_c$ to $q$; initialize $S$ to $\emptyset$.

$T$ will be a "threshold value" that tracks the minimum distance that any unseen element $\bar{c} \notin S$ can achieve to $q$; initialize $T$ to $0$.

**while** $S$ has no element $c$ s.t. $d_c \leq T$ **do:**

for $1 \leq i \leq m$ do:

let $a_i = |v_{i,h_i} - q_i|$ and $b_i = |v_{i,l_i} - q_i|$

if $a_i < b_i$ then

set $c = c_{i,h_i}$ and decrement $h_i$

else

set $c = c_{i,l_i}$ and increment $l_i$

if $c \notin S$, then

$s = 0$

for $1 \leq j \leq m$ do:

$p = P(j, c)$; $s = s + v_{j,p}^2$

end-for

add $c$ to $S$ and set $d_c = \sqrt{s}$

end-if

end-for

$T = (\sum_{i=1}^{m} \min(a_i, b_i)^2)^{1/2}$

**end-while**

Output the element $c \in S$ with the smallest $d_c$.

# 3. EXPERIMENTS

## 3.1 Data collection

我们的实验是建立在两个数据集上的，其中我们分别称之为 STOCK 和 HW。在下面，我们将会详细地描述这两个数据库的细节。注意到通过一个 feature 向量在一个合适的高维空间中，相似性检索和分类可以转化成最相邻问题来看待。

第一个数据集是 STOCK 这是从历史的一些美国公司的股票价格得来的。这些数据第一次是在雅虎的商业网站上获取的，数据包含了 7999 家公司，每家公司的数据都是切分成以 100 连续天为周期的数据(如果有与数的话就去掉)，我们假设 1 美元投资在这个股票的周期开始处，然后跟随这一美元在 100 天里的进展，这样就创造了一个 100 维的向量，这个过程为每个公司都做，之后我们得到了 145，619 个 100 维的向量。这样做的原因有两个，第一增加数据集合的大小，第二能够在不同的时间窗口对比股票。

第二个数据集合从公开的数据库 MNIST 得来的(http://yann.lecun.com/exdb/mnist).

原先的数据包含了一个有 60,000 个已经标签好的手写数字图片样本的训练集，还有 10,000 个测试样本，每张图片是 28*28 的大小，标签就是 0 到 9.所以 feature 向量就是 784 个像素值，因此每个向量是 784 维的。因为我们只对基于最相邻问题的分类感兴趣，而不是训练模型，所以我们就不这训练和测试的数据集合并在了一起，形成了 70,000 个向量。

这两个数据集是有意选择来自不同的领域的，目的是为了增加我们实验的多样性，STOCK 是一个大量的维度较少的数据集，而 HW 是少量的维度较高的数据集。另外一个重要的不同点在于 HW 是隐式聚簇的应为他们是 10 个类的(数字 0 到 9)，另一方面，没有自然的分类可以用在 STOCK 的数据集上。最后，我们在 STOCK 上考虑的是相似性检索问题，而在 HW 上考虑的是分类问题。注意在 HW 的数据集中的特征检查问题是个有争论的重要的近似最相邻问题的应用。在这里分类好各个特征比找到精确的最相邻元素更重要。

我们决定把所有者实验数据一次性全部装载进内存，所以硬件限制了我们选择更大的数据集。之所以要将数据全部导入到内存是因为 L2NN 和 L2TA 算法的需要，我们将会拿 MEDRANK 和 OMEDRANK 算法和它们比较。如果大量的数据存在二级存储设备中，这些算法计算代价会更加昂贵，因为它们倾向于访问数据库的大部分数据，这会造成很高的硬盘访问。

## 3.3 Parameters studied

(1) 时间。我们研究计算出 top-10 元素的基本运行时间，运行时间基本包括了查询的预处理，因为 L2NN 是在全维上检索的所以有道理认为是”决定正确”的我们比较其他的算法和全维额 L2NN 的算法的运行时间(即使 L2TA 在全维上市一个”精确”的解，但它比 L2NN 慢，因此我们对比所有算法是相对于 L2NN 来说的)

(2)质量。我们使用两个不同的质量记号来标记 STOCK 和 HW。对于 STOCK，这是如下的，让 q 成为一个查询，让 p 成为算法对于 q 返回的 top 点(可能用到了投影数据)，令 p*成为 L2NN 算法返回的对于 q 的最相邻的值。直观上，p*是一个“正确”的答案(最相邻元素)，质量就记为 d(p,q)/d(p*,q)。

在 HW 的 case 当中，质量是如下定义的：回忆起我们对每个 HW 的数据都有标签，令 ε 记为我们的算法对于查询 q 的分类的错误（可能用到了投影数据），令 ε*记为在全维上 L2NN 算法对于查询 q 的分类的错误。错误率是查询结果和真正的分类结果不同的比例。而质量定义为 ε/ε*。这样做的，而不是用绝对错误数来衡量的主要原因是，分类的错误不仅仅是一个最相邻或聚集函数的算法的好坏，还关系到 feature 集合的选取。（我们没有对 feature 集合进行优化，因为这不是我们工作范围之类的事，因此我们应该仅仅和硬算的最相邻问题的结果进行对比）

(3)探测的深度和部分访问。回想我们的算法 L2TA,MEDRANK 和 OMEDRANK 不会访问我们整个的数据库。对于 MEDRANK 和 OMEDRANK 算法我们可以通过对数据库友好的方式进行顺序访问，我们记录这个顺序访问的次数，实际上我们记录的是返回 top-10 的结果所需要的访问次数。

我们预计探寻的深度和我们期望的的最相邻元素的排名相关。(我们谈到期望是因为，m 条链是随机生成的)我们计算 rank(w)的分布，w 是查询相对应 q 的获胜者(q 是数据库 D/{q}的一个查询).rank(w)的分布是通过超过 1000 次的随机查询所平均出来的。图 1 展示了这个分布；rank(w)量化后的期望值（相对于 STOCK 数据）来说大概是 0.13 那么也就是说我们期望 MEDRANK 和 OMEDRANK 算法的探测深度平均来说只是 13%

算法 L2TA，除了顺序访问外，还会有随机访问，我们也记录下了这些信息。

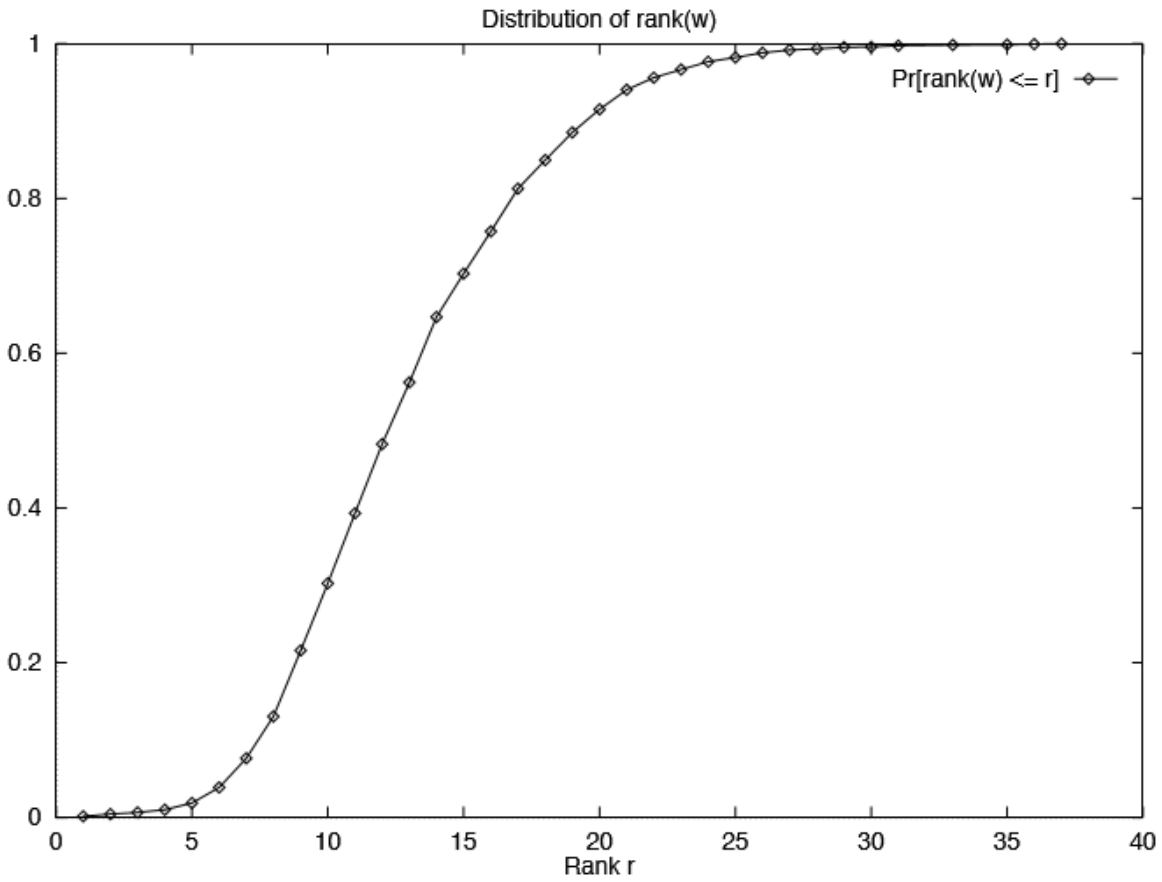



**Figure 1: Distribution of rank$(w)$.**

## 3.4 Result

为了避免读者淹没在大量的数据里，我们只是呈现了一先基本的分别对于 STOCK 和 HW 的数计在表 1 和表 2 中。

| | L2NN | L2TA | | MEDRANK | | OMEDRANK | |
|---|---|---|---|---|---|---|---|
| DIM | Time | Time | Qual. | Time | Qual. | Time | Qual. |
| | | | | MINFREQ = 0.5 | | | |
| 10 | 0.195 | 0.065 | 1.399 | 0.002 | 1.794 | 0.004 | 1.790 |
| 20 | 0.289 | 0.139 | 1.270 | 0.005 | 1.518 | 0.006 | 1.514 |
| 30 | 0.376 | 0.232 | 1.231 | 0.008 | 1.430 | 0.009 | 1.426 |
| 40 | 0.466 | 0.344 | 1.201 | 0.013 | 1.338 | 0.013 | 1.332 |
| 50 | 0.555 | 0.440 | 1.186 | 0.017 | 1.333 | 0.015 | 1.330 |
| 100 | 1.000 | 11.00 | 1.000 | 0.459 | 1.360 | 0.352 | 1.434 |
| | | | | MINFREQ = 0.7 | | | |
| 10 | 0.195 | 0.065 | 1.399 | 0.003 | 1.654 | 0.004 | 1.663 |
| 20 | 0.289 | 0.139 | 1.270 | 0.007 | 1.414 | 0.009 | 1.412 |
| 30 | 0.376 | 0.232 | 1.231 | 0.012 | 1.344 | 0.013 | 1.345 |
| 40 | 0.466 | 0.344 | 1.201 | 0.020 | 1.273 | 0.018 | 1.274 |
| 50 | 0.555 | 0.440 | 1.186 | 0.026 | 1.264 | 0.023 | 1.259 |
| 100 | 1.000 | 10.99 | 1.000 | 0.817 | 1.253 | 0.645 | 1.286 |

**Table 1: Basic performance measures for the algorithms on** STOCK **data at** MINFREQ **= 0.5, 0.7. Time denotes the time relative to** L2NN **in full dimensions and qual. denotes the distance ratio relative to the one obtained by** L2NN **in full dimensions.**

| | L2NN | L2TA | | MEDRANK | | OMEDRANK | |
|---|---|---|---|---|---|---|---|
| dim. | Time | Time | Qual. | Time | Qual. | Time | Qual. |
| | | | | MINFREQ = 0.5 | | | |
| 20 | 0.042 | 0.236 | 11.38 | 0.004 | 23.75 | 0.004 | 23.25 |
| 40 | 0.063 | 0.616 | 6.042 | 0.010 | 12.50 | 0.011 | 14.17 |
| 60 | 0.087 | 1.030 | 3.875 | 0.019 | 10.47 | 0.018 | 10.00 |
| 80 | 0.110 | 1.458 | 3.625 | 0.029 | 7.917 | 0.026 | 7.167 |
| 100 | 0.134 | 1.876 | 3.542 | 0.040 | 7.083 | 0.033 | 6.625 |
| 120 | 0.156 | 2.319 | 3.333 | 0.052 | 6.667 | 0.042 | 5.208 |
| 160 | 0.203 | 2.400 | 2.830 | 0.078 | 4.583 | 0.063 | 4.583 |
| 200 | 0.250 | - | - | 0.098 | 4.583 | 0.083 | 4.167 |
| | | | | MINFREQ = 0.9 | | | |
| 20 | 0.042 | 0.236 | 11.38 | 0.011 | 14.58 | 0.012 | 13.25 |
| 40 | 0.063 | 0.616 | 6.042 | 0.029 | 7.500 | 0.029 | 7.708 |
| 60 | 0.087 | 1.030 | 3.875 | 0.051 | 5.833 | 0.047 | 5.125 |
| 80 | 0.110 | 1.458 | 3.625 | 0.078 | 5.000 | 0.067 | 5.000 |
| 100 | 0.134 | 1.886 | 3.542 | 0.106 | 7.083 | 0.086 | 4.250 |
| 120 | 0.156 | 2.319 | 3.333 | 0.137 | 5.833 | 0.108 | 3.583 |
| 160 | 0.203 | 2.400 | 2.830 | 0.197 | 3.750 | 0.160 | 3.750 |
| 200 | 0.203 | - | - | 0.253 | 3.750 | 0.208 | 3.750 |

**Table 2: Basic performance of various algorithms on HW data at MINFREQ = 0.5, 0.9. Time denotes the time relative to L2NN in full dimensions and Qual. denotes the classification error relative to the one incurred by L2NN on full dimensions.**

(1)时间.我们可以从表中看见，即使是全维的数据，MEDRANK 和 OMEDRANK 的运行时间都是持续小于 L2NN 的(大概 35-45%于 L2NN 的时间)，在投影过了的数据上 MEDRANK 和 OMEDRANK 是要快两个数量级的，这些算法在非常高的 MINFREQ 下也是比 L2NN 高的。

我们提到的这些不同点要是在有硬盘访问的情况下会更加地有说服力。而且如果我们要是仅仅计算 top-1 的最相邻值而不是现在的 top-10 的相邻值，MEDRANK 和 OMEDRANK 会有更加出色的表现。

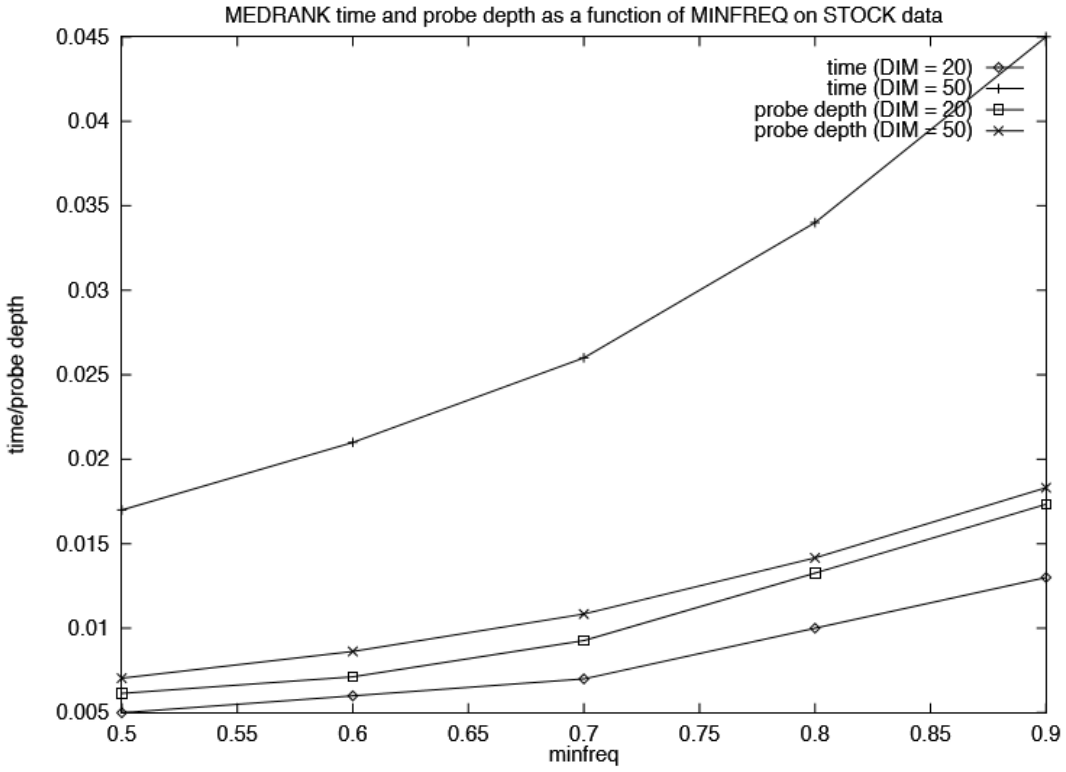
算法 L2TA 提供了一个在低维的 STOCK 数据下有效的加速，但是在高维的情况下就很糟糕了，这可以归结于该算法的 bookkeeping 的结果。

(2)质量.再一次这个表的数据表明了 MEDRANK 和 OMEDRANK 的结果的质量是相当高的。对于 STOCK 来说 近似的因子大概为 2，意味着这些算法找到的最近的点再真正的最相邻点的 2 倍范围之内，注意 L2TA 算法会找到最终的最相邻点，因此会符合 L2NN 的 quality 在那一个维度上。

一个更重要的点要注意的是因子为 2 的相似最相邻元素可以在 1%的运行时间内找到，而在 HW 中就没有那么厉害，要用 6%的时间。最终的错误的比率会比 5 更多。

# 3.4.1 Other statistics

我们接下来给出更多这些算法的统计数据(图 2-5)它们包括:参数 MINFREQ 对时间的影响，运行时间和探索深度的关系，参数 MINFREQ, top-k，对 MEDRANK 算法的检索质量的影响。还有 L2TA 算法的探索深度和访问的统计。图片下面的字描述了主要的观察。



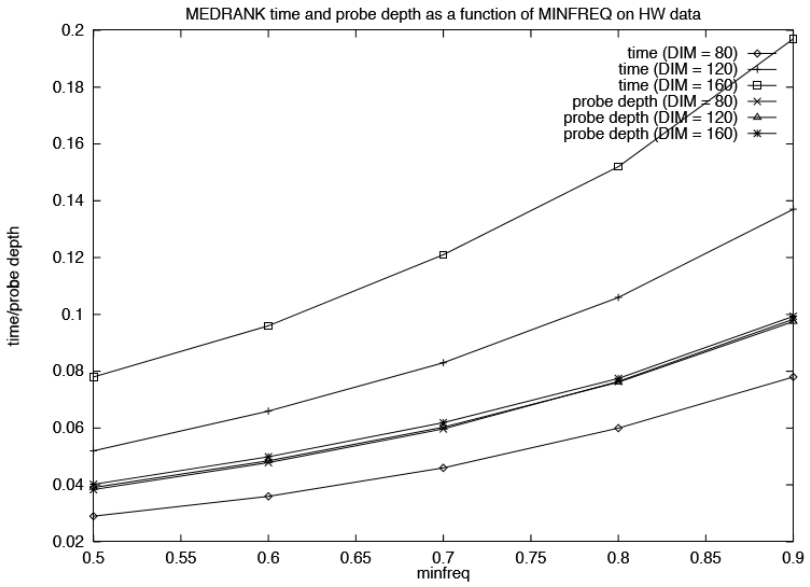



**Figure 2:** MEDRANK **time and probe depth as a function of** MINFREQ **on** STOCK **and** HW **data. Notes: (1) In both cases, dimension has almost no effect on the probe depth. (2) Even at** MINFREQ $= 0.9$**, time taken is very small.**

### 3.4.2 Inferences

(1) MEDRANK 和 OMEDRANK 算法都是极其地快和仅仅扫描数据库的一小部分，即使 MINFREQ 达到了 0.9(看图 2)，因此，这些算法相对于 L2TA 而言非常地高效和有效。

(2) 如果只是考虑大概的结果的话，把数据投影到低维通常是一个有优势的步骤。在保持相关性的同时，随机的投影降低了噪音。在投影的数据中(我们最感兴趣的部分)，在同样的数据集上，这些算法的检索质量是相比于 L2NN 查不到哪里去，同时运行时间大大减少。还有投影可以有效地减少探寻的深度(至少一个数量级)(图 5)。我们由此总结投影是一个好的 idea，如果只是考虑近似的结果的话。在投影数据上，MEDRANK 和 OMEDRANK 比 L2NN(或者甚至是 L2TA)要好得多。

(3) MEDRANK 和 OMEDRANK 的比较，我们发现在几个 case 中，OMEDRANK 比 MEDRANK 要快 20%，同时还保持了结果的质量(看表 1 和 2)

(4) 参数 MINFREQ 在算法 MEDRANK 和 OMEDRANK 中扮演了不同的角色，在 STOCK 中我们注意到这个参数没有什么有效的作用，因此让 MINFREQ 保持在 0.5 就差不多了，而对于 HW 来说，它能有效地降低错误率(看图 3)，然而这回影响探测的深度(从而就是运行时间，看图 2)。就算是这样探测深度至少还是整个数据集的 1%-10%，这保证了鲁棒性。

(5) 我们检验了检索出 MEDRANK 在 top-10 或仅仅是 top-1 间的时间有什么不同，发现(图 4)并没有什么重大的不同。

(6) 我们总结了下 L2TA 对最相邻问题提供了有意义的贡献，但是在低维下提高不是非常高的，而在高维下则很差(看图 1，2)，深入的确信是在图 5，那里展示了探测的深度和数据库的访问对比图。很容易看到 L2TA 是要访问数据库的很大部分的，然而 MEDRANK 则是少量的。

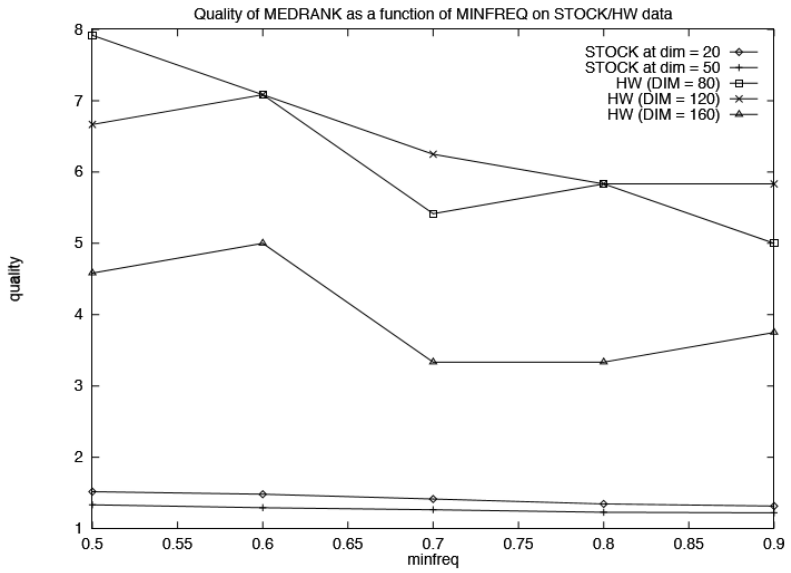



**Figure 3: Quality of** MEDRANK **as a function of** MINFREQ **on** STOCK/HW **data. Notes: (1)** HW **data shows much more improvements as a function of dimensionality than** STOCK **data. (2)** MINFREQ **seems not to affect results on** STOCK **data much, but on the** HW **data, a value of roughly 0.7 seems to be the best.**

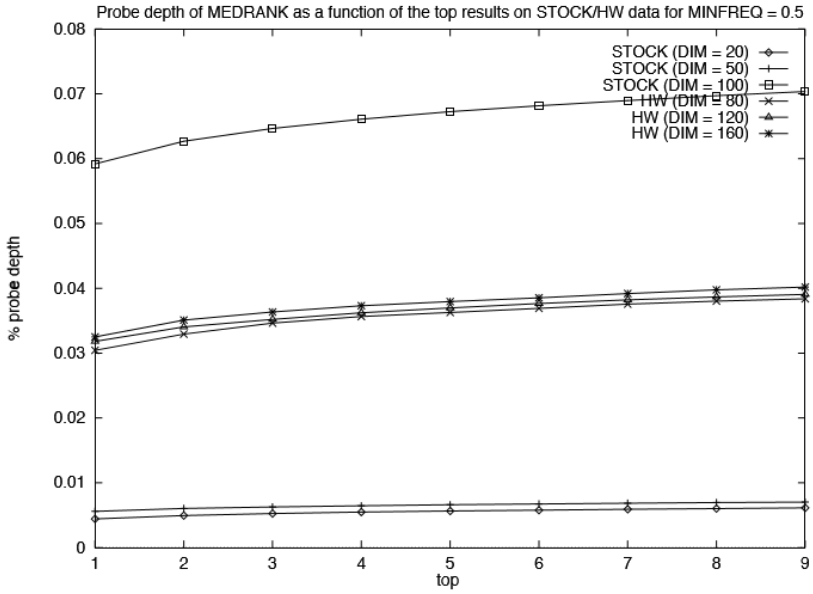



**Figure 4: Probe depth of** MEDRANK **as a function of the top results on** STOCK/HW **data for** MINFREQ**=0.5. Notes: (1) Dimensionality reduction causes significant improvement in the probe depth for** STOCK **data (compare 100 dimensions vs. lower dimensions). Note that we did not conduct the** HW **data experiments on the full 784 dimensions. (2) Whether we are computing top 1 or top 10 seems not to affect probe depth by much in both cases.**

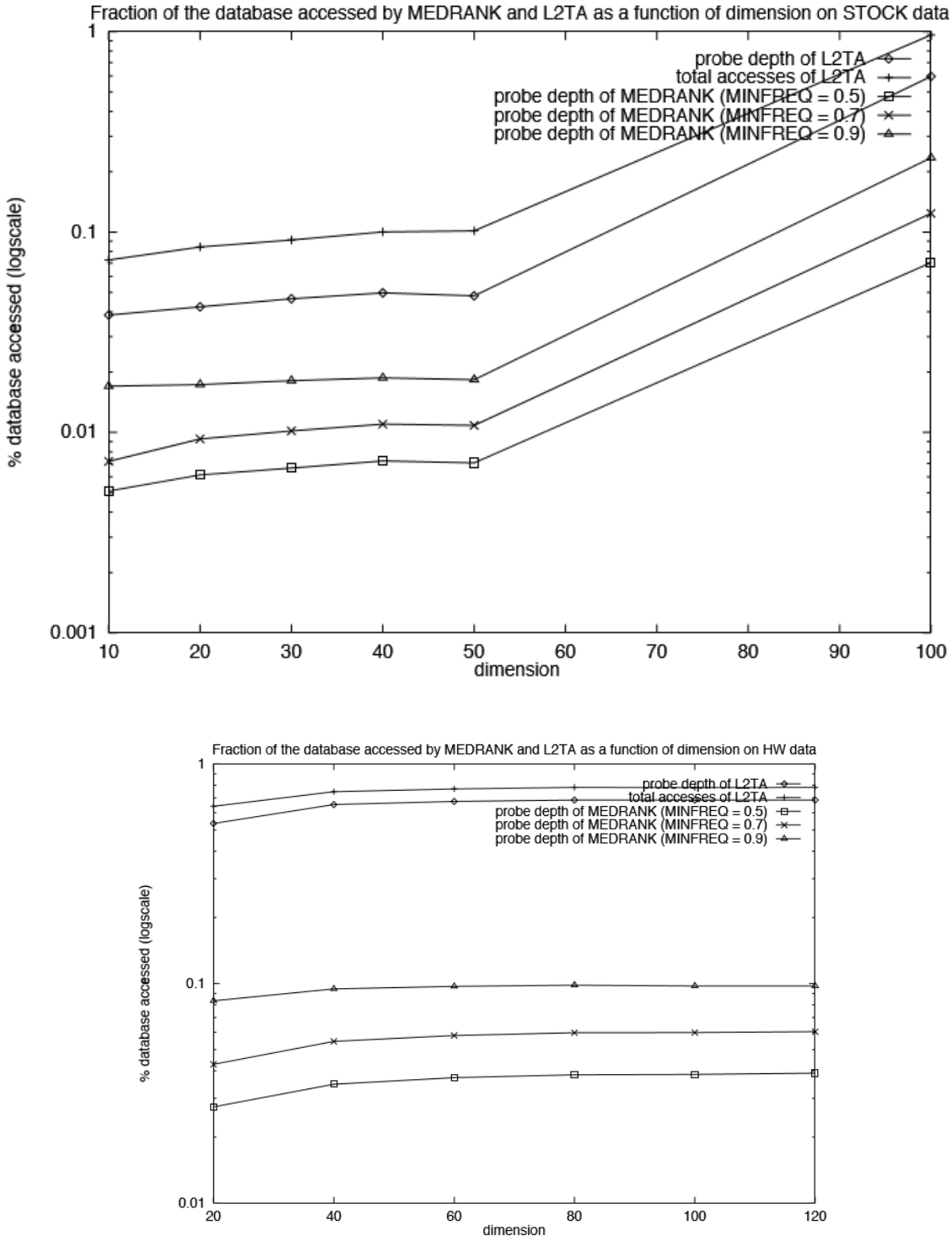



**Figure 5: Fraction of the database accessed by** L2TA **and** MEDRANK **as a function of dimension on** STOCK **and** HW **data. Note: (1)** MEDRANK **and** OMEDRANK **access an order of magnitude fewer elements of the database than** L2TA**.**

# 4.CONSLUSIONS

我们已经介绍过排名聚集的用来解决相似性搜索和分类的新方法了，我们将查询和候选人表达为多维坐标中的点。每个坐标轴充当一个 voter，voter 则根据候选人和查询在相应坐标中的距离来给出排名。而获胜者就是那些将所有 voter 的排名聚集起来的最终排名最高的那个候选人。通过投影降维的处理，这将会产生出一个简单，数据库友好的算法，该算法会给出一个很大的最相邻为题的近似解。这个算法是极其高效的，它经常不会访问超过 5%的数据库的数据却能给出质量非常高的解。我们感觉这个方法是概念上非常有意思的，而不仅仅是它能给出一个很好的近似解。我们的结果也是很好地表明了 median rank aggregation 是一个高效的有用的排名聚集方法。

一个有趣的研究方向是考虑这样的 case：在小的定义域中一些点被强制拥有相同的排名，然后修改我们的方法去探讨它的特征。