通过排列聚合的有效相似性查找与分类

摘要：

我们计划使用一种新奇的方法在海量数据中演示高效相似性查找与分类。在这个框架中，数据库中的元素是欧式空间的向量。给定一个相同空间中要查询的向量，要找到数据库中与之相似的元素。为了实现这个目标，我们会使用小数量独立的“voters”基于与查询向量的相似程度对数据库中的元素进行排序。接下来这些排序结果将会通过一个高效的聚合算法结合起来。我们的方法使得计算近似最近邻居和对最近邻居的最优替代更接近技术化。

我们方法论的一个例子如下所示：每一个 voter 将所有的向量（数据库元素和查询元素）投射到任意一条直线上（每一个 voter 投射的直线不同），并且根据与查询元素的预测投射距离对数据库中元素进行排序。聚合原则将排序后数据库中最中间的元素摘取出来。这个聚合原则有几个吸引的特性。在理论上，我们很大程度上证明了得出的结果与欧式空间最近邻居的误差仅有 1+ε。在实际应用上，它体现了极大的效率性，经常在查询不超过 5%的数据便可以得出高精确性的结果。这种方法更是对数据库友好，因为它主要以预定义的顺序而不是随意地对数据进行访问，并且不同于其他计算相似最近邻居的方法，它无需额外的内存开销。并且，我们进一步扩展了我们的方法使其可以处理 k 个最近邻居的问题。

我们进行了两个系列的实验估计我们这个方法的效率，包括相似性查找和分类问题，在这两种情况中最近邻居都典型地被应用。在两个实验中，我们根据几种评价的标准来分析我们的方法的效果，并且总结出，他们在结果的精确性和效率上一致显示出优越性。

介绍

最近邻居问题在计算机科学的许多应用领域都是普遍存在的。非正式上来说，这个问题可以解释为 给定一个包含某个空间中 n 个点的数据库 D 和在同一空间的一个查询要素 q，要找到在 D 中最接近 q 的一个或者 k 个点。一些查找最近邻居的优秀应用包括信息检索中的相似性查找，模式分类，数据分析，等等。最近邻居问题的流行性是因为它相当简单和自然将显示生活中的物件定位到欧式空间的一个向量，一些关于相似性和分类的问题也转变成了最近邻居问题。因为将物体投射成一个向量经常是一个探索性的步骤，在许多应用中它足够在数据库中找到一个近似最近邻居的点。这些问题引出了一个非常有趣的计算问题，有大量关于计算最近与近似最近邻居的论文。对于现在一些理论性工作，可以参考[18, 16, 19]，对于现在的理论应用工作，可以参考[13, 1, 12, 5, 4, 20]。

在这篇论文中，我们对相似性查找，分类问题和其他基于最近邻居搜索应用使用了一个新奇的方法。我们的方法基于两个基础范例：排序聚合和实例优化算法。我们的方法满足一下两种要求甚至更具矛盾性的标准：它是一种对最近邻居的暴力一般化，允许高效和数据库友好的算法。

我们工作开始的点是以下简单的一个构思。假设给定一个查询 q 属于 X^d，我们要在含有 d 维空间 X^d 中 n 个点的数据库 D 中构建最近邻搜索。我们可能会考虑把 d 维空间中的每一个坐标看做一个“voter”，n 个数据库点看做在选举过程中的“candidates”。Voter j（1<=j<=d），基于与 voter j 有多接近的原则对 n 个 candidates 进行排序。这个过程中我们可以得到 d 个关于 candidate 的有序表，而我们的目标是合成这些简单排序的 candidates

表，我们对在这个聚合排序中最靠前的几个 candidates 非常感兴趣。

排序聚合问题准确来说就是如何聚合由 d 个坐标提供的 d 个有序表的问题。这个问题的历史可以追溯到至少 2 个世纪以前，但对它的数学理解却只是发生在 60 年以前，并且最根本的计算问题仍然潜在于当今活跃的调查中[3, 14, 8]。聚合算法中最重要的一个数学问题是识别对聚合的暴力性机制，尤其值得注意的是 Young 和 Young and Levenglick 在这个领域做出的贡献。Young 和 Young and Levenglick 提出了一个关于 Kemeny 的计划，并导致了一种可以占有很多可观性质的暴力性机制。例如，它满足孔多塞标准，如果存在一个 candidate c，那么对每一个其他的 candidate c1，大部分的 voter 都会偏向于 c 而不是 c1，则 c 是这场选举中的胜者。满足孔多塞标准和它的自然扩展性质的聚合机制被认为可以从不能被一些坏 voters 抛弃的大量结果中获益。

Kerneny’ s 方法表达如下：给定 n 个 candidates 和这些 candidates 中 d 个不同向量的排列，我们可以得到一个排列σ使得 $\sum_{i=1}^{d} K(\tau_i, \sigma)$，最小，其中 $K(\tau, \sigma)$ 表示 Kendall tau 距离，那就是说，candidates 中(c, c1)的数目在排列τ和σ不一致（有些吧 c1 排在 c 前面，而有些则相反）。我们把这叫做 Kendall optimal aggregation。然而不幸运的是，计算超过四个表的 Kendall optimal aggregation 是不完全多项式，所以其中一个表必须采取估算算法。

让我们现在来解释一下最近邻居和排序聚合之间的联系吧。作为一个非常简单却十分有力的例子来说，如果基础空间是以汉明度量的{0，1}^{d}，那么每一个 voter 都提供一个局部顺序，给定一个查询 q，那么第 i 个 voter 将数据库 D 划分为两个区域，$D_i^+ = \{x \in D \mid x_i = q_i\}$ and $D_i^- = \{x \in D \mid x_i \neq q_i\}$，将所有的 D+排在 D-之前。（Kendall tau 距离和 Kendall optimal aggregation 的概念仍然具有意义，因为他们是基于在一次比较两个 candidates。）在这种情况下，我们不难看到，部分排序下的 Kendall optimal aggregation 是由 voters 精确地根据数据库中每一个点与查询 q 的汉明距离进行有序排序的。同时考虑到在几种有趣度量下的最近邻居问题可以简化为汉明度量[19,16,6]下的问题，我们认为排序聚合，一般来说，至少与最近邻居同等强大。（我们稍后会提供更加复杂的证据。）

在另一方面，我们采取了一个可以被在 O(nd)时间复杂度的直截了当的算法解决并且能够重铸为不完全多项式问题的问题。而一些相当好的近似算法也需要至少 $\Omega(nd + n^{\widetilde{2}})$ 时间解决聚合问题（e.g. 见[8]）。但是，这两种重要因素的汇合为我们解决了这种两难的困境。首先，我们只对聚合排序中前几个元素而不是对数据库中每一个点的完全排序感兴趣。其次，在寻找聚合排序中最前 k 个赢家的过程中，一个基于中值排序的搜索显示出了极端有效的解决途径。我们接下来讲这个问题。

1.1 中位数排序聚合

在计算 Kendall optimal aggregation 不同于承认一种高效算法的时候，一种在 footrule sense 中体现合理的多项式时间可计算排序对 Kendall optimal ordering 接近 2 个因子的近似。更进一步来说，footrule-optimal 聚合有如下优秀的 heuristic，我们称之为中位数排序聚合：对基于从 d 个 voters 得到的排序的中位数的数据库中 n 个点进行排序。这是一个合理的 heuristic, 因为如果中位数排序都是明显不同的话, 那么这个过程就会提供一个 footrule optimal 聚合。因此，我们把问题简化为找到数据库中有最佳中位数排序的那些点（或前几

名中位数排序的点）。

不同于把中位数排序聚合仅仅看做 Kendall optimal aggregation 的一个 heuristic 近似，我们把它看成一个自然的排序聚合方法。正如我们在 2.1.1 展示的那样，中位数排序聚合为那些与 footrule 距离概念相似的距离提供了一个合适的解决方法。进一步来说，中位数排序聚合有两个令人满意的性质，这些我们接下来将会提及。

数据库的友善和实例优化算法 对于中位数排序聚合的一个很大的争议性是他的数据库友好。明确来说，我们想要提出一个针对在数据库系统中具有令人满意性质的最近邻居问题的解。理想上，每个人都想要避开涉及到复杂数据结构，大量内存消耗，或者需要大量随机存取的方法。例如，这些考虑立刻排除了那些理论上可能的好方法[19,16,18]；甚至现在数据库文献[13,1,5]中的一些方法也受到一个或多个问题的阻塞。对比来说，中位数排序聚合将排序作为唯一一个预处理步骤，几乎不需要额外多的内存，执行几乎不需要随机访问。为了避免随机访问，我们的方法不需要可以定位每一个元素坐标的值的索引。

我们现在讨论一下对中位数排序聚合的有效方法。我们先对 n 和数据库点进行预排序。给定一个查询 q = (q1, ⋯, qd)，我们可以很容易定位在第 i 个有序表中的 qi，其中 1<=i<=d，并且在这个位置上放置两个光标。一旦 2d 个光标已经放置，对于每个 i 位置有 2 个光标，通过移动一个光标向上，一个光标向下，我们现在提供一个可以提供第 i 个 voter 的有序表的流，一次一个元素。也就是说，我们认为 d 个 voters 在以下在线表现中操作：首先第 i 个 voter 被调用，它会在 i 的坐标上返回数据库中最靠近 q 的一个元素，第二次它会返回第二靠近 q 的一个元素，如此下去。因此，有效的是，我们有一个关于在线排序版本的问题需要解决。

我们能够很容易提供对第 d 个 voter 的一个在线访问和我们能够提供具有最佳中位数排序的 candidate，暗示着我们不需要读有序表便可以识别出赢家是谁。实在地说，计算分数表的聚合使用一种最佳数目的连续顺序和对这些表的随机访问——并且希望不用完全查询这些表——在数据库文献中具有很大的吸引力。我们会设计一个算法，算法基于 NRA 思想或者无 “随机访问” 。[11]的方法，应用到在线中位数排序赢家问题，产生了一个可以用一句话总结的非常有趣的算法。每一次从 d 个 voters 中，每个表中中的元素访问有序表直到一些 candidates 在超过半数的表中出现，这就是赢家。我们把这个算法称为 MEDRANK 算法。我们接下来会说明 MEDRANK 不是一个好的算法，但是如果超过某个常数，对于每个常例来说，在同等级可以按照顺序排序访问有序表的算法中，这是一个最好的算法。事实上，尽管我们同时允许顺序算法和任意随机访问，这个算法下对每一个实例需要花费一个常数因子的最好情况时间。这个概念在[11]中称为实例最佳性。我们将 MEDRANK 算法一般化为以正常方式寻找最前 k 个目标。例如，在赢家已经找到的情况下，我们将算法延续到访问有序表直到在超过一半的表中发现第二个元素——这是排名为 2 的元素。这个一般化算法也是实例最佳的。

近似最近邻居 中位数排序聚合可以和另一个在 Kleinberg 创造性的工作[18]之后经常在最近邻居文献中考虑到的强大的构思结合在一起。这个构思是关于对 d 维空间中任意直线的投射问题的。明确来说，我们将在第二节展示这个构思，使用一种第一次出现在[18]中的简单的几何引理，如果我们将 n 个数据库中的点（和查询点）投射到 m 为空间中，其中 $m = O(\epsilon^{-2} \log n)$，并且在投影的数据中运行 MEDRANK 算法，那么有很大的可能性，根据 MEDRANK 算法得到的赢家与欧氏空间下查询点的最近邻居只有ε的误差。(我们认为，如果 c1 属于 D，那么 d(c,q)小于等于(1+ε)d(c1,q)，c 是 q 的ε误差下的最近邻居，其中 d(⋯)是欧氏空间。)

1.3 组织

这篇论文剩余部分内容组织如下。第二节展示了关于 MEDRANK 算法和其他相关算法的技术结果，并且用对这些算法的正式描述进行总结。第三节描述了我们的实验并对其进行分析。我们的实验包括最近邻居的两个最主要的应用——简单搜索和分类。在两种情形下，无论是从品质还是效率上来讲，我们发现聚合方法更能获得更好的结果。我们将在第四节做一些总结性的结论。

2 框架和算法

在这一节的第一部分，我们描述一下框架，包括关于排序聚合和实例最优算法的一些必要的预先说明。在这一部分有两个主要的技术结果：（1）从ε近似欧氏空间最近邻居问题简化到在具有 n 个 candidates 和 $O(\varepsilon^{-2}\log n)$个 voters 的选举中使用最佳中位数排序找到 candidate。（2）对算法 MEDRANK 的证明。对数据库和查询来说，其能够只使用顺序访问进入 d 个有序表中，并且相比较于其他同类算法最多消耗一个常数因子。因此，MEDRANK 在计算中值赢家时在数据库模型中是实例最优的，并且可以得到可证明的近似最近邻居。

2.1 排序聚合，最近邻居和实例最优算法

2.1.2 关于最近邻居的算法

对 d 维空间中任意选择的一条直线投影数据的概念是在 Kleinberg 的最近邻居搜索文章中提出的。明确来说，考虑点 q 属于 R^d，并且使 u，v 属于 R^d，那么有 d(v,q) > (1+ε)d(u,q)。假设我们在 d 维空间任意取一个单位向量 r，实现这个目标的一种有效方法是根据标准正态分布 N(0,1)取 d 个坐标 r1, …, rd 作为 i.i.d 任意变量，并且把这些向量一般化为单位向量。然后我们在 r 上投影 u,v,q。用<.,.,.>表示一般的内积，那么直观上来说，我们希望 u 的投影<u, r>比起<v, r>更接近<q, r>。也就是说，我们希望<u-q, r>比<v-q, r>要小。以下的引理揭示了一个对该事实更加正式的解释：

引理 3：([18])假设 x, y 属于 R^d，并且使 ε>0，那么有||y|| > (1+ε)||x||。如果 r 是 R^d 中的任意一个单位向量，那么有 Pr[<y, r> <= <x, r>] <= 1/2-ε/3。

通过使 x=u-q, y=v-q, 我们很容易得到<u-q, r>比<v-q, r>要小的概率至少是 1/2-ε/3。

现在令 q 为查询点，w 是 D 中最靠近 q 的点，且 $B = \{x \in D \mid d(x,q) > (1+\epsilon)d(w,q)\}$。考虑到 x 是 B 中固定的一个点。如果我们取一个任意向量 r，根据 D 上这些点与 q 在 r 上投影的距离进行排序，那么 w 有 1/2-ε/3 的概率排在 x 的前面。假设我们取几个任意向量 r1，…，rm，然后如下在 D 上创建关于这些点的 m 个有序表：第 j 个表是根据他们在 rj 上的投影对 D 上的点排序得到的。W 排名在 x 之前的表的理想数目大概是 m(1/2-ε/3)。确实，由切尔诺夫标准范围可知，如果 $m=a\log\varepsilon^{-2}$，那么至少有 $1-1/n^2$ 的概率，我们至多有 m(1/2-ε/6)张 w 排在 x 之前的表。将所有 x 属于 B 的误差概率叠加起来，我们可以看到这意味着至少有 1-1/n 的概率，我们有

m(1/2-ε/6)张 w 排在 B 中每一个 x 之前的表。特别来说，m 张表中的 w 的中位数排序比 x 的中位数排序要更好。因此，如果我们计算在 m 张表中具有最佳中位数排序的点 z，那么(有 1-1/n 的概率) 可以说，z 不属于 B 中的元素，因为它满足 d(z, q)<=(1+ε)d(w,q)。我们总结了如下的论据：

定理 4：令 D 是 R^d 中一系列的 n 个点。令 r1，…，rm 是 R^d 中任意的单位向量，其中 m= alogε^{-2}。令 q 是 R^d 中的任意一点，并且给出定义，D 中 n 个点的有序表 Li 是以 n 个点与 ri 下投影的 q 的距离升序排列的，其实 1<=i<=m。对于 D 中的每一个元素 x，令 $medrank(x) = median(L_1(x), \ldots, L_m(x))$。并且 z 是 D 中的一个成员，那么 medrank(z)最小。并且，有至少有 1-1/n 的概率，对于 D 中的每一个 x，都有 d(z, q)<=(1+ε)d(x,q)。

事实上，以上的论证体现了更多的东西。令 q 是一个查询点，w，w2 分别是 D 上第一和第二靠近 q 的点。定义集合 B2={x 属于 D| d(x, q)>(1+ε)d(w2,q)} (考虑到 B2 属于 B，则 B 定义为{x 属于 D| d(x, q)>(1+ε)d(w,q)})通过相似的论证，我们可以得到有很高的概率，w2 的中位数排序比 B2 中的任意一个元素都要好，这意味着有第二最佳中位数排序的 z2 必然满足 d(z2, q) <= (1+ε)d(w2,q)。相似的，对于任意常数 k，相应的实现了第三最佳中位数排序的 z3，…，zk 同样满足 d(zj, q) <= (1+ε)d(wj,q)，wj 表示第 j 靠近 q 的点。

对于这个计划的目的，我们不能针对每个查询对数据库中的 n 个点排序 m 次。相反，作为预处理的一部分，我们创建了 m 个关于 D 中 n 个点的表。第 i 个有序表基于他们与第 i 个任意向量 ri 的投影的值对这些点进行排序。第 i 个有序表是这样的形式 $(c_1^i, v_1^i), (c_2^i, v_2^i), \ldots, (c_n^i, v_n^i)$，其中对于每一个 t，$v_t^i = \langle c_t^i, r_i \rangle$，$v_1^i \leq v_2^i \leq \ldots v_n^i$，和 $c_1^i, \ldots, c_n^i$ 是 1，…，n 的排列。给定 R^d 空间的一个查询 q，我们首先计算 q 关于 m 个任意向量的投影。对每一个 i，我们在第 i 个有序表中定位<ri, q>，也就是说，找到一个 t，使其满足 $v_t^i \leq \langle r_i, q \rangle \leq v_{t+1}^i$，并且初始化两个光标到 v_t^i 和 v_{t+1}^i。c_t^i 和 c_{t+1}^i 中的一个点现在是数据库中投影最靠近 q 的投影的点。通过合适地移动两个光标中的其中一个上或下，我们可以创建一个数据库中的点是根据投影到 q 的距离升序排列的有序表。这个结果体现在以下顺序访问 m 个表的形式中：一般来说，取 R^d 空间中的一个查询 q，并且初始化 2m 个光标，那么将会返回第 i 个表中的下一个元素。

在更多内存消耗和预处理的代价下，我们也可以采取对 n 个有序表的目录进行随机访问。那么，给定一个属于 D 的点 x，正常运行下会返回 x 在第 i 个有序表中的排序。我们的 MEDRANK 算法并不需要这样的随机访问。

2.2 算法的总结

我们现在来说明关于 MEDRANK 算法和两个相关算法 OMEDRANK 和 L2TA 的正式描述。OMEDRANK 是针对减少运行时间的探索性改进算法，而 L2TA 是计算每一个坐标都支持顺序访问和随机访问的模型中欧氏空间最近邻居的实例最优算法的极限算法的实现。我们将通

过线性浏览数据库所有数据找到最近邻居的直接算法用 L2NN 来表示。

这些描述都是标准的“伪代码”风格。并且，我们会描述一些步骤如何找到赢家，找到最前 k 个元素的扩张步骤也十分直接。

我们假设我们有一个包含 R^m 中 n 个点的数据库，m=d 或者 $m=O(\varepsilon^{-2}\log n)$。对 c 属于 D，1<=i<=m，我们用 ci 表示 c 在第 i 个坐标的值。

算法 MEDRANK 是聚合算法中的一个成员，我们通过考虑 50%位置来强化中位数的概念。我们介绍了在 MEDRANK 中的参数 MINFREQ 来使这个值不同于其他分位数。尽管具有其他 MINFREQ 值的算法表面上与最近邻居没有任何的关系，我们希望他们也成为最好的聚合算法。在被宣布为赢家之前，表中这些元素的数目是参数 MINFREQ 严格下界。取中位数排序则相应的 MINFREQ=0.5。通过增大 MINFREQ，我们希望在对数据库进行更多探查的代价下提高质量，从而实现质量代价的平衡。

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. $f_c > \text{MINFREQ} * m$ **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.

我们描述的第二个算法 OMEDRANK，是基于以下对 MEDRANK 的观察的。并不是比较 vi，hi 和 vi，li 的值和选择一个最靠近 qi 的点，我们将考虑 ci，hi 和 vi，li 两个元素。因为我们不需要执行任何随机访问，这将增加我们考虑为 S 中成员但在许多比较中保存的元素的数目。

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. $f_c > \text{MINFREQ} * m$ **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，我们将通过计算欧氏空间最近邻居来描述实例最佳算法；这个算法是“极限算法”或者 TA 对计算欧氏空间最近邻居的一个应用。这个算法，我们也称为 L2TA，可以用来取代简单的最近邻居算法，并且运行速度更快。

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 $\tilde{c} \notin S$ 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.实验

3.1 数据收集

我们实验程序包括我们相应称为 STOCK 和 HW 的两个数据集合。在下面的内容中，我们将详细地描述这两个数据集合的内容。通过使用对于每个类在合适和高维度空间下的特征向量，我们可以将相似的分类问题看成最近邻居问题。

第一个数据集合 STOCK 是从几家美国公司的历史股市数据中得到的。这些数据首先从雅虎的商业页面中收集而来并且包含了 7999 家公司的历史中所有记录的股票价格，但不包括他们公共的基金。每一家公司的数据都会分成两个部分代表 100 个连续的交易日子（如果剩余的日子有余数则会舍弃）。我们假设在最开始的一个交易日子里对某只股票投资 1 美元，并且记录这一美元在 100 个交易日期里面的进度。这将在一个 100 维度的空间里面创建一个特征向量。这个步骤，对每一家公司都不断重复，将会在一个 100 维度的空间里面产生关于 145619 个向量的数据集合。做这种分离数据的方法主要有两个原因：首先，在维持非平凡维度的时候增大数据集合的数目；其次，可以在合适的时间窗口比较不同的股票。

第二个数据集合公共的手写数据 MNIST 数据库中得到的。原始数据是由一个测试的 60000 个标签样本测试数据和 10000 个样本测试数据组成的。每一个灰度图片的大小是 28x28 且用标签从 0 到 9 进行标号。每一张图片的特征向量是 784 像素。因此，每一个向量都在 784 维的空间中。因为我们只是对最近邻居分类问题感兴趣，我们将这两个集合收缩成为一个包含 70000 个向量的简单集合。

为了给我们的实验增加更多的多样性，这两个数据集合被选择在多个方面上来进行比较。STOCK 是一个关于合适维度空间的大量数据集合，而 HW 是一个相对高维度来说相对较小的数据集合。另一个更重要的不同是，数据集合 HW 是含蓄的聚合数据，因为它是从 10 个优先的级别中获得的。另一方面，没有自然的聚合语义可以很容易的与 STOCK 联系起来。最后，考虑每个问题的数据集合都是不同的——对 STOCK 数据的相似搜索和对 HW 数据的分类操作。在对字符的识别上，在 HW 数据集合中，是一个具有争议性的近似最近邻居的重要应用，在这里，相对于找到确切的最近邻居，对字符进行准确的分类显得更加的重要。

我们决定通过在主内存中对所有的数据进行排序来实施所有的实验，然而一些硬件上的限制使我们不能对大量的数据进行操作。但是，注意到，通过强行在主内存中加载所有的数据，我们只能使用 L2NN 和 L2TA 算法，稍后我们将使用 MEDRANK 和 OMEDRANK 进行与之比较。如果大量的数据储存在第二内存中，这些算法的代价将会更加的高昂，正如他们试图访问数据库中大块的空间，这将导致硬盘访问的增加。

3.2 设置

为了学习这些算法在简化空间中的表现，我们对这些数据进行任意的投影操作。对于 STOCK，我们将数据投影到维度为 DIM=10，20，30，40，50 的空间。对于 HW，我们将数据投影到维度为 DIM=20，40，60，80，120，160。对于 HW，采集每个向量的标签是为了决定分类的误差。参数 MINFREQ 被选择为 0.5，0.6，0.7，0.8，0.9，这个参数在 MEDRANK 和 OMEDRANK 的探索深度具有一定的影响。

我们使用 C++执行算法 L2TA，L2NN，MEDRNAK 和 OMEDRANK。我们的实验在 0.5GRAM，1GHZ 的奔腾机器上运行。我们对数据的选择确保了它可以适合在主内存上保存。

对于原始的维度和每一个简化的维度，且对 MINFREQ 的每一个值，算法在 STOCK 和 HW 上运行 1000 个查询。这些查询是从相同的数据集合中任意选择出来的。在下面描述的

不同的参数，平均值都在这 1000 个查询之上。

3.3 参数学习

（1）时间。我们学习算法的基础运行时间来计算最前的 10 个结果。这些运行时间包括特别查询预处理。因为在全维度数据上的 L2NN 可以被认为是“相对真相”的一个合理的近似，我们将每个算法的运行时间与 L2NN 在全维度数据上的运行时间来进行比较。

（2）质量。我们对 STOCK 和 HW 使用关于质量的两种不同概念。对于 STOCK，如下描述。令 q 是一个查询，p 是算法对 q 返回的数据集合中最前的一个点，p*是 L2NN 在全维度数据上对相同的 q 返回的数据集合的点。直观上，p*是“正确的答案”（一个确切的最近邻居）。这个质量可以定义为 d(p, q)/d(p*, q)。

对于 HW 的情况，它的质量定义如下。回想我们从 HW 数据中收集的标签。令ε是我们算法对一系列查询的分类误差，ε*是 L2NN 在全维度空间数据中对相同集合的查询数据的分类误差；这里，分类误差是算法返回的标签不同于查询的真正的标签的部分查询。这个质量定义为ε/ε*。这个的最主要的原因，不是展示相对的分类误差，而是分类误差不仅是最近邻居或者聚合的一个函数，还是优先特征集合的函数。

因此，两个质量都是根据 L2NN 在全维度数据的执行来定义的。

（3）探索深度和部分访问。回想算法 L2TA，MEDRANK 和 OMEDRANK 都不会对完整数据库进行访问。对于 MEDRANK 和 OMEDRANK 这种通过顺序访问数据库的算法来说，我们记录这些顺序访问的次数。事实上，我们记录这些访问的次数并且输出最前的 10 个结果。

我们预测探索深度与数据库中每 m 个表中最近的那个点的排序有关。（我们谈到期望值，因为这 m 个表是按照一定的概率来提供的。）我们对量的排序 w 的分类进行计算，其中 w 是针对查询 q 的赢家。这些分布通过计算平均 1000 个任意查询。图一表示了这样的分布，；rank（w）的数学期望大概在 0.13，意味着我们希望 MEDRANK 和 OMEDRANK 只需要索 13%的数据。

算法 L2TA，除了顺序访问，同样使用任意访问。我们也记录这些信息。

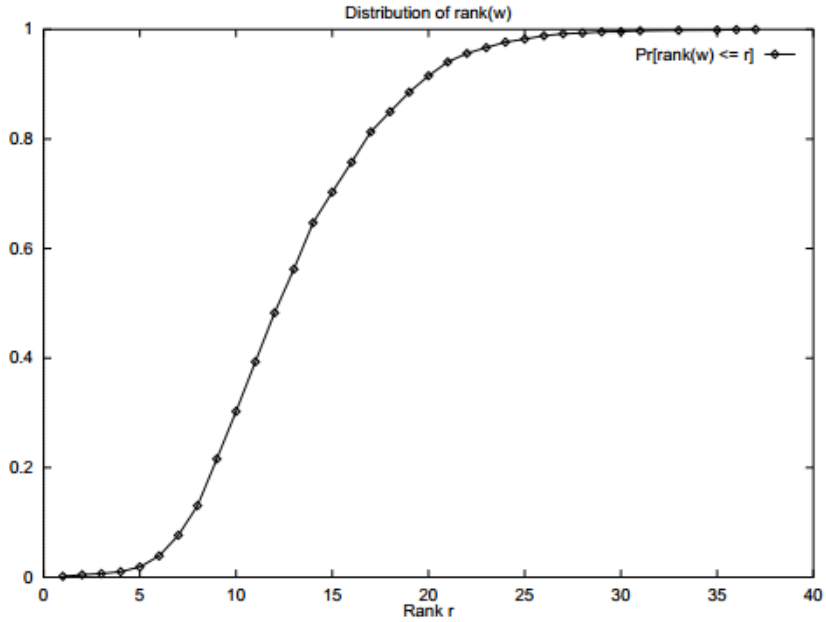


Figure 1: Distribution of rank(w).

3.4 结果

为了避免读者被大量的数据淹没，我们只采取基于 STOCK 和 HW 实验的基础数据的一个子集。相应的在表 1 和表 2 表示：

	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。正如这些表中所看到的，尽管在全维度数据上，MEDRANK 和 OMEDRANK 的运行时间比 L2NN 要小得多（大概只是 L2NN 所需时间的 35%-45%）。在投影数据上，MEDRANK 和 OMEDRANK 大概快 2 个量度。这些算法甚至在 MINFREQ 数值很大时仍然比 L2NN 要快。

我们认为差异之所以这么巨大主要是数据从磁盘中读入。而且，如果我们计算运行时间并当作计算最前结果的时间，MEDRANK 和 OMEDRANK 的执行结果更加引人注目。

算法 L2TA 对 STOCK 数据在低维度上演示有很高的速度，但在高维度的结果却差强人意，并且在 HW 数据的执行上比 L2NN 还要糟糕。这可以归因于算法的页记录努力。

（2）质量。这些表的数据在 MEDRANK 和 OMEDRANK 的质量上争议是很大的。对 STOCK 数据，估计因子大概是 2，意味着使用这些算法找到的点与最佳的点至多有 2 个因子的误差。而 L2TA 将会实际找到最近邻居，因此匹配在相应维度下 L2NN 的质量。一个需要注意的更加重要的与最近邻居相距 2 个因子的点在令人惊讶的相当小的运行时间下找到。这个提高对于 HW2 数据并没有很引人注目；在大概 6%的运行时间下，我们能够实现大概 5 次甚至更多的误差。

3.4.1 其他数据

我们在图 2-5 提供了一些关于算法的其他数据，包括 :MINFREQ 参数关于时间的影响，算法搜索数据库的深度和它与运行时间的相关性，MEDRANK 算法作为关于 MINFREQ 和最前 k 个参数方程提供的解的质量，L2TA 算法的搜索深度和访问数据。关于图片的描述点明了主要的观察点。

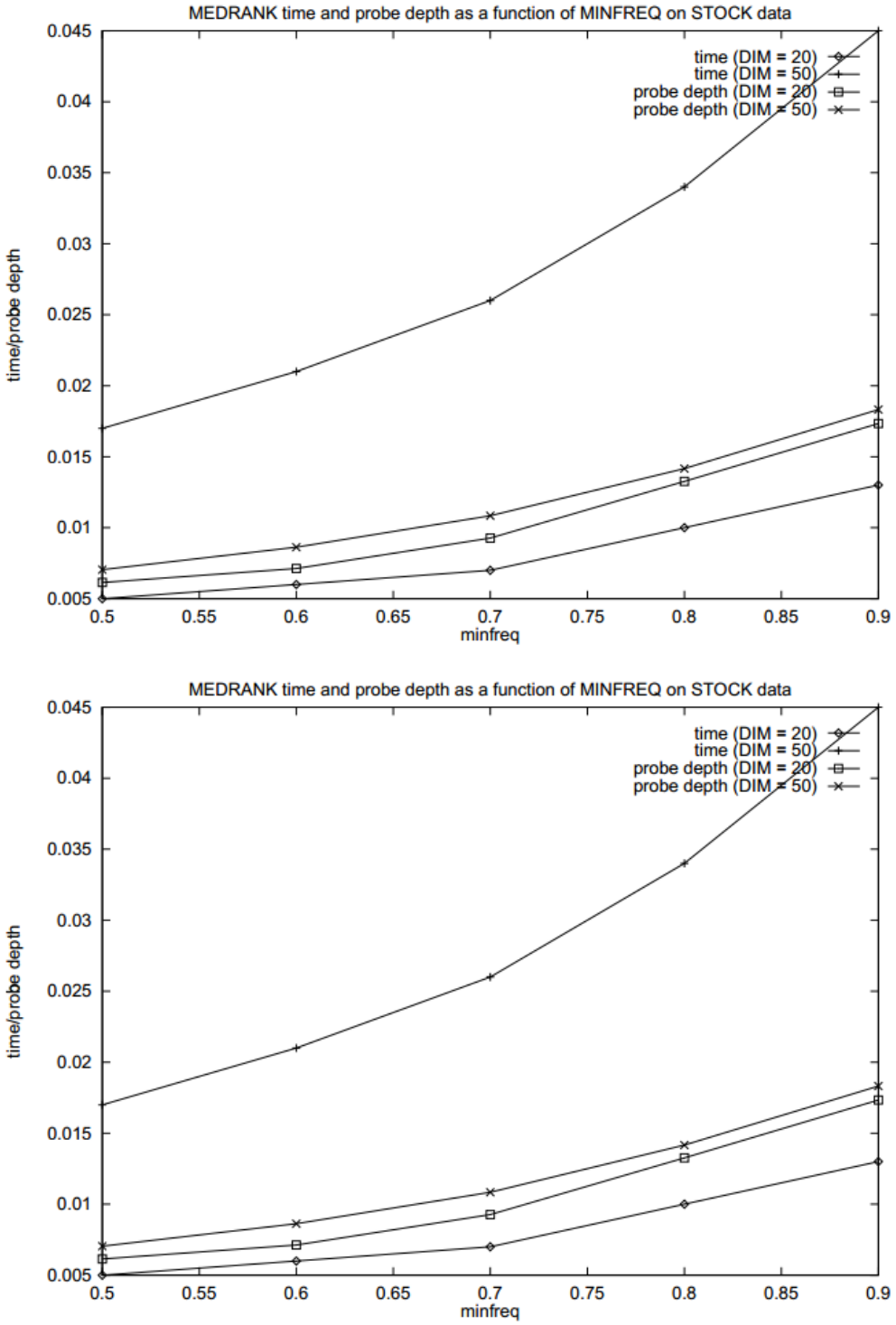


Figure 2: MEDRANK **time and probe depth as a function of** MINFREQ **on** STOCK **and** HW **data. Notes: (1) In both cases, dimension has almost no effect on the probe depth. (2) Even at** MINFREQ $= 0.9$**, time taken is very small.**

3.4.2 推论

（1）MEDRANK 和 OMEDRANK 都极度地快并且只需要查询数据库的一小部分尽管 MINFREQ 增加到 0.9（见图 2）。因此，这些算法都是非常地对数据库友好的和提供极度高效和 L2NN 的有效的替代。

（2）如果只关心大概的最近邻居，那么在低维度投影数据经常是十分有效的步骤。当保存相关性的时候，任意投影将减少杂音的影响。在投影数据上，当运行时间足够好的情况下，这些算法的质量大概与 L2NN 在相同数据上的质量一致。当投影至少减少了一个量度的时候，这些算法的深度搜索见图 5。我们总结出，如果只是满足近似最近邻居，那么投影是一个很好的主意，在投影数据上，MEDRANK 和 OMEDRANK 是 L2NN 很好的替代。

（3）计算 MEDRANK 和 OMEDRANK，我们在几种情况中发现，在保持结果质量的时候，OMEDRANK 的速度比 MEDRANK 快 20%以上。

（4）参数 MINFERQ 根据其对 MEDRANK 和 OMEDRANK 的重要性有不同的作用。对 STOCK, 我们发现 MINFERQ 没有体现重要的作用 ;因此, 这足够使它维持很低的数值(0.5)，恰好可以获得最佳的运行时间。对 HW，它可以减少误差（见图 3)。但是，有人会怀疑，它也会影响这些算法的搜索深度。但是，在指向这些算法的稳定性的时候，搜索深度仍然保持着比数据库的大小要小 1 到 2 个量度。

（5）我们测试 MEDRANK 要跑多远才不能覆盖它所提供的前 10 个结果这个问题。图 4 将这表示成关于最前结果的函数。正如我们所能看到，在获得第一和第十的结果上并没有很大的差别。

（6）我们总结出 L2TA 对最近邻居问题提供非平凡但不是在低维下速度上引人注目的提高，并且当维度升高时表现更差。（见表 1 和 2）。更进一步对这进行确定是展示了 MEDRANK 的深度搜索和被作为维度函数的 L2TA 访问的部分数据库的深度搜索的图 5。很容易看出 L2Ta 访问很大一部分的数据库，而 MEDRANK 只访问很小的一部分。

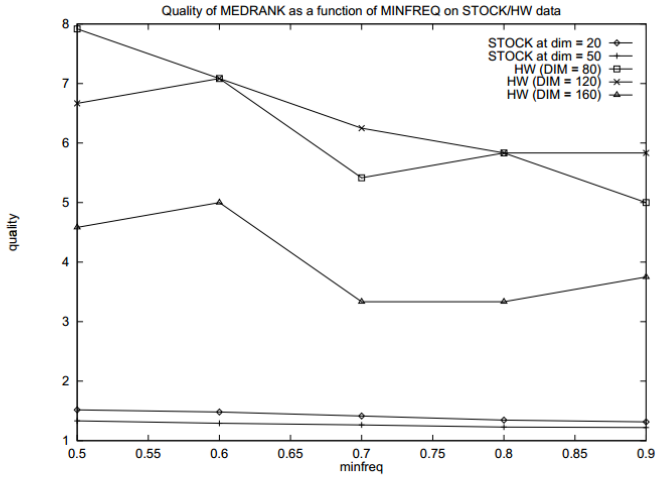


Figure 3: Quality of MEDRANK as a function of MINFREQ on STOCK/HW data. Notes: (1) HW data shows much more improvements as a function of dimensionality than STOCK data. (2) MINFREQ seems not to affect results on STOCK data much, but on the HW data, a value of roughly 0.7 seems to be the best.

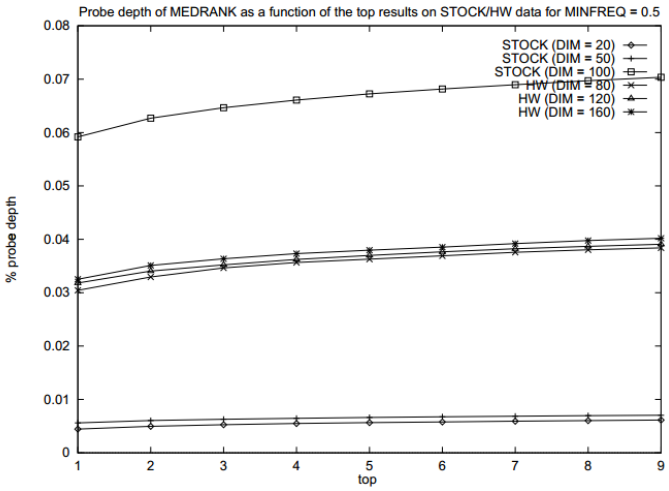
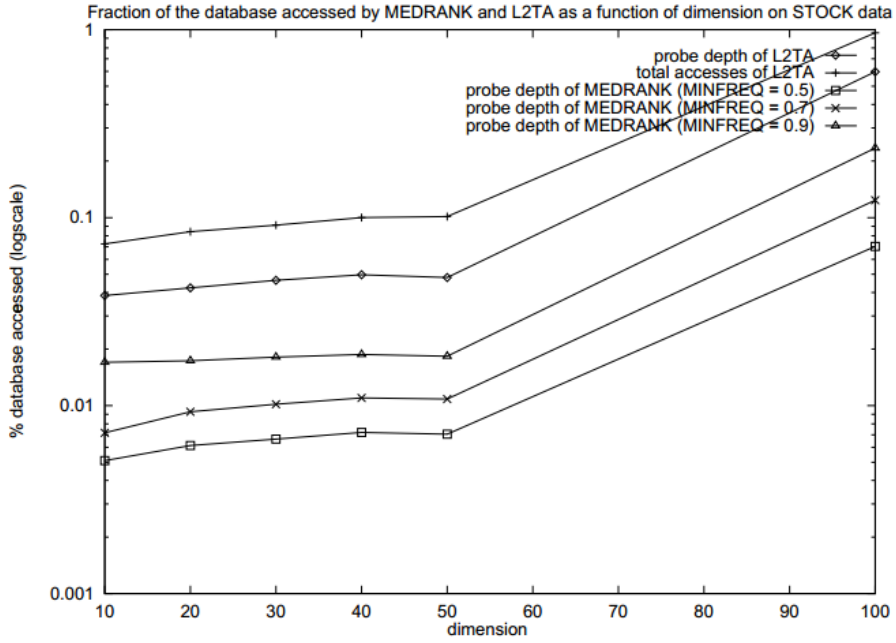


Figure 4: Probe depth of MEDRANK **as a function of the top results on** STOCK/HW **data for** MINFREQ**=0.5. Notes: (1) Dimensionality reduction causes significant improvement in the probe depth for** STOCK **data (compare 100 dimensions vs. lower dimensions). Note that we did not conduct the** HW **data 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.**



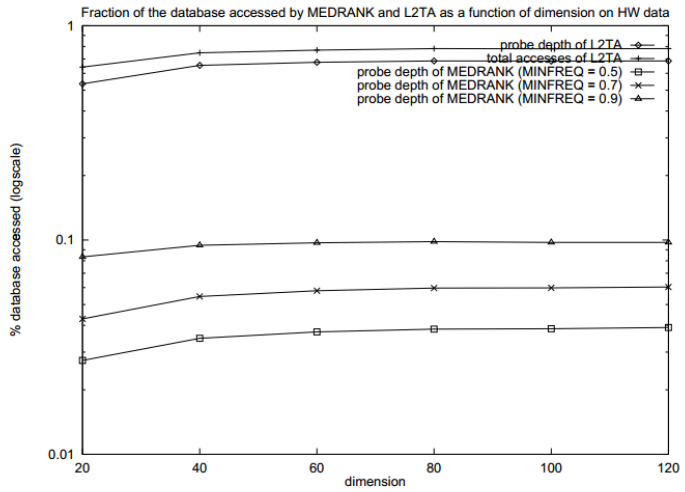


Figure 5: Fraction of the database accessed by L2TA and MEDRANK as a function of dimension on STOCK and HW data. Note: (1) MEDRANK and OMEDRANK access an order of magnitude fewer elements of the database than L2TA.

4 结论

我们介绍了一个做相似搜索和分类的新的方法——排序聚合。我们将查询和 candidate 看成多维空间的一些点。每一个坐标都当做一个基于与查询的相对坐标的距离排序的 voter。而赢家是有最高聚合排序的那些点。将维度的简化结合起来，这个方法可以得到一个简单的，数据库友好的，可以提供一个非常好的近似最近邻居的算法。这个算法非常的有效，经常查询不到 5%的数据就可以得到一个非常高质量的结果。我们认为这个方法在概念上为它的准确性而自豪，而不仅仅是计算大概最近邻居。我们的结果同样强调了中位数排序聚合是高效和有用的排序聚合。

一个有趣的研究方向是为了考虑那些几个点被强制有相同的排序的小定义域和使我们的方法能够适应探索这个特点。