1 Introduction

Gregory Shakhnarovich, Piotr Indyk, and Trevor Darrell

The nearest-neighbor (NN) problem occurs in the literature under many names, including the *best match* or the *post office* problem. The problem is of significant importance to several areas of computer science, including pattern recognition, searching in multimedia data, vector compression, computational statistics, and data mining. For many of these applications, including some described in this book, large amounts of data are available. This makes nearest-neighbor approaches particularly appealing, but on the other hand it increases the concern regarding the computational complexity of NN search. Thus it is important to design algorithms for nearest-neighbor search, as well as for the related classification, regression, and retrieval tasks, which remain efficient even as the number of points or the dimensionality of the data grows large. This is a research area on the boundary of a number of disciplines: computational geometry, algorithmic theory, and the application fields such as machine learning. This area is the focus of this book, which contains contributions from researchers in all of those fields.

Below we define the exact and approximate nearest-neighbor search problems, and briefly survey a number of popular data structures and algorithms developed for these problems. We also discuss the relationship between the nearest-neighbor search and machine learning. Finally, we summarize the contents of the chapters that follow.

1.1 The Nearest-Neighbor Search Problem

The *exact* nearest-neighbor search problem in a Euclidean space is defined as follows:

Definition 1.1 (Nearest neighbor) *Given a set P of points in a d-dimensional space $\Re^d$, construct a data structure which given any* query *point q finds the point in P with the smallest distance to q.*

The problem is not fully specified without defining the distance between an arbitrary pair of points p and q. Typically, it is assumed that this distance is induced by an l_s norm. That is, the distance between p and q is defined as $\|p - q\|_s$, where $\|x\| = (\sum_{i=1}^{d} |x_i|^s)^{1/s}$. Other (more general) notions of distance are possible as well.

A naive algorithm for this problem is as follows: given a query q, compute the distance from q to each point in P, and report the point with the minimum distance. This *linear scan* approach has query time of $\Theta(dn)$. This is tolerable for small data sets, but is too inefficient for large ones. The "holy grail" of the research in the area is to design an algorithm for this problem that achieves *sublinear* (or even *logarithmic*) query time.

The nearest-neighbor problem has been extensively investigated in the field of computational geometry. As a result of this research effort, many efficient solutions have been discovered for the case when the points lie in a space of *constant* dimension. For example, if the points lie in the plane, the nearest-neighbor problem can be solved with $O(\log n)$ time per query, using only $O(n)$ storage [36, 27]. Similar results can be obtained for other problems as well. Unfortunately, as the dimension grows, the algorithms become less and less efficient. More specifically, their space or time requirements grow *exponentially* in the dimension. In particular, the nearest-neighbor problem has a solution with $O(d^{O(1)} \log n)$ query time, but using roughly $n^{O(d)}$ space [11, 29].

At the same time, many data structures that use the optimal $O(dn)$ space are known (e.g., see the survey [9] for a review of nearest-neighbor data structures from the database perspective). These data structures often provide significant speedups on many data sets. At the same time, they are known to suffer from linear query time for "high enough" dimension (e.g., see [9], p. 365).

The lack of success in removing the exponential dependence on the dimension led many researchers to conjecture that no efficient solutions exist for this problem when the dimension is sufficiently large (e.g., see [31]). At the same time, it raised the question: Is it possible to remove the exponential dependence on d, if we allow the answers to be *approximate*? The approximate nearest neighbor search problem is defined as follows.

Definition 1.2 (c-Approximate nearest neighbor) *Given a set P of points in a d-dimensional space $\Re^d$, construct a data structure which given any* query *point q, reports* any *point within distance at most c times the distance from q to p, where p is the point in P closest to q.*

During recent years, several researchers have shown that indeed in many cases approximation enables reduction of the dependence on dimension from exponential to polynomial. A survey of these results can be found in [23]. In addition, there are many approximate nearest-neighbor algorithms that are more efficient than the exact ones, even though their query time and/or space usage is still exponential in the dimension. This includes the algorithms given in [3, 5, 12, 26, 10, 20, 1]. The algorithm in [5] has an efficient implementation (ANN); see [2] for details.

In the following we present an brief overview of three data structures: kd-trees (and relatives), balltrees, and locality-sensitive hashing (LSH).

1.1.1 Kd-Trees

The kd-tree [8] is a data structure invented by Jon Bentley in 1975. Despite its fairly old age, kd-tree and its variants remain probably the most popular data structures used for searching in multidimensional spaces, at least in main memory. According to Google Scholar, the paper [8] that introduced this data structure is one of the most cited papers in computational geometry, with 734 citations as of June 2005.

Given a set of n points in a d-dimensional space, the kd-tree is constructed recursively as follows. First, one finds a median of the values of the ith coordinates of the points (initially, $i = 1$). That is, a value M is computed, so that at least 50% of the points have their ith coordinate greater-or-equal to M, while at least 50% of the points have their ith coordinate smaller than or equal to M. The value of x is stored, and the set P is partitioned into P_L and P_R, where P_L contains only the points with their ith coordinate smaller than or equal to M, and $|P_R| = |P_L| \pm 1$. The process is then repeated recursively on both P_L and P_R, with i replaced by $i + 1$ (or 1, if $i = d$). When the set of points at a node has size 1, the recursion stops.

The resulting data structure is a binary tree with n leaves, and depth $\lceil \log n \rceil$. In particular, for $d = 1$, we get a (standard) balanced binary search tree. Since a median of n coordinates can be found in $O(n)$ time, the whole data structure can be constructed in time $O(n \log n)$.

For the problem of finding the nearest neighbor in P of a given query q, several methods exist. The first one was suggested in the original paper. However, it was quickly superseded by a different procedure, introduced in [25] (the original procedure was deleted from the journal version of [8]). The latter search procedure is as follows. The search starts from the root of the tree, and is recursive. At any point in time, the algorithm maintains the distance R to the point closest to q encountered so far; initially, $R = \infty$. At a leaf node (containing, say, point p') the algorithm checks if $\|q - p'\| < R$. If so, R is set to $\|q - p'\|$, and p' is stored as the closest point candidate. In an internal node, the algorithm proceeds as follows. Let M be the median value stored at the node, computed with respect to the ith coordinates. The algorithm checks if the ith coordinate of q is smaller than or equal to M. If so, the algorithm recurses on the left node; otherwise it recurses on the right node. After returning from the recursion, the algorithm performs the "bounds overlap ball" test: it checks whether a ball of radius R around q contains any point in $\Re^d$ whose ith coordinate is on the opposite side of M with respect to q. If this is the case, the algorithm recurses on the yet-unexplored child of the current node. Otherwise, the recursive call is terminated. At the end, the algorithm reports the final closest-point candidate.

It was shown in [25] that if the data and the query point are chosen independently at random from a random distribution (say from a d-dimensional hypercube $[0,1]^d$), then the procedure terminates in time $G(d)\log n$, for some function G. However, $G(d)$ is exponential in d, at least for values of d that are "small" compared to $\log n$; note that the running time is always at most $O(dn)$. It is easy to construct data sets which achieve this worst-case running time.

A different search procedure is obtained by modifying the order in which the nodes are visited. Specifically, several authors [4, 5, 6, 21] proposed to examine the nodes in the order of their distance to the query point (the exact definition of this process depends on the implementation). This enables the algorithm to locate a "close enough" neighbor much faster. For example, experiments in [4] indicate that this "priority" approach enables finding very good approximate nearest neighbors up to ten times faster than the original kd-tree search procedure.

Another bonus of the priority search approach is that one can prove worst-case guarantees for the running times of the resulting algorithm (for a somewhat modified data structure, called box-decomposition tree, see [5]). Further results of this type are given in [17].

1.1.2 Balltrees and Metric Trees

Balltrees, introduced by Omohundro in [34], are complete binary trees, where the leaves correspond to the data points and each interior (non-leaf) node corresponds to a ball in the data space. The ball associated with a given node is required to be the smallest ball that contains the balls associated with that node's children. Closely related to balltrees are *metric trees* [38], in which a node is constructed by thresholding the distance between the points it contains and a pivot.

In contrast to kd-trees, the regions associated with sibling nodes in balltrees and metric trees may intersect and do not have to cover the entire space. This may allow a more flexible coverage of the space, reflecting the structure inherent in the data, and thus make the data structure more efficient for data embedded in high-dimensional spaces. A number of algorithms have been proposed for fast construction, updating, and searching of these data structures [34, 38, 32]. For a more detailed description, see section 4.2 in chapter 4.

1.1.3 Locality-Sensitive Hashing (LSH)

LSH, as well as several other algorithms discussed in [23], is *randomized*. The randomness is typically used in the construction of the data structure. Moreover, these algorithms often solve a *near*-neighbor problem, as opposed to the *nearest*-neighbor problem. The former can be viewed as a decision version of the latter. Formally, the problem definitions are as follows. A point p is an *R-near neighbor* of q if the distance from p to q is at most R.

Definition 1.3 (Randomized c-approximate near-neighbor) *Given a set P of points in a d-dimensional space $\Re^d$, and parameters $R > 0$, $\delta > 0$, construct a data structure which, given any* query *point q, does the following with probability $1 - \delta$: if there is an R-near neighbor of q in P, it reports a cR-near neighbor of q in P.*

Definition 1.4 (Randomized near-neighbor reporting) *Given a set P of points in a d-dimensional space $\Re^d$, and parameters $R > 0$, $\delta > 0$, construct a data structure which, given any* query *point q, reports each R-near neighbor of q in P with probability $1 - \delta$.*

Note that, in the second definition, the probabilities of reporting different R-near neighbors might *not* be independent. Also note that, in both problems, the value R is known during the preprocessing time. Therefore, by scaling the coordinates of all points, we can assume that $R = 1$.

Among the algorithms discussed in [23], the LSH algorithm is probably the one that has received most attention in the practical context. Its main idea is to hash the data points using several hash functions so as to ensure that, for each function, the probability of collision is much higher for points which are close to each other than for those which are far apart. Then, one can determine near neighbors by hashing the query point and retrieving elements stored in buckets containing that point.

The LSH algorithm can be used to solve either the approximate or the exact near-neighbor problem. It relies on the existence of *LSH functions*, defined in the following manner. Consider a family $\mathcal{H}$ of hash functions mapping $\Re^d$ to some universe U.

Definition 1.5 (Locality-sensitive hashing) *A family $\mathcal{H}$ is called* (r, cr, P_1, P_2)-sensitive *if for any $p, q \in \Re^d$*

– *if $\|p - q\| \leq R$ then $\mathrm{P}_{\mathcal{H}}[h(q) = h(p)] \geq P_1$,*
– *if $\|p - q\| \geq cR$ then $\mathrm{P}_{\mathcal{H}}[h(q) = h(p)] \leq P_2$.*

In order for a LSH family to be useful, it has to satisfy $P_1 > P_2$.

An LSH family can be utilized as follows. Given a family $\mathcal{H}$ of hash functions with parameters (r, cr, P_1, P_2) as in the above definition, we amplify the gap between the "high" probability P_1 and the "low" probability P_2 by concatenating several functions. In particular, for k and L specified later, we choose L functions $g_j(q) = (h_{1,j}(q), \ldots, h_{k,j}(q))$, where $h_{t,j} (1 \leq t \leq k, 1 \leq j \leq L)$ are chosen independently and uniformly at random from $\mathcal{H}$. During preprocessing, we store each $p \in P$ (input point set) in the bucket $g_j(p)$, for $j = 1, \ldots, L$. Since the total number of buckets may be large, we retain only the nonempty buckets by resorting to hashing the values $g_j(p)$.

To process a query q, we search through the buckets $g_1(q), \ldots, g_L(q)$. Two concrete strategies are possible:

1. Interrupt search after finding the first L' points (including duplicates).
2. Continue search until all points from the buckets are retrieved.

It is shown in [24] that the first strategy, with $L' = 3L$, enables solving the *randomized c-approximate near-neighbor problem*, with parameters R and δ, for some constant $\delta < 1$. To obtain that guarantee, it suffices to set L to $\Theta(n^\rho)$, where $\rho = \frac{\ln 1/P_1}{\ln 1/P_2}$. Note that this implies that the algorithm runs in time proportional to n^ρ, which is sublinear in n if $P_1 > P_2$.

On the other hand, in practice, the second strategy appears to be more popular, presumably since it avoids the need to specify the additional parameter L'. Also, there is no need to specify c, since the algorithm can solve the exact near neighbor problem, as described in the remainder of this paragraph. The analysis in [24] can be used to show that the second strategy enables solving the randomized *exact near-neighbor reporting problem* for a given parameter R. The value of δ depends on the choice of the parameters k and L; alternatively, for each δ one can provide parameters k and L so that the error probability is smaller than δ. The query time is also dependent on k and L: it could be as high as $\Theta(n)$ in the worst case, but for many data sets a proper choice of parameters results in a sublinear query time. See more details in chapter 3.

The quality of LSH functions is characterized by the parameter ρ. It was shown in [24, 19] that if the distance is measured according to the l_1 norm, then there exists a (R, cR, P_1, P_2)-sensitive family of functions with $\rho = 1/c$. In chapter 3 another family of functions is described which works if the distances are measured according to the l_s norm for $s \in (0, 2]$.

1.2 Nearest-Neighbor Methods in Learning and Vision

In the context of machine learning, nearest-neighbor methods are applied to supervised problems. Suppose one is given a *reference set*–a set of examples for which the target concept (a class label, function value, etc.) is known, and a *query*, for which such a value is unknown and is to be recovered. The nearest-neighbor approach consists of finding one or more examples most similar to the query, and using the labels of those examples to produce the desired estimate of the query's label.

This broad description leaves two important questions:

1. What are the criteria of similarity? The answer depends on the underlying distance measure as well as on the selection criteria, e.g., the k examples closest to the query for a fixed k, or the examples closer than a fixed threshold r. The choices of these criteria are known to have a very significant influence on the performance on the algorithm.
2. How are the labels from the neighbors to be combined? The simplest way is to take the majority vote, in classification setup, or the average,

in regression setup. However, more sophisticated methods, such as locally weighted regression, have been shown to produce sometimes dramatically better results.

The following fundamental property of nearest-neighbor classification has been shown by Cover and Hart in [15]. Let $R(n)$ be the expected probability of error of the M-category nearest-neighbor classifier for a training set of n examples. Then the limit $R = \lim_{n\to\infty} R(n)$ is bounded by $R^* \leq R \leq R^*(2 - MR^*/(M-1))$, where R^* stands for the *Bayes risk*– the probability of error of the optimal classifier given the distribution of the data. A similar result can be shown for the nearest-neighbor regression [13].

Unfortunately, these asymptotic results do not necessarily translate into similar bounds in practice when training sets of finite size are used. As has been shown by Cover in [14], the rate of convergence to this limit may be arbitrarily slow. A significant body of work has been devoted to analysis of nearest-neighbor performance on samples of finite size. Although no “distribution-free” bounds have been found, it is possible to characterize the finite sample risk of the nearest-neighbor classifier based on various properties of the input space and the data distribution (see [18, 35, 37] for some interesting results). Moreover, despite the lack of theoretical guarantees, it has often been observed that the nearest-neighbor classifiers perform very well in practice, and achieve accuracy equivalent to or even better than that of significantly more complicated methods; see, e.g., the extensive comparative study reported in [16].

Similar success has been seen in the application of nearest-neighbor methods to a variety of vision tasks, such as shape matching [22, 7] and object recognition [33, 30, 28]. However, in computer vision the limitations of some nearest-neighbor search methods have particularly high impact, since the data here is typically high-dimensional, and in many cases the databases required to sufficiently represent the visual phenomena are large. On the other hand, appropriate similarity measures in vision applications are often more expensive than the Euclidean distance. Furthermore, these measures may not even be metrics. All this poses a challenge for a practitioner interested in using an example-based technique in a computer vision problem. Possible solutions include randomized approximate algorithms, embedding techniques, and machine learning algorithms for adapting the data structures for the properties of the data in the given domain. The book describes some of the recent advances in these directions.

1.3 Contributions to this book

Theory

The book opens with two contributions from computational geometry. Chapter 2, by Clarkson, focuses on the formulation of exact nearest-neighbor search, and studies this problem in metric spaces. In particular, it investigates the various dimensionality properties that make the problem tractable in practice, and presents theoretical results. Chapter 3, by Andoni, Datar, Immorlica, Indyk, and Mirrokni, is devoted to a randomized algorithm that allows approximate as well as exact similarity search in very high dimensions in time sublinear in the size of the database. The algorithm is an extension of LSH, described in 1.1.3, to a family of distance metrics beyond the l_1 norm.

Learning

The unifying theme of the three chapters in the second part of the book is exploring the ways to make the nearest-neighbor approach practicable in machine learning application where the dimensionality of the data, and the size of the data sets, make the naive methods for nearest-neighbor search prohibitively expensive. In chapter 4, Liu, Moore, and Gray notice that in nearest-neighbor classification, what matters for the decision is the distribution of labels of the query's neighbors, rather than the neighbors themselves. They use this observation to develop new algorithms, using previously introduced balltree data structures, that afford a significant speedup while maintaining the accuracy of the exact k-nearest-neighbor classification.

Chapter 5, by Vijayakumar, D'Souza, and Schaal, is devoted to methods that build a local model of the target function in the vicinity of the query by finding its neighbors in the reference set. The chapter proposes an approach that extends such local learning methods to very high-dimensional spaces by exploiting the low intrinsic dimensionality within a neighborhood. Another contribution is a new Bayesian regression algorithm. The proposed framework has been shown to be fast and accurate in a number of applications, including real-time motor control tasks for robots.

Chapter 6, by Athitsos, Alon, Sclaroff, and Kollios addresses the issue that arises in the context of many applications of nearest-neighbor methods in learning, be they for classification, regression, or database retrieval. When computing the distance of interest is expensive, a significant reduction in search time may be gained if a cheap way exists to evaluate the relative distances from the query and eliminate the majority of the reference examples from contention, thus requiring the distance to be explicitly computed only on a small subset of the database. The approach proposed in this chapter is to learn an embedding of the original distance into a Euclidean space, in which the ranking with respect to the Manhattan metric reflects the ranking under the original distance. Connecting this chapter to the following ones, the approach is evaluated on two computer vision tasks: estimating the articulated pose of human hands and classifying sign language utterances.

Vision

The chapters in the third part of the book describe successful applications of LSH to vision tasks. Moreover, as some of the chapters show, the basic LSH framework may be adapted to better fit the properties of the data space and the task at hand.

In chapter 7, Shakhnarovich, Viola, and Darrell deal with a regression task: estimating articulated body pose from images of people. The basic idea here is to learn, using pairs of images with known poses, hash functions sensitive to the distance in the pose space. The resulting method, parameter-sensitive hashing, allows very fast example-based estimation of pose using a very large set of labeled examples. Chapter 8, by Grauman and Darrell, is concerned with database retrieval scenarios where the distance is known but very expensive to compute: the earth mover's distance (EMD) between contours. The authors propose a technique, based on multiresolution histograms, that embeds the contours into a Euclidean space of very high dimensions, and replaces EMD with the Manhattan distances between sparse points in that space, thus allowing application of very fast approximate search algorithms such as LSH. The experiments on data sets of handwritten digits and human body contours show that with this embedding, neighbor-based methods remain very accurate while gaining dramatic speedup.

In contrast to the previous chapters, the task of interest in chapter 9, by Shimshoni, Georgescu, and Meer, is an unsupervised one: clustering. Specifically, the authors show how LSH can be used to reduce the complexity of mean-shift clustering, and apply the resulting algorithm to the tasks of visual texture classification and image segmentation. They also propose significant modifications in the algorithms for constructing the data structure used to index the examples: the partitions in the hash tables are data driven.

Finally, in chapter 10 Frome and Malik present an application of LSH to another vision task: automatic classification of vehicles from their three-dimensional range scans. The approach proposed by the authors relies on LSH to perform fast search in a set of reference objects represented using shape context–a representation that has been shown to be useful but also to make distance calculations very costly. This chapter, too, presents a modification of the basic LSH algorithm: Associative LSH. In this algorithm, the results returned by LSH for a query are further refined by local search, improving the quality of the approximate nearest-neighbor and consequently affording a gain in classification accuracy.

References

1. S. Arya and T. Malamatos. Linear-size approximate Voronoi diagrams. *In Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 147–155, 2002.
2. S. Arya and D. Mount. ANN: Library for approximate nearest neighbor searching. `http://www.cs.umd.edu/~mount/ANN/`.
3. S. Arya and D. Mount. Approximate nearest neighbor searching. *In Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 271–280, 1993.
4. S. Arya, D. Mount, J. Storer, and M. Cohn. Algorithms for fast vector quantization. *Data Compression Conference*, pages 381–390, 1993.
5. S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *In Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, 1994.
6. J. Beis and D. Lowe. Shape indexing using approximate nearest neighbour search in high-dimensional spaces. *Proceedings of the IEEE Computer Society Conference on Computer Vision*, page 1000, 1997.
7. S Belongie, J. Malik, and J. Puzicha. Shape Matching and Object Recognition Using Shape Contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, April 2002.
8. J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509–517, 1975.
9. C. Bohm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys (CSUR)*, 33(3), 2001.
10. T.M. Chan. Approximate nearest neighbor queries revisited. *In Proceedings of the Thirteens Annual ACM Symposium on Computational Geometry*, pages 352–358, 1997.
11. K. Clarkson. A randomized algorithm for closest-point queries. *SIAM Journal on Computing*, 17:830–847, 1988.
12. K. Clarkson. An algorithm for approximate closest-point queries. *In Proceedings of the Tenth Annual ACM Symposium on Computational Geometry*, pages 160–164, 1994.
13. T. M. Cover. Estimation by the nearest neighbor rule. *IEEE Transactions on Information Theory*, 14:21–27, January 1968.
14. T. M. Cover. Rates of Convergence for Nearest Neighbor Procedures. In *Proc. 1st Ann. Hawaii Conf. Systems Theory*, pages 413–415, January 1968.
15. T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, January 1967.

16. D. Michie, D.J. Spiegelhalter, and C.C. Taylor, editors. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.
17. C. Duncan, M. Goodrich, and S. Kobourov. Balanced aspect ratio trees: Combining the advantages of kd-trees and octrees. *Journal of Algorithms*, 38:303, 2001.
18. K. Fukunaga and D. M. Hummels. Bias of nearest neighbor error estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(1):103–112, January 1987.
19. A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. *In Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, 1999.
20. S. Har-Peled. A replacement for Voronoi diagrams of near linear size. *Annual Symposium on Foundations of Computer Science*, 2001.
21. G. Hjaltason and H. Samet. Ranking in spatial databases. *Proceedings of the 4th Symposium on Spatial Databases*, pages 83–95, 1995.
22. D. Huttenlocher, G. Klanderman, and W. Rucklidge. Comparing images using the hausdorff distance,. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15, 1993.
23. P. Indyk. Nearest neighbors in high-dimensional spaces. *CRC Handbook of Discrete and Computational Geometry*, 2003.
24. P. Indyk and R. Motwani. Approximate nearest neighbor: towards removing the curse of dimensionality. *Proceedings of the Symposium on Theory of Computing*, 1998.
25. J.L. Bentley J.K. Friedman and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3:209–226, 1977.
26. J. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, 1997.
27. R. Lipton and R. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9:615–627, 1980.
28. S. Mahamud and M. Hebert. The optimal distance measure for object detection. In *IEEE Conf. on Computer Vision and Pattern Recognition*, Madison, WI, June 2003.
29. S. Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106:286–303, 1993.
30. B. W. Mel. SEEMORE: Combining color, shape, and texture histogramming in a neurally inspired approach to visual object recognition. *Neural Computation*, 9(4):777–804, 1997.
31. M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.
32. Andrew Moore. The anchors hierarchy: Using the triangle inequality to survive high-dimensional data. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 397–405. AAAI Press, 2000.

33. S. A. Nene and S. K. Nayar. A simple algorithm for nearest neighbor search in high dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):989–1003, September 1997.

34. S. M. Omohundro. Five balltree construction algorithms. Technical Report TR-89-063, International Computer Science Institute, Berkeley, CA, December 1989.

35. D. Psaltis, R. R. Snapp, and S. S. Venkatesh. On the Finite Sample Performance of the Nearest Neighbor Classifier. *IEEE Transactions on Information Theory*, 40(3):820–837, May 1994.

36. M. I. Shamos and D. Hoey. Closest point problems. *Proceedings of the Sixteen IEEE Symposium of Foundations of Computer Science*, pages 152–162, 1975.

37. R. R. Snapp and S. S. Venkatesh. Asymptotic derivation of the finite-sample risk of the k nearest neighbor classifier. Technical Report UVM-CS-1998-0101, University of Vermont, Burlingotn, Burlington, VT, October 1997.

38. J.K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40:175–179, 1991.