Fast *k*-Nearest Neighbor Classification Using Cluster-Based Trees

Bin Zhang, *Member*, *IEEE*, and
Sargur N. Srihari, *Fellow*, *IEEE*

Abstract—Most fast k-nearest neighbor (k-NN) algorithms exploit metric properties of distance measures for reducing computation cost and a few can work effectively on both metric and nonmetric measures. We propose a cluster-based tree algorithm to accelerate k-NN classification without any presuppositions about the metric form and properties of a dissimilarity measure. A mechanism of early decision making and minimal side-operations for choosing searching paths largely contribute to the efficiency of the algorithm. The algorithm is evaluated through extensive experiments over standard NIST and MNIST databases.

Index Terms—Nearest neighbor classification, nonmetrics, metrics, cluster tree.

1 Introduction

The k-nearest neighbor (k-NN) classification has been extensively used as a powerful nonparametric technique of pattern recognition [1]. However, the exhaustive k-NN search, which requires intensive dissimilarity computations—particularly for a large training set, becomes unacceptable. Accelerating the k-NN search has been an active research field in the past three decades. While nonmetric dissimilarity measures have been briefly explained in pattern recognition literature, most of the existing fast algorithms for the k-NN search are effective only with metric dissimilarity measures.

Algorithms for speeding-up the k-NN search fall into two categories: template condensation and template reorganization. Template condensation removes redundant patterns in a template set [2], [3], [4], [5] and template reorganization restructures templates for efficient search of k nearest neighbors [6], [7], [8], [9], [10], [11]. Incorporating the template condensation rules into template reorganization leads to an innovative algorithm, the condensation-based tree algorithm by Brown [12].

While a number of template reorganization algorithms rely on the essential properties (e.g., the triangle inequality) of metric dissimilarity measures [6], [9], [11], [13], the others, which are applicable for nonmetrics, are only effective in low-dimensional feature spaces [8], [10], [14]. One exception is the condensation-based tree algorithm, which is applicable for any dissimilarity measure, metric, or nonmetric. Algorithms exploiting the triangle inequality of metrics are more efficient in searching and classification [15]. However, many applications of pattern recognition in high-dimensional feature spaces often employ nonmetric dissimilarity measures for better performance, i.e., ℓ_p distances ($0 < p < 1$) [15], χ^2-statistic (or χ^2-distance) [16], Kullback-Leibler divergence (KL) [17], Jeffrey-divergence (JD) [17], the Sokal-Michener measure (SM) [18], Hausdorff distance [19], and deformable models [20], etc. The aforementtioned measures, except the SM measure, violate the triangle inequality. Deformable models are even not symmetric and SM measure does not obey relexitivity. Therefore, for these applications, most existing algorithms that accelerate k-nearest neighbor classification are either inapplicable or ineffective.

- *B. Zhang is with the Departments of Human Genetics and Biostatistics, School of Medicine, UCLA, Los Angeles, CA 90095-7088. E-mail: binzhang@mednet.ucla.edu.*
- *S.N. Srihari is with CEDAR, Department of Computer Science and Engineering, The State University of New York at Buffalo, 520 Lee Entrance, Amherst, NY 14228-2567. E-mail: srihari@cedar.buffalo.edu.*

Manuscript received 3 June 2002; revised 4 Mar. 2003; accepted 31 July 2003. Recommended for acceptance by Y. Amit.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 116680.

The condensation-based tree algorithm, however, is not sufficiently efficient. The algorithm requires intensive sorting operations in intermediate nodes when performing classification and the computation cost of sorting becomes substantial although dissimilarity computations can be largely pruned. Given an n-level condensation tree with the same depth in all branches and with the nodes evenly distributed, each intermediate node in the tree has the same number of immediate subnodes. With *a recognition margin* p, a constant percentage of subnodes in each node will be followed. Under these conditions, for a classification with L nodes visited, the number of intermediate nodes traversed is estimated as: $\frac{L((1+p)^n-1)}{(1+p)^{n+1}-1}$. For example, given $n = 5$ and $p = 0.2$, there are $0.75 * L$ intermediate nodes traversed, that is, 75 percent of the nodes visited have to perform sorting operation; given $n = 10$ and $p = 0.2$, 80.7 percent of the nodes visited need sorting operation.

In the rest of the paper, we first propose a cluster-based tree algorithm for accelerating k-NN classification without any presuppositions about the metric form and properties of dissimilarity measures, then evaluate its effectiveness in comparison with the exhaustive k-NN and the condensation-based tree algorithm through extensive experiments on the standard NIST and MNIST databases.

2 The Cluster Tree Algorithm

Let $\mathcal{T} = \{Z_i \in \Omega, i = 1, 2, \cdots m\}$ denote a set of training templates. Each template $Z_i \in \mathcal{T}$ has a class label $C(Z_i)$. The objective is to efficiently find the k-nearest neighbors of a test pattern X ($X \in \Omega$) in $\mathcal{T}$ based on a dissimilarity measure $d(\cdot, \cdot)$.

The cluster-based algorithm consists of two phases: tree generation and classification. Different from the existing tree classification methods, this algorithm introduces class-conditional clustering and establishes two *decision levels* for early decision making. **A decision level** in a tree is a level where each node and its subnodes have a unique class label. Thus, at a decision level, the class of an unseen template can be decided using k-nearest neighbor classification.

2.1 Cluster-Based Tree Generation

Given a set $\mathcal{T}$ of m templates, the bottom level of the tree, $\mathcal{B}$, consists of all templates in $\mathcal{T}$. Obviously, $\mathcal{B}$ is a decision level. Another decision level, the **hyperlevel**, is generated through class-conditional clustering over $\mathcal{T}$. In the *hyperlevel*, each node (called **hypernode**) is the cluster center of certain templates in $\mathcal{T}$ with the same class label. $\mathcal{T}^c$ represents a set of cluster centers at a level above the bottom.

For a template $Z \in \mathcal{T}$, its local properties are measured: 1) $\gamma(Z)$, the dissimilarity between Z and its nearest neighbor with a different class label, 2) $\Psi(Z)$, a set of all neighbors which have the same class label as Z and are less than $\gamma(Z)$ distant from Z, and 3) $\ell(Z)$, the size of the set $\Psi(Z)$. Similarly, we define the local properties of a cluster center $Y \in \mathcal{T}^c$ for a given threshold η: 1) $\Psi^c(Y)$, a set of all neighbors which are less than η far from Y, and 2) $\ell^c(Y)$, the size of the set $\Psi^c(Y)$. By the definition, $\Psi^c(Y)$ may contain cluster centers belonging to different pattern classes.

A cluster tree is generated as follows:

Step 0: Initialize the cluster tree to be a single level $\mathcal{B}$ without node.

Step 1: Compute the localities of each template Z in $\mathcal{T}$, i.e., $\gamma(Z)$, $\Psi(Z)$, and $\ell(Z)$. Then, rank all templates in $\mathcal{T}$ in descendant order of $\ell(\cdot)$.

Step 2: Take the template with the biggest $\ell(\cdot)$, Z_1, as a hypernode, and copy all templates of $\Psi(Z_1)$ as nodes at the bottom level of the tree, $\mathcal{B}$. Then, remove all templates in $\Psi(Z_1)$ from $\mathcal{T}$, and set up a link between Z_1 and each pattern of $\Psi(Z_1)$ in $\mathcal{B}$.

Step 3: Repeat *Step 1* and *Step 2* until $\mathcal{T}$ becomes empty. At this point, the cluster tree is configured with a hyperlevel, $\mathcal{H}$, and a bottom level, $\mathcal{B}$.

Step 4: Select a threshold η and cluster all templates in $\mathcal{H}$ so that radius of each cluster is smaller or equal to η. All cluster centers form another level of the cluster tree, $\mathcal{P}$.

Step 5: Increase the threshold η and repeat *Step 4* for all nodes at the level $\mathcal{P}$ until a single node is left in the resulting level.

Generation of the hyperlevel is a process of class-conditional clustering, that is, templates belonging to the same class are grouped together. Building-up of the levels above the hyperlevel is based only on nearness among a set of nodes. Specifically, the clustering procedure in *Step 4* is to iterate *Step 1* and *Step 2* based on *the localities of the cluster centers* (previously defined) at the current level. All cluster centers are actual data points so that the same dissimilarity measure can be used in both the tree generation and the classification phases.

Selection of the threshold η is critical for generating a cluster tree. Obviously, η should be an increasing function of the number of iterations at *Step 4* so that clusters have greater abstraction capacity than their descendants. Let $\eta(i)$ be the threshold for the ith iteration, a simple solution is $\eta(i) = \mu_i - \alpha\frac{\sigma_i}{1+i}$, where α is a constant and μ_i and σ_i represent the mean and the standard deviation of the dissimilarities between the nodes at the current level, respectively.

The hyper level $\mathcal{H}$ and the bottom level $\mathcal{B}$ are "meaningful" since a decision can be made at the two levels, but the levels above the hyperlevel are "meaningless," i.e., the class of a given template cannot be decided at those levels, instead, they act as routes to direct the search downward.

The computation complexity is estimated as follows:. Building up the hyperlevel $\mathcal{H}$ (Step 1 to Step 3) takes $O(\sum_{i=1}^{m} ilgi) \leq O(m^2lgm)$ time in the worst-case. Let n $(n << m)$ be the number of nodes in $\mathcal{H}$, building up the levels above $\mathcal{H}$ will take at most $O(n^2lgn)$ time. Thus, the computation complexity for generating a complete cluster tree is $O(m^2lgm)$, where m is the size of the template set for growing the tree.

2.2 Classification

An unseen template X is classified through adaptively searching the trained cluster tree.

Step 0: Compute dissimilarity between X and each node at the top level of the cluster tree, and choose the ζ nearest nodes as a node set $\mathcal{L}_x$.

Step 1: Compute dissimilarity between X and each subnode linked to the nodes in $\mathcal{L}_x$, and again choose the ζ nearest nodes, which are used to update the node set $\mathcal{L}_x$.

Step 2: Repeat *Step 1* until *the hyperlevel* in the tree. When the searching stops at *the hyperlevel*, $\mathcal{L}_x$ consists of ζ hypernodes.

Step 3: Search $\mathcal{L}_x$ for the hypernodes: $\mathcal{L}_h = \{Y | d(Y,X) \leq \gamma(Y), Y \in \mathcal{L}_x\}$. If all nodes in $\mathcal{L}_h$ have the same class label, then this class is associated to X and the classification process stops; otherwise, go to *Step 4*.

Step 4: Compute the dissimilarity between X and every subnode linked to the nodes in $\mathcal{L}_x$, and choose the k nearest templates. Then, take a majority voting among the k nearest templates to decide the class label of X.

At *the hyperlevel*, the class of a given template X is decided only if all clusters (the elements in $\mathcal{L}_h$) into which X falls have the same class label. Due to this strict condition, the recognition rate in *the hyperlevel* is quite high. The algorithm terminates searching at *the hyperlevel* for most testing templates. Once the algorithm fails in classifying X at *the hyperlevel* (that is, the clusters into which X falls have at least two different class labels.), it further searches *the bottom level*, where a final decision will be made.

The k nearest neighbors of X can be found by slightly modifying the classification procedure. In *Step 3*, if all nodes in $\mathcal{L}_h$ have the same class label, the k nearest templates are chosen from all subnodes (*the bottom level*) linked to the nodes in $\mathcal{L}_h$; otherwise, the k nearest templates are obtained by following *Step 4*.

The bigger the value of ζ, the more dissimilarity computations are carried out with higher accuracy of recognition results. Notice that the cluster tree requires very limited sorting operations, so the cost of side operation is very low.

3 Performance Evaluation

The standard NIST and MNIST handwritten numeral databases are used as benchmarks to compare the proposed algorithm with the exhaustive k-NN and the condensation-based tree algorithm.

3.1 Feature Extraction and Dissimilarity Measure

Gradient-based binary features are extracted from binary handwritten numeral images. Specifically, the binary features for each image consist of 512 bits corresponding to gradient (192 bits), structural (192 bits), and concavity (128 bits) features [18].

Let X and Y be two N-dimensional binary vectors. The dissimilarity between X and Y is given by a generalized Sokal-Michener dissimilarity measure [21], [22]: $D(X,Y) = N - X{\cdot}Y - \beta\overline{X}{\cdot}\overline{Y}$, where $0 \leq \beta \leq 1$ and $\overline{X}$ and $\overline{Y}$ are the complement vectors of X and Y, respectively.

When $\beta = 1$, $D(\cdot,\cdot)$ becomes the metric Manhattan distance, but when $\beta \neq 1$, $D(\cdot,\cdot)$ is nonmetric because it violates reflexitivity. A number of experiments of k-NN classification using $D(\cdot,\cdot)$ show that nonmetric versions of $D(\cdot,\cdot)$ (with β around 0.5) always outperforms the metric one [22]. SM dissimilarity measure has been proven to obey the triangle inequality and possess a special property: In certain subspaces, the sum of *any* two dissimilarities is always bigger than *any* single one [22]. This special property actually disables fast k-NN methods exploiting the triangle inequality like the branch and bound algorithm [9].

In the subsequent experiments, $D(\cdot,\cdot)$ is parameterized by $\beta = 0.5$ and computed using efficient bit-based operations.

3.2 Experimental Settings

For generating cluster trees, the constant α in $\eta(i)$ is set as 2. Each test set is repeatedly classified through a trained cluster tree by tuning ζ.

Since there are multiple parameters with the condensation-based tree algorithm, a number of condensation trees have to be grown in order to find the best one. For all condensation trees, *the tree margin* is set as 0.20 and the maximum number of immediate subnodes allowed for each intermediate node is 80 percent of the one for the root. For each training set, 10 condensation trees are generated by using 10 different maximum numbers ($\{50 \cdot j | j = 1, 2, \ldots, 10\}$) of immediate subnodes allowed at the root. For convenience, we use Condense (*j*) to represent a condensation tree with maximum *j* subnodes allowed for the root. For each condensation tree, the corresponding test set is recognized repeatedly by tuning *the recognition margin*.

For each test, overall recognition accuracy, average recognition time per numeral, and average number of dissimilarity computations per numeral are recorded. Recognition time does not include the time for feature extraction. Classification performance of a tree is also characterized by computation cost versus classification accuracy. Specifically, two relation curves, average number of dissimilarity computations versus classification accuracy (*D-A* curve) and average recognition time versus classification accuracy (*T-A* curve), are sketched for performance comparison.

All experiments in this section were performed on an UltraSPARC workstation with an 800MHz CPU, 512MB RAM, and SunOS 5.8.

3.3 Experiments on NIST Database

NIST handwritten numeral database was collected among census employees. The *nist_hsf0* set with 53,449 handwritten numeral images was taken for growing cluster and condensation trees, and the *nist_hsf1* set with 53,313 handwritten numerals was used for

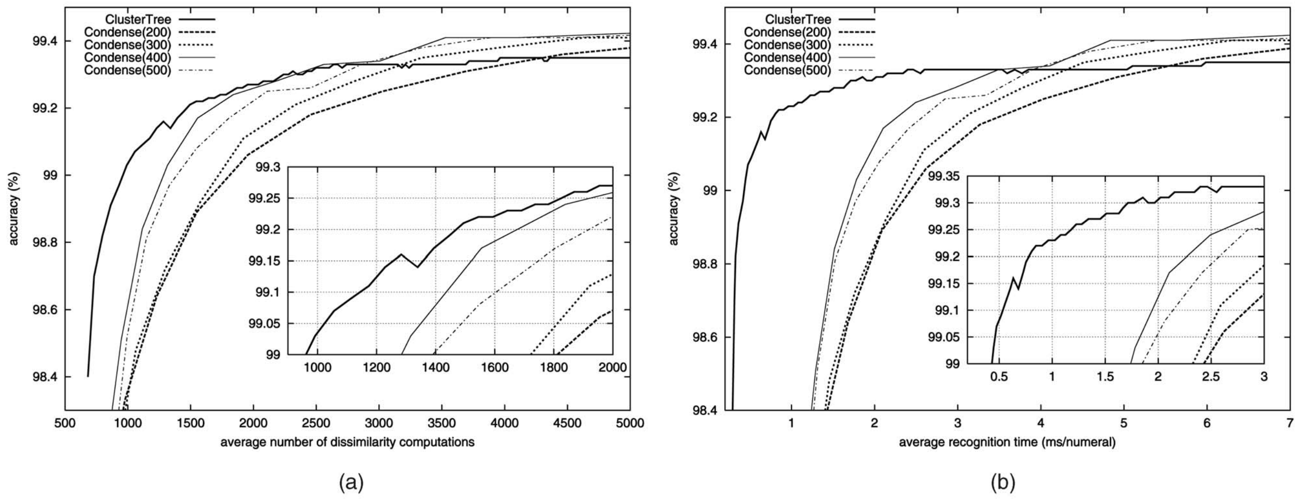

Fig. 1. Performance comparison of five trees: (a) number of dissimilarity computations versus classification accuracy and (b) average recognition time (milliseconds per numeral) versus recognition accuracy. The inside graphs magnify the performance of high accuracy and high speed.

testing. The exhaustive k-NN achieves an accuracy 99.25 percent with an average of 20.68 milliseconds per numeral.

Growing the cluster tree took 45 minutes. The resulting cluster tree includes four levels, with 28, 252, 1,564, and 53,449 nodes from the top to the bottom levels, respectively. Growing a condensation tree is faster than training a cluster tree. Growing Condense (*100*) took about 17 minutes and growing Condense (*400*) required only 12 minutes. Condense (*400*) consistently outperforms all the other condensations trees.

Fig. 1a compares D-A curves with the cluster tree and four condensation trees. Fig. 1b compares T-A curves of the five trees. In our pursuit of a classifier with both high accuracy and high speed, the settings for a classifier, which lead to low accuracy or long recognition time, are not of interest. D-A and T-A curves with regard to the most useful settings for the five trees are enlarged in Fig. 1. For any recognition accuracy varying from 99 percent to 99.28 percent, the cluster tree requires 76 percent to 90 pecent of the dissimilarity computations needed by Condense (*400*) and takes only one-fourth to half of the recognition time used by Condense (*400*). In other words, under the same recognition accuracy, classification by the cluster tree is two to four times faster than by the best condensation tree. A more detailed comparison can be seen in Table 1.

Moreover, with the settings leading to the same accuracy as the exhaustive k-NN classification, the cluster tree requires only 1,826 dissimilarity computations, about one-thirtieth of the computations for the exhaustive search, and uses only 1.179 milliseconds per numeral, 16.5 times faster than the exhaustive search.

3.4 Experiments on MNIST Database

The MNIST handwritten numeral database was created by mixing NIST's Special Database 3 (SD-3) and Special Database 1 (SD-1). As the samples in SD-1 were written by high-school students, the MNIST database is more difficult to recognize than the NIST database. We used the MNIST database provided by LeCun et al. [23]. The training set includes 60,000 samples with a half from SD-3 and another half from SD-1, and the test set contains 5,000 samples from SD-3 and 5,000 samples from SD-1. Without using artificially distorted images for training, the best recognition rate based on the MNIST database is around 99 percent (99.05 percent for LeNet-5, 99 percent for SVM, 98.9 percent for LeNet-4, and k-NN with the tangent distance) [23].

In our experiments, the gray-scale images in the training and the test sets are first binarized by a given threshold (35 for our experiment), then the binary images are used for binary feature extraction. Although more complicated binarization algorithm and skew correction will lead to higher recognition accuracy rate, they are not important for our comparison purpose. The algorithmic parameters for generating and testing cluster and condensation trees based on the MNIST database are the same as those over the NIST database. The exhaustive k-NN classification achieves an accuracy rate 98.24 percent with 23.26 milliseconds per numeral. The resulting cluster tree has four levels with 29, 337, 2,888, and 60,000 nodes, respectively. Notice that here, the number (2,888) of hypernodes almost doubles that (1,564) from the NIST training set, indicating that the MNIST database is much more diversified than the NIST one, thus more difficult to recognize. Out of the 10 condensation trees, Condense (400) still performs the best.

Table 2 compares the cluster tree and the best condensation tree with the settings leading to high speed and high accuracy. The proposed algorithm is still much faster than the condensation tree algorithm. Different from the results based on the NIST database, here, the accuracy rates are lower than the one using the exhaustive k-NN.

With longer recognition time, the proposed fast algorithm can actually approach the accuracy rate of the exhaustive k-NN. For example, this algorithm can correctly classify 98.13 percent samples of the test set with 10.123 milliseconds per image, about a half of the time using the exhaustive k-NN. However, with about the same time the condensation tree can achieve an accuracy rate 98.31 percent, even better than the accuracy rate 98.24 percent using the exhaustive

TABLE 1
Comparison of the Recognition Performance of the Cluster Tree and the Best Condensation Tree (Condense (*400*)) Based on the NIST Database

	Recognition Performance							
	accuracy=99.03%		accuracy=99.17%		accuracy=99.24%		accuracy=99.28%	
	computation	time	computation	time	computation	time	computation	time
Cluster	992	0.447	1394	0.725	1737	1.080	2079	1.511
Condense	1316	1.782	1555	2.104	1839	2.492	2178	2.960

Computation represents the average number of dissimilarity computations and time means the average recognition time in milliseconds per numeral.

TABLE 2
Performance Comparison of the Cluster Tree and the Best Condensation Tree (Condense (*400*)) Based on the MNIST Database

	Recognition Performance							
	accuracy=97.25%		accuracy=97.70%		accuracy=97.86%		accuracy=98.00%	
	computation	time	computation	time	computation	time	computation	time
Cluster	1115	0.506	1919	1.066	2329	1.484	3840	4.010
Condense	1580	1.998	2675	3.415	3509	4.524	4052	5.230

k-NN. Essentially, the proposed cluster tree algorithm improves the search efficiency with a cost of very small accuracy loss while the condensation-based tree algorithm can keep (even increase) accuracy with relatively long search time.

3.5 Analysis

The results from the extensive experiments consistently reveal the superior performance of the proposed fast algorithm. Not only does the algorithm substantially alleviate the computation cost of k-NN search with only minimal accuracy loss, but it also performs classification much faster than the condensation-based tree algorithm (considering settings leading to high speed and high accuracy). Two factors contribute to the efficiency of a cluster tree: a mechanism of early decision making and minimal side-operations for choosing searching paths. Additional experiments also show that most test samples are recognized at *the hyperlevel* with very high reliability, and only a small percentage have to be classified at the bottom level. Classification through a condensation tree suffers from intense sorting operations in intermediate nodes, resulting in much longer recognition time than through a cluster tree.

In both the tree generation and the classification phases, the cluster tree algorithm does not make any presuppositions about the metric form and properties of a dissimilarity measure, thus it is applicable to any k-NN problem with any dissimilarity measure, metric, or nonmetric. Notice that the cluster tree algorithm can be easily tuned to fit any trade off between accuracy and speed by changing the value of ζ. The bigger ζ, the higher accuracy but low recognition speed. When ζ is big enough, a cluster tree will have the same accuracy as the exhaustive k-NN classification. But, if ζ is too small, the branches selected will not be sufficient enough to capture the variation in a testing image and more likely lead to a wrong classification. Obviously, ζ may vary at different levels. Further, ζ may also vary with testing templates by applying the idea of *the recognition margin* in the condensation tree. Optimization of ζ is our future work.

As the proposed algorithm is essentially a lossy classification method, in general, it is not efficient for recognition with accuracy rate very close to or above that of the exhaustive k-NN. We have seen the possibility to improve the cluster-based tree algorithm by incorporating the principles in the condensation-based tree algorithm which is less efficient but lossless.

4 Conclusions

We propose a cluster-based tree algorithm to accelerate k-nearest neighbor classification without any presuppositions about the metric form and properties of dissimilarity measures. A mechanism of early decision making and minimal side-operations for choosing searching paths largely contribute to the efficiency of classification through a cluster tree. The proposed algorithm substantially improves the search efficiency with minimal accuracy loss.

Acknowledgments

This work was supported in part by the US Department of Justice, National Institute of Justice grant 2002-LT-BX-K007.

References

[1] T.M. Cover and P.E. Hart, "Nearest Neighbor Pattern Classification," *IEEE Trans. Information Theory,* vol. 13, pp. 21-27, Jan. 1968.
[2] G.L. Ritter, H.B. Woodruff, S.R. Lowry, and T.L. Isenhour, "An Algorithm for a Selective Nearest Neighbor Decision Rule," *IEEE Trans. Information Theory,* vol. 21, pp. 665-669, Nov. 1975.
[3] C.L. Chang, "Finding Prototypes for Nearest Neighbor Decision Rule," *IEEE Trans. Computers,* vol. 23, no. 11, pp. 1179-1184, Nov. 1974.
[4] P.E. Hart, "Condensed Nearest Neighbor Rule," *IEEE Trans. Information Theory,* vol. 14, pp. 515-516, May 1968.
[5] D.W. Jacobs and D. Weinshall, "Classification with Non-Metric Distances: Image Retrieval and Class Representation," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 22, no. 6, pp. 583-600, June 2000.
[6] A.J. Broder, "Strategies for Efficient Incremental Nearest Neighbor Search," *Pattern Recognition,* vol. 23, nos. 1/2, pp. 171-178, Nov. 1986.
[7] A. Farago, T. Linder, and G. Lugosi, "Fast Nearest-Neighbor Search in Dissimilarity Spaces," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 15, no. 9, pp. 957-962, Sept. 1993.
[8] J.H. Friedman, F. Baskett, and L.J. Shustek, "An Algorithm for Finding Nearest Neighbors," *IEEE Trans. Computers,* vol. 24, no. 10, pp. 1000-1006, Oct. 1975.
[9] K. Fukunaga and P.M. Narendra, "A Branch and Bound Algorithm for Computing k-Nearest Neighbors," *IEEE Trans. Computers,* vol. 24, no. 7, pp. 750-753, July 1975.
[10] B.S. Kim and S.B. Park, "A Fast k Nearest Neighbor Finding Algorithm Based on the Ordered Partition," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 8, no. 6, pp. 761-766, Nov. 1986.
[11] E. Vidal, "An Algorithm for Finding Nearest Neighbors in (Approximately) Constant Average Time," *Pattern Recognition Letters,* vol. 4, no. 3, pp. 145-157, July 1986.
[12] R.L. Brown, "Accelerated Template Matching Using Template Trees Grown by Condensation," *IEEE Trans. Systems, Man, and Cybernetics,* vol. 25, no. 3, pp. 523-528, Mar. 1995.
[13] S.A. Nene and S.K. Nayar, "A Simple Algorithm for Nearest-Neighbor Search in High Dimension," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 19, no. 9, pp. 989-1003, Sept. 1997.
[14] J.L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Comm. ACM,* vol. 18, no. 9, pp. 509-517, Sept. 1975.
[15] M. Donahue, D. Geiger, R. Hummel, and T Liu, "Sparse Representations for Image Decompositions with Occlusions," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* pp. 7-12, 1996.
[16] T. Hastie and R. Tibshirani, "Discriminant Adaptive Nearest-Neighbor Classification," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 18, no. 6, pp. 607-615, June 1996.
[17] J. Puzicha, J. Buhmann, Y. Rubner, and C. Tomasi, "Empirical Evaluation of Dissimilarity Measures for Color and Textures," *Proc. Int'l Conf. Computer Vision,* pp. 1165-1172, 1999.
[18] J.T. Favata and G. Srikantan, "A Multiple Feature/Resolution Approach to Handprinted Character/Digit Recognition," *Proc. Int'l J. Imaging Systems and Technology,* vol. 7, pp. 304-311, 1996.
[19] D. Hunttenlocher, G. Klanderman, and W. Rucklidge, "Comparing Images Using Hausdorff Distance," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 19, no. 1, pp. 1-14, Jan. 1997.
[20] A.K. Jain and D. Zongker, "Representation and Reconstruction of Handwritten Digits Using Deformable Templates," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 19, no. 12, pp. 1386-1391, Dec. 1997.
[21] J.D. Tubbs, "A Note on Binary Template Matching," *Pattern Recognition,* vol. 22, no. 4, pp. 359-365, 1989.
[22] B. Zhang and S.N. Srihari, "Properties of Binary Vector Dissimilarity Measures," *Proc. JCIS Int'l Conf. Computer Vision, Pattern Recognition, and Image Processing,* Sept. 2003.
[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proc. IEEE,* vol. 81, no. 11, pp. 2278-2324, Nov. 1998.