ing to Takiyama's formula, an element with two thresholds can only distinguish 30 of the 32 total dichotomies. In reality, all dichotomies are possible. To see this, note that one of the sets in any given dichotomy must contain two or fewer elements. The patterns in this set can be sandwiched between arbitrarily close parallel hyperplanes. If these hyperplanes are sufficiently close, the remaining points will lie in the outside regions and will be classified into a second category. This shows that all dichotomies are possible. Thus, Theorem 2 is in error for $N = 5$, $n = 2$, $k = 2$, $r = 0$.

We now show that the number of dichotomies is configuration dependent. Consider six points in the plane. If these six points are vertices of a regular hexagon, then it can be shown that only 62 of the 64 total dichotomies can be obtained with a two-threshold element. On the other hand, if five of the points are vertices of a regular pentagon and the sixth is the center of the pentagon, then all 64 dichotomies are possible. As a result, there is no hope of obtaining a distribution-free formula for the separating capacity of a multithreshold element.

The error in Takiyama's analysis occurs in the last step of the induction argument of Theorem 2. It is argued that if one constrains all of the threshold hyperplanes to pass through different points, then the number of dichotomies is the same as if one had a single hyperplane constrained to pass through one point. The latter quantity is known to be configuration free, but the former, unfortunately, is configuration dependent.

A Fast k Nearest Neighbor Finding Algorithm Based on the Ordered Partition

BAEK S. KIM AND SONG B. PARK

Abstract—**We propose a fast nearest neighbor finding algorithm, named tentatively an ordered partition, based on the ordered lists of the training samples of each projection axis. The ordered partition contains two properties, one is ordering—to bound the search region, and the other is partitioning—to reject the unwanted samples without actual distance computations. It is proved that the proposed algorithm can find k nearest neighbors in a constant expected time. Simulations show that the algorithm is rather distribution free, and only 4.6 distance calculations, on the average, were required to find a nearest neighbor among 10 000 samples drawn from a bivariate normal distribution.**

Index Terms—**Density estimation, k nearest neighbors, nonparametric pattern recognition, ordered partition, preprocessing, search tree.**

I. Introduction

The k nearest neighbor search has been proved to be a powerful nonparametric technique for multivariate density estimation and pattern classification [1]–[3]. This technique is very simple in principle, but to find k nearest neighbors is time consuming, particularly for a large number of training samples. Therefore, several strategies have been proposed to reduce the number of distance calculations.

Fischer and Patrick [4] proposed a preprocessing technique that reorders the training samples so that each sample tends to be far away from its predesessors in the ordered list. The nearest neighbor of a test sample is found by the training samples in the listed order. Simulation showed a considerable reduction of the number of distance calculations.

Friedman *et al.* [5] ordered the training samples according to their projection values on a given coordinate axis. Then the search proceeds by examining the training samples in the order of their projection distances from the test sample until the projection distance becomes larger than the full d-dimensional distance to the kth nearest neighbor among the training samples already examined. It was shown that the expected number of distance calculations is $O(n^{1-1/d})$ where n is the number of training samples.

Yunck [8] extended the simple projection algorithm to the cube algorithm, which is based on the d ordered lists of each projection axis. He showed that the expected number of distance calculations is $O(1)$, i.e., independent of n; however, his algorithm still requires $O(n)$ shift operations.

Fukunaga and Narendra [6] partitioned the training samples and used the branch-and-bound search method, resulting in a considerable reduction of the number of examined samples. For partitioning, they used the well-known k-means clustering algorithm.

Friedman *et al.* [7] partitioned the d-dimensional feature space using the k-d tree method in which the storage and the preprocessing time are $O(n)$ and $O(n \log n)$, respectively. In searching, the expected number of examined samples is independent of n, while the time for tree searching is $O(\log n)$.

This correspondence describes a preprocessing technique, to be called the ordered partition, to reduce the computational requirements for finding k nearest neighbors to a test sample among n training samples in a d-dimensional feature space as measured by the Euclidean distance metric. The basic idea of this technique is as follows. First, if the training samples are partitioned by their coordinate values along each axis such that the ordering characteristic is preserved between the partitions, then the number of examined partitions will be independent of n. Second, if we adjust the widths of the partitions such that each partition contains the same number of samples, the performance of the algorithm will be nearly independent of the underlying distribution. The branch-and-bound search method is used for searching.

We describe the algorithm in Section II, and analyze its performance in Section III. The simulation results and conclusions are given in Section IV.

II. Search Procedure

Let $X = \{x_1, x_2, \cdots, x_n\}$ be an ordered set of n training samples in the d-dimensional feature space, and let $x_i = (x_{i1}, x_{i2}, \cdots, x_{id})$ be the ith training sample. The problem is then stated as follows: given a test sample $x_q = (x_{q1}, x_{q2}, \cdots, x_{qd})$, find the k nearest samples to x_q among the members of X as quickly as possible.

The procedure is divided into two steps. The first, the preprocessing step, involves the ordering and partitioning of n training samples independently of x_q in order to construct the search tree. The second, the searching step, finds k nearest samples of x_q among the members of X based on the search tree constructed in the first step.

Preprocessing: First, select $p(i)$, the number of partitions of the ith axis, for $i = 1, 2, \cdots, d-1$. As will become clear later, $p(i)$ equals the number of son nodes expanded from a node in the $(i-1)$th level.

The search tree is then constructed by the following recursive procedure.

Manuscript received August 3, 1985; revised March 22, 1986. Recommended for acceptance by J. Kittler.

B. S. Kim is with the Department of Medical Information Management, College of Medicine, Hanyang University, 17 Haengdang-dong, Seongdong-ku, Seoul 133, Korea.

S. B. Park is with the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, P.O. Box 150, Chongyangni, Seoul 131, Korea.

IEEE Log Number 8609232.

0162-8828/86/1100-0761$01.00 © 1986 IEEE

Authorized licensed use limited to: GOOGLE. Downloaded on December 11, 2009 at 13:56 from IEEE Xplore. Restrictions apply.

```
procedure MAKE TREE(file, l) ;
begin
   let data be the lth coordinate values of the members in file ;
   order the data ;
   partition the ordered data into p(l) groups ;
   for each group do
      begin
         let subfile be the samples corresponding to the
           group ;
         if l = d then MAKE TERMINAL(subfile, d)
            else begin
               create a node for the subfile ;
               MAKE TREE(subfile, l + 1);
               end
         end
   end ; { of MAKE TREE }
```

MAKE TREE(X, 1) produces the search tree with the number of nodes given by

$$\sum_{i=1}^{d-1} \prod_{j=1}^{i} p(j) + n + 1. \tag{1}$$

Each term corresponds to the number of nonterminal nodes except the root, the number of terminal nodes, and the root, respectively. The procedure MAKE TERMINAL generates the terminal nodes for each sample in the file.

A node consists of a set of samples and four pointers pointing to other nodes, namely, son, parent, left sibling, and right sibling nodes. (The sibling nodes mean the nodes having the identical parent node.) In fact, two boundary values are sufficient to represent the set of samples in a node. The terminal node contains only one sample, and it has an index pointer pointing to the sample.

After completion of the preprecessing step, the search tree is stored together with the set of training samples X. For example, let us consider the case that $d = 3$, $n = 12$, $p(1) = p(2) = 3$, and $X = \{(1, 2, 5), (3, 8, 7), (9, 10, 8), (12, 9, 2), (8, 7, 20), (6, 6, 23), (0, 3, 27), (2, 13, 9), (11, 11, 15), (14, 17, 13), (7, 4, 12), (10, 12, 3)\}$. Fig. 1 shows the generated search tree. The numbers in parentheses denote the low and high bounds, respectively, representing the training samples in a node; for example, the node b contains the samples $-\infty < x_1 \leq 3$ and $2 < x_2 \leq 8$, i.e., the second and seventh samples. At the terminal node, the first number in the parentheses is the dth coordinate value in the node and the second is the index of the corresponding training sample. Fig. 2 depicts partition of the feature space by the nodes in this case.

Searching: As can be seen from the above example, a nonterminal node at the level l corresponds to a subspace

$$R_l = \{x = (x_1, x_2, \cdots, x_d) / a_i < x_i \leq b_i, \quad \text{for } i = 1, 2, \cdots, l\} \tag{2}$$

where a_i and b_i are the low and high bounds of the parent node at the level i, respectively. For the nodes at the level d, i.e., the terminal nodes, R becomes

$$R_d = \{a_i < x_i \leq b_i, \quad \text{for } i = 1, 2, \cdots, d-1 \quad \text{and} \quad x_d = a_d\}. \tag{3}$$

The k nearest neighbor of a test sample x_q can then be obtained by applying the following recursive procedure.

```
procedure SEARCH(node, l) ;
begin
   if l = d
      then TERMINAL SEARCH(node)
      else begin
       dist ← NODE DISTANCE(node) ;
       s ← FIRST SON(node) ;
       while CANDIDATE(s, dist, l + 1) do
       begin SEARCH(s, l + 1) ;
       s ← NEXT NODE(s)
       end
       end
end ;    { of SEARCH }
```

The function NODE DISTANCE returns the squared distance from the test sample to the node at the level l, which is defined recursively as

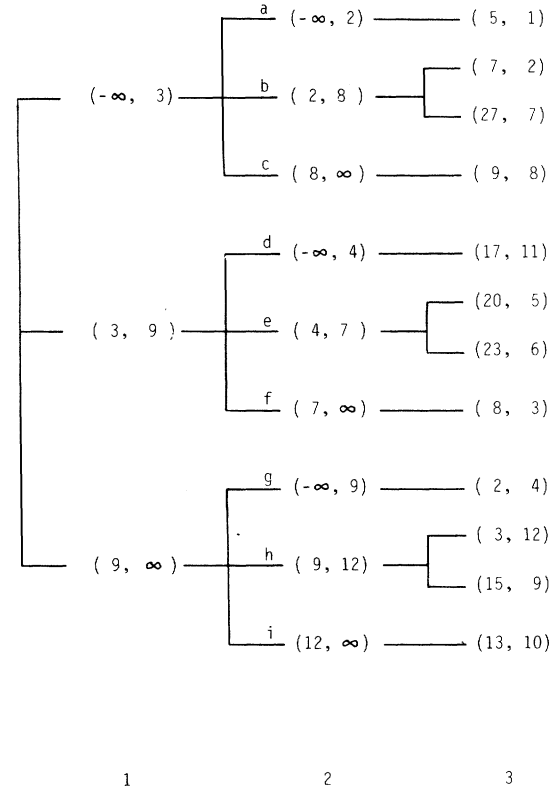

Fig. 1. An example of a search tree.

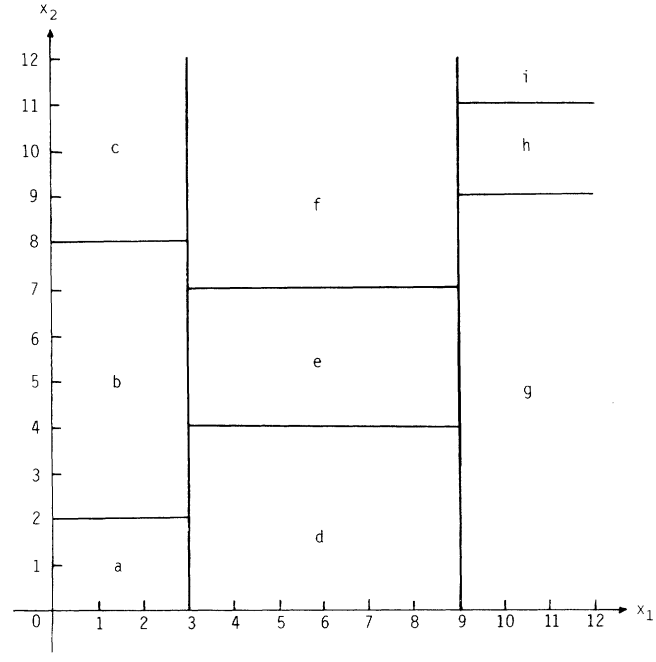

Fig. 2. An illustration of the feature space partitioned by a search tree.

Authorized licensed use limited to: GOOGLE. Downloaded on December 11, 2009 at 13:56 from IEEE Xplore. Restrictions apply.

$$r_l^2 = \begin{cases} r_{l-1}^2 + \min\,[(x_l - a_l)^2, (x_l - b_l)^2], & \text{for } l \neq d \\ r_{d-1}^2 + (x_d - a_d)^2, & \text{for } l = d \end{cases} \tag{4}$$

with $r_0 = 0$.

Now, let d_k^2 be the squared distance from the test sample to the kth nearest training sample up to the current searching stage. d_k^2 is updated by the procedure NODE DISTANCE when the candidate terminal node is visited. Any node whose distance is greater than or equal to d_k^2 need not be searched; this condition is tested by the Boolean function CANDIDATE.

The function FIRST SON returns a node which contains the $(l + 1)$th coordinate value of the test sample. The search proceeds to the next closest node to the $(l + 1)$th coordinate value of the test sample node by the function NEXT NODE.

In brief, the procedure searches from the closest node, and if a node is not a candidate of the k nearest neighbors, then the rest of the sibling nodes will not be searched. The k nearest neighbor distances are updated when the candidate terminal node is visited. The search terminates when a node at level 1 is not the candidate of the k nearest neighbors. As an example, consider the case that $x_q = (10, 10, 10)$ and $k = 2$ with the search tree given in Fig. 1. The algorithm searches for training samples whose indexes are 9, 12, 10, and 3 in this order, and finds two nearest samples, 3 and 9.

III. Performance Analysis

For simplicity, assume that n training samples are drawn from the uniform distribution in a d-dimensional unit hypercube. Consider an arbitrary region R containing the test sample x_q and $m(>k)$ training samples. The volume of R is denoted by v_r. Also, define $S(\subset R)$ as the region containing exactly k nearest neighbors of x_q. The probability content u_s in S is a random variable distributed according to a beta distribution:

$$p(u_s) = \frac{m!}{(k-1)!\,(m-k)!}\,u_s^{k-1}(1-u_s)^{m-k}, \qquad 0 < u_s < 1. \tag{5}$$

The expected value and the variance of u_k are

$$E[u_s] = \frac{k}{m+1} \tag{6}$$

and

$$\text{Var}\,[u_s] = \frac{k(m-k+1)}{(m+1)^2\,(m+2)}, \tag{7}$$

respectively.

Since the volume v_s of S is a random variable

$$v_s = v_r u_s, \tag{8}$$

the expected value of v_s is

$$E[v_s] = v_r \frac{k}{m+1} \approx v_r \frac{k}{nv_r + 1} \approx \frac{k}{n} \tag{9}$$

if n is sufficiently large and $nv_r \gg 1$.

We can now use (9) to describe the upper bound of the expected number of the samples searched by the proposed algorithm. Consider the sibling nodes at the ith level. The proposed algorithm first searches the node, which contains x_{qi}, by the function FIRST SON. The next closest son to x_{qi} is then searched. This procedure is continued until a node distance exceeds the distance from x_q to the kth nearest neighbor among the training samples already searched. The distance therefore determines the number of son nodes of a node in searching. Since the distance is reduced as search proceeds, the search region K can be viewed as a hyperellipse centered on x_q if n is unlimitedly large. Let h_i be the projection distance from the center to a vertex at the ith axis, i.e., the ith level. The search region for the first level is the entire feature space. The expected value of h_1 is bounded by [5]

$$E[h_1] < C_d E^{1/d}[v_s] \approx C_d \left[\frac{k}{n}\right]^{1/d} \tag{10}$$

where

$$C_d = \left[\frac{d\Gamma(d/2)}{2\pi^{d/2}}\right]^{1/d} < \frac{\sqrt{d}}{2}. \tag{11}$$

The inequality in (11) comes from the fact that the volume of the d-dimensional hypershare is larger than that of the inner hyperrectangle touching the hypersphere.

Let us denote w_i as the width, the difference of the high and low bound, of the node which contains x_q at the ith level for $i = 1, 2, \cdots, d-1$. The expected value of w_i is $1/p(i)$ if the samples were drawn from the uniform distribution in a unit hypercube. The jth coordinate of the search region at the ith level is bounded by w_j for $j = 1, 2, \cdots, i-1$. Therefore,

$$v_s \approx A h_i^{d-i+1} \prod_{j=1}^{i-1} w_j, \quad \text{if } w_1, w_2, \cdots, w_{i-1} \text{ are small.} \tag{12}$$

Here, A is a constant defined by

$$A = \frac{2\pi^{(d-i+1)/2}}{(d-i+1)\,\Gamma[(d-i+1)/2]}. \tag{13}$$

The expected value of h_i is bounded by

$$\begin{aligned} E[h_i] &\lesssim B\left[E[v_s] \Big/ \prod_{j=1}^{i-1} w_j\right]^{1/(d-i+1)} \\ &\approx B\left[\frac{k}{n}\right]^{1/(d-i+1)} \left[\prod_{j=1}^{i-1} w_j\right]^{-1/(d-i+1)}, \qquad \text{for } i > 1 \end{aligned} \tag{14}$$

where

$$B = A^{-1/(d-i+1)} = C_{d-i+1} < \frac{\sqrt{d-i+1}}{2}. \tag{15}$$

Since an axis i is quantized by an interval w_i, the maximum distance of the search region on the ith axis is $2h_i + w_i$. Therefore, the expected value of the volume v_k of the search region K becomes

$$\begin{aligned} E[v_k] &< \prod_{i=1}^{d} (2E[h_i] + w_i) \\ &\lesssim \left[2C_d\left(\frac{k}{n}\right)^{1/d} + w_1\right] \prod_{i=2}^{d} \left[2C_{d-i+1}\left(\frac{k}{n}\right)^{1/(d-i+1)} \cdot \left(\prod_{j=1}^{i-1} w_j\right)^{-1/(d-i+1)} + w_i\right]. \end{aligned} \tag{16}$$

Let $w_1 = w_2 = \cdots = w_{d-1} = w = n^{-1/d}$. Then (16) becomes

$$\begin{aligned} E[v_k] &< (\sqrt{d}\,k^{1/d} + 1)\,w \\ &\quad \cdot \prod_{i=2}^{d} [(\sqrt{d-i+1}\,k^{1/(d-i+1)} + 1)w]. \end{aligned} \tag{17}$$

The above assumption is true when n is very large and if we set $p(1) = p(2) = \cdots = p(d-1) = n^{1/d}$.

Since the training samples are assumed to be drawn from the uniform distribution, the expected number of training samples in K, i.e., the number of terminal nodes to be searched, is

$$\begin{aligned} E[n_k] &= nE[v_k] \\ &< \prod_{i=1}^{d} (\sqrt{i}\,k^{1/i} + 1) \end{aligned} \tag{18}$$

which is independent of the number of training samples n.

Equation (18) states that the expected number of terminal nodes to be searched is independent of the number of the training samples. However, the proposed algorithm requires us to search not only the terminal nodes, but also the nonterminal nodes. Note that h_i is the maximum projection distance from x_q to the point where the search terminates for the nodes at the ith level. Therefore, the

Authorized licensed use limited to: GOOGLE. Downloaded on December 11, 2009 at 13:56 from IEEE Xplore. Restrictions apply.

number of sibling nodes to be searched at the ith level is less than $2h_i/w_i + 1$. This leads to the fact that the maximum number of nodes to be searched at the ith level becomes

$$\prod_{j=1}^{i} (2h_j/w_j + 1). \tag{19}$$

The number of nonterminal nodes n_t to be searched is obtained based on the previous assumption: $w_1 = w_2 = \cdots = w_{d-1} = w = n^{-1/d}$, that is,

$$n_t < \sum_{i=1}^{d-1} \prod_{j=1}^{i} (2h_j/w + 1). \tag{20}$$

Assuming that h_i's are independent, the expected value of n_t becomes

$$\begin{aligned} E[n_t] &< \sum_{i=1}^{d-1} \prod_{j=1}^{i} (2E[h_j/w] + 1) \\ &\leq \sum_{i=1}^{d-1} \prod_{j=1}^{i} \left[2C_{d-j+1} \left(\frac{k}{n}\right)^{1/(d-j+1)} w^{-d/(d-j+1)} + 1 \right] \\ &< \sum_{i=1}^{d-1} \prod_{j=d-i+1}^{d} [\sqrt{j}\, k^{1/j} + 1], \end{aligned} \tag{21}$$

which is also independent of the number of the training samples n.

The computational time required for the search of a nonterminal node at the ith level is mainly determined by two factors: the test whether a node can be a candidate of the search and the process of finding a son node which contains the $(i + 1)$th coordinate value of the test sample (the function FIRST SON in the procedure SEARCH). The former is not a function of n and the latter can be done in a constant time if a hash lookup is used. This, along with (18) and (21), establishes the fact that the proposed algorithm finds k nearest neighbors of a test sample in a constant expected time when the number of training samples is very large.

IV. Simulation Results and Conclusions

Simulations were performed with $p(1) = p(2) = \cdots = p(d - 1) = n^{1/d}$. Various numbers of training samples were drawn from the uncorrelated uniform distributions over [0, 1]. Another set of 1000 test samples was also drawn from the same distribution, and the results were averaged over these 1000 test samples.

Fig. 3 shows the average number of terminal nodes $E[n_k]$ and nonterminal nodes $E[n_t]$ required to find a nearest neighbor ($k = 1$) in the bivariate case ($d = 2$), and it demonstrates that both are nearly independent of n. The theoretical upper bounds in this case are $E[n_k] < 2(\sqrt{2} + 1)$ and $E[n_t] < \sqrt{2} + 1$, that is, less than 7.2 nodes, on the average, are sufficient to be searched to find a nearest neighbor regardless of the training sample size.

Fig. 4 is the same as Fig. 3, except that $k = 3$ and $d = 4$. Both n_k and n_t by the simulation show a slight increase with n. The unexpected large value of n_t at $n = 1000$ is due to the fact that we set $p(1) = p(2) = p(3) = 6$, which are greater than $(1000)^{1/4}$.

Fig. 5 shows the dependency on increasing k for the bivariate case with $n = 10\,000$. Note in this case that the upper bounds for $E[n_k]$ and $E[n_t]$, respectively, are given from (18) and (21) as

$$E[n_k] < (k + 1)(\sqrt{2k} + 1) \tag{22}$$

and

$$E[n_t] < (\sqrt{2k} + 1). \tag{23}$$

As can be seen from the figure, these upper bounds are somewhat overestimated. This is due to the fact that we postulated the search region as a hyperrectangle in driving the upper bounds where the true search region is viewed as a hyperellipse. The tighter upper bounds are found in [9] for the bivariate case.

Fig. 6 shows how $E[n_k]$ and $E[n_t]$ vary with increasing dimensionality ($k = 1$ and $n = 10\,000$). For the eight-dimensional case,

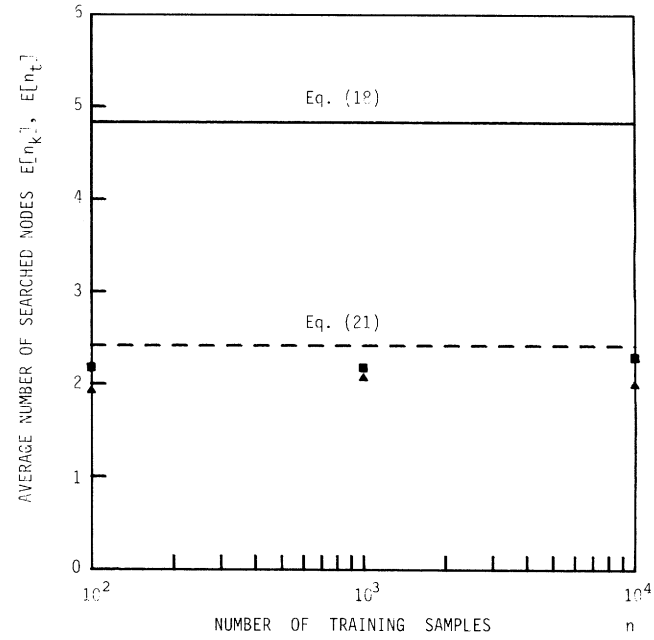

Fig. 3. The number of terminal nodes(square) and nonterminal nodes(triangle) searched to find a nearest neighbor from the samples drawn from the bivariate uniform distribution as a function of n. Solid and dashed lines respectively represent the theoretical upper bounds of them.

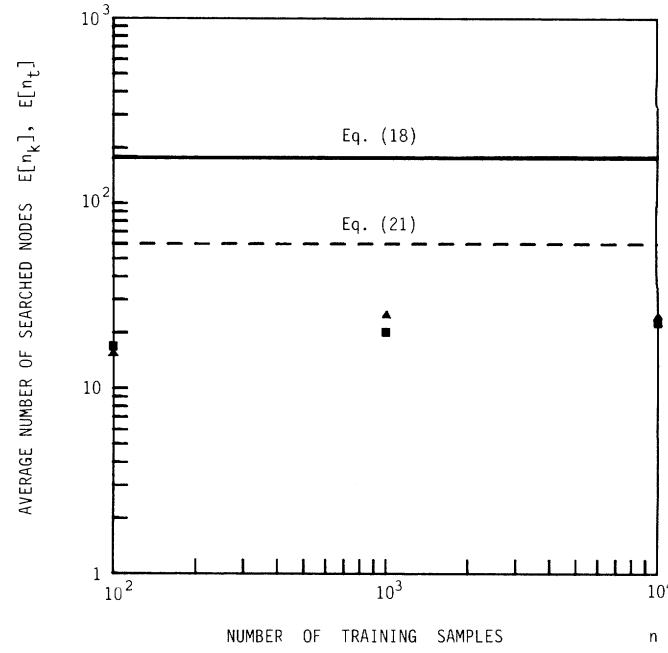

Fig. 4. The number of terminal nodes(square) and nonterminal nodes(triangle) with increasing n for four-dimensional uniform distribution case. The number of nearest neighbors k is set to three.

the number of partitions was set to $p(1) = p(2) = p(3) = 3$ and $p(4) = p(5) = p(6) = p(7) = 4$.

In order to show that the proposed algorithm is relatively distribution free, the same simulations are those given above were performed with the samples drawn from the normal distribution with zero-mean vector and identity covariance matrix. A set of another 1000 test samples was also drawn independently from the same distributions as those of the training samples.

Figs. 7 and 8, respectively, show the same simulation results as those of Figs. 5 and 6 for the samples drawn from the normal dis-

Authorized licensed use limited to: GOOGLE. Downloaded on December 11, 2009 at 13:56 from IEEE Xplore. Restrictions apply.

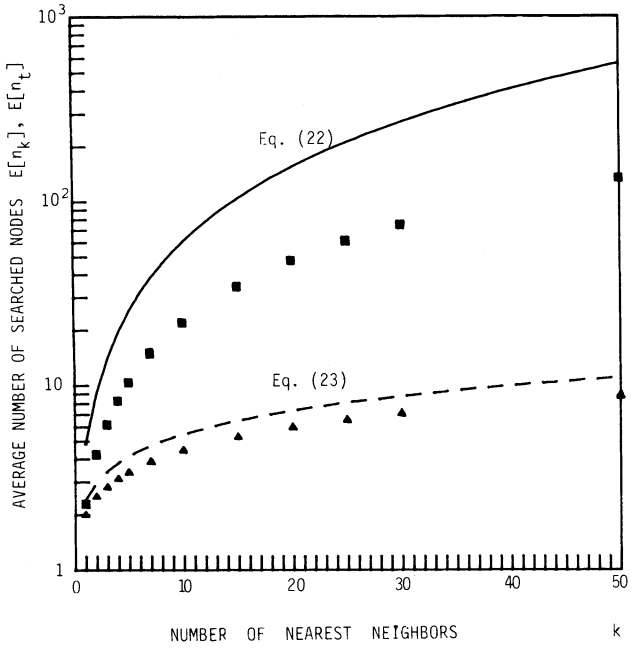

Fig. 5. The number of terminal nodes(square) and nonterminal nodes(triangle) with increasing k among 10 000 samples drawn from a bivariate uniform distribution.

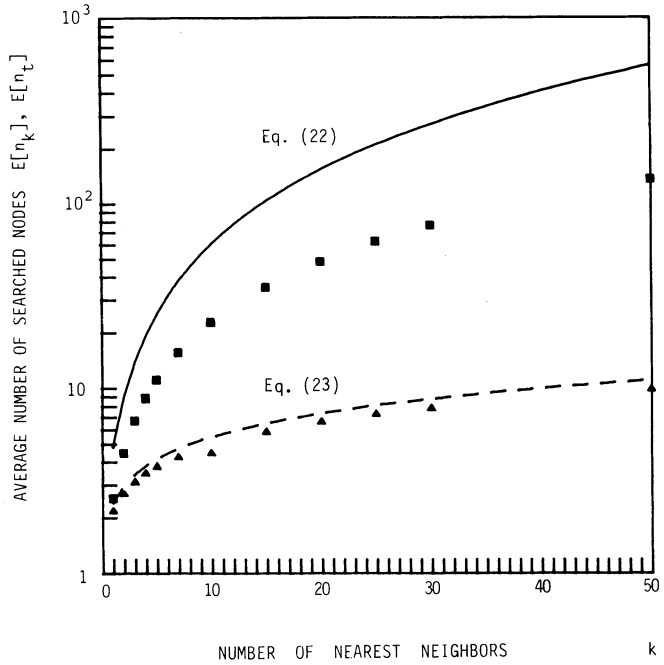

Fig. 7. The number of terminal nodes(square) and nonterminal nodes(triangle) with increasing k among 10 000 samples drawn from a bivariate normal distribution.

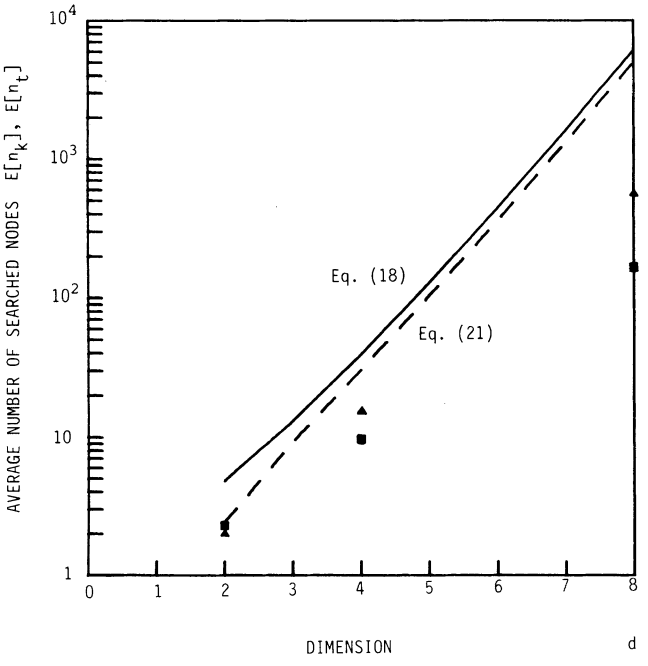

Fig. 6. The number of terminal nodes(square) and nonterminal nodes(triangle) with increasing d among 10 000 samples drawn from the uniform distributions. k is set to one.

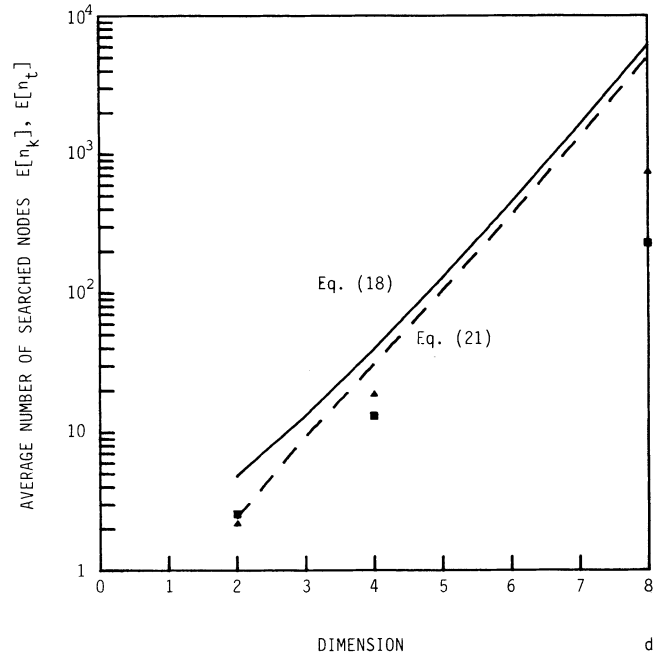

Fig. 8. The number of terminal nodes(square) and nonterminal nodes(triangle) with increasing d among 10 000 samples drawn from the normal distributions. k is set to one.

tribution. Both $E[n_k]$ and $E[n_t]$ show a slight increase compared to the case of the uniform distribution, yet they are still smaller than (18) and (21).

In conclusion, we proposed an algorithm, called the ordered partition, for finding k nearest neighbors of a test sample in a constant expected time. The algorithm is based on two properties: ordering—to bound the search region, and partitioning—to reject the training samples which cannot be the candidate of the solution. It is proved that the upper bounds of both the number of terminal nodes and the number of nonterminal nodes searched to find the solution are independent of the number of training samples when it is very large. Simulations showed that the ordered partition is rather distribution free, and only 4.6 distance calculations, on the average, were required to find a nearest neighbor of a test sample among 10 000 training samples drawn from a bivariate normal distribution.

Acknowledgment

The authors would like to express their thanks to Dr. U. R. Ko at Hanyang University and Dr. K. J. Lee at Choongnam University for their helpful comments.

Authorized licensed use limited to: GOOGLE. Downloaded on December 11, 2009 at 13:56 from IEEE Xplore. Restrictions apply.

REFERENCES

[1] D. O. Loftsgaarden and C. P. Quesenberry, "A nonparametric density estimation," *Ann. Math. Statist.*, vol. 36, pp. 1049–1051, 1965.
[2] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 21–27, Jan. 1967.
[3] T. J. Wagner, "Convergence of the nearest neighbor rule," *IEEE Trans. Inform. Theory*, vol. IT-17, pp. 566–571, Sept. 1971.
[4] F. P. Fischer and E. A. Patrick, "A preprocessing algorithm for nearest neighbor decision rules," in *Proc. 1970 NEC*, pp. 481–485.
[5] J. H. Friedman, F. Baskett, and L. J. Shustek, "An algorithm for finding nearest neighbor," *IEEE Trans. Comput.*, vol. C-24, pp. 1000–1006, Oct. 1975.
[6] K. Fukunaga and P. M. Narendra, "A branch and bound algorithm for computing k-nearest neighbors," *IEEE Trans. Comput.*, vol. C-24, pp. 750–753, July 1975.
[7] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Software*, vol. 3, pp. 209–226, Sept. 1977.
[8] T. P. Yunck, "A technique to identify nearest neighbors," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-6, pp. 678–683, Oct. 1976.
[9] B. S. Kim, "Feature space partitioning techniques in nonparametric pattern recognition," Ph.D. dissertation, Dep. Elec. Eng., Korea Adv. Inst. Sci. Technol., Seoul, 1985.

Authorized licensed use limited to: GOOGLE. Downloaded on December 11, 2009 at 13:56 from IEEE Xplore. Restrictions apply.