HOKKAIDO UNIVERSITY

Title	Probably correct k-nearest neighbor search in high dimensions
Author(s)	Toyama, Jun; Kudo, Mineichi; Imai, Hideyuki
Citation	Pattern Recognition, 43(4), 1361-1372 https://doi.org/10.1016/j.patcog.2009.09.026
Issue Date	2010-04
Doc URL	http://hdl.handle.net/2115/42786
Type	article (author version)
File Information	PR43-4_1361-1372.pdf

Instructions for use

Hokkaido University Collection of Scholarly and Academic Papers : HUSCAP

Probably Correct k-Nearest Neighbor Search in High Dimensions

Jun Toyama Mineichi Kudo * Hideyuki Imai

Division of Computer Science

Graduate School of Information Science and Technology

Hokkaido University, Sapporo 060-0814, JAPAN

Abstract

A novel approach for k-nearest neighbor (k-NN) searching with Euclidean metric is described. It is well known that many sophisticated algorithms cannot beat the brute-force algorithm when the dimensionality is high. In this study, a probably correct approach, in which the correct set of k-nearest neighbors is obtained in high probability, is proposed for greatly reducing the searching time. We exploit the marginal distribution of the kth nearest neighbors in low dimensions, which is estimated from the stored data (an empirical percentile approach). We analyze the basic nature of the marginal distribution and show the advantage of the implemented algorithm, which is a probabilistic variant of the partial distance searching. Its query time is sublinear in data size n, that is, $O(mn\delta)$ with $\delta = o(1)$ in n and $\delta \leq 1$, for any fixed dimension m.

Key words: Pattern recognition, The k-nearest neighbor method, Probably correct algorithm, PAC framework

* Corresponding author. Tel:+81-11-706-6852, Fax:+81-11-706-7393
Email address: mine@main.ist.hokudai.ac.jp (Mineichi Kudo).

1 Introduction

The k-nearest neighbor (k-NN) method [1] is widely used in pattern recognition. This method is useful both for estimation of densities and for classification. As a classifier, this algorithm involves the following three steps: (1) calculating the distances between a query sample and all training samples, (2) choosing the k nearest training samples to the query sample, and (3) assigning a class label by applying the majority rule to the k nearest samples. A major problem, however, is that this method requires a large amount of computation to calculate the distances of a given query sample to all training samples.

Many algorithms, for example, see [2–8], have been proposed for reducing the computational cost in the search phase. These algorithms are divided into the following four categories: (a) exhaustive searching, (b) hashing and indexing, (c) static space partitioning, and (d) dynamic space partitioning. Hashing and indexing algorithms are the fastest search algorithms and run in constant time (for example, see [2]). However, they require storage that grows exponentially in dimensionality, m. To cope with this difficulty, geometric hashing [9,10] using a mapping into cells in a low-dimensional space has been proposed, but it still suffers from an exponential increase of the number of cells to be examined.

Perhaps the most widely used algorithms belong to category (c) or (d). Typical algorithms are a branch and bound algorithm [4], k-d tree [6,11,12], $R(R^+)$-tree [13,14] and X-tree [15]. They have also been improved in several ways [16–18]. There is a trade-off between time complexity in the search phase and space complexity for keeping the rules needed for searching; faster algorithms require larger storage in general. For n training samples with m features, many of the

above algorithms require an exponential storage or an exponential search time in m. A serious problem of high dimensionality therefore arises.

As a means for coping with such a *curse of dimensionality*, principal components analysis (PCA) is sometimes used for reducing the dimensionality. Such a technique often performs since some features are correlated in high dimensional problems. If a small number of efficient principal components can be found, searching time can be greatly reduced. Obviously, the partial distance strategy [19] performs well with PCA. It terminates the distance calculation if the partial distance calculated so far exceeds the distance of the currently obtained kth nearest neighbor. Such a trial is seen in [20].

The most promising approach to gain efficiency even in high dimensional problems is to introduce some kind of tolerance into the problem. There are two kinds of tolerance. The first one is "approximation." With this kind of tolerance, we are satisfied with suboptimal k nearest neighbors in the sense that the solution has a distance less than $(1 + \eta)$ times the distance to the true kth nearest neighbor. Such a trial is called *approximate nearest neighbors* (ANN framework) [18,21]. The ANN algorithm in [21] requires $O(mn \log n)$ time and $O(mn)$ space for preprocessing and runs in $O(c_{m,\eta} \log n)$ with $c_{m,\eta} \leq m\lceil 1+6m/\eta \rceil^m$. Note that the coefficient grows exponentially in m in the worst case. Kleinberg [22] also proposed an algorithm that has a near-linear storage and query time in m when the approximate nearest neighbor is the goal to attain.

The other kind of tolerance is "probabilistic correctness." With this kind of tolerance, the complete correctness of a solution is not guaranteed, but the error probability is upper-bounded. In this case, a *confidence (error)* parameter ϵ is introduced to upper-bound the error probability of the algorithm

missing the true k nearest neighbors. Such a trial is called *probably correct nearest neighbors* (PCNN framework). The goal is to find $\hat{X}_{kNN}$ such that $P(X_{kNN} \neq \hat{X}_{kNN}) < \epsilon$. That is, in high probability, the correct nearest neighbors are found. A study in this line is shown in [23]. Both kinds of tolerance can be considered simultaneously. Then we come to *probably and approximately correct nearest neighbors* (PACNN framework). The goal is to find $\hat{X}_{kNN}$ such that $P\left(D(X, \hat{X}_{kNN}) \leq (1+\eta)D(X, X_{kNN})\right) \geq 1 - \epsilon$ for an approximation parameter $\eta \geq 0$ and for a confidence parameter $\epsilon < 1$. Such a trial is introduced in [24] with an index-based search. The algorithm stops the search once the found points are approximate k nearest neighbors in high probability. Unfortunately, the algorithm in [24] is only applicable for $k = 1$, that is, the nearest neighbor searching. Probably the fastest algorithms for PACNN are locality-sensitive hashing algorithms [25–27]. They adopt multiple sets of hashing functions randomly chosen for narrowing the search area and run in a sublinear order of n. The algorithm in L_2 distance achieves the query time of $O(mn^{1/(1+\eta)^2+o(1)})$ and space $O(mn + n^{1+1/(1+\eta)^2+o(1)})$.

Naturally, ANN and PCNN are two special but stronger cases of PACNN. In this study, PCNN framework is considered with Euclidean metric, because it is very useful for several applications. In PCNN framework, except for a few cases, the correct answer is necessarily obtained. In addition, as will be shown in the proposed algorithm, such exceptions occur only for query samples that are located at the tail of the underlying distribution. This is a good property of PCNN framework because queries are satisfactorily responded if they have very close nearest neighbors. Of course, to suppress the amount of exceptions, only a small value of ϵ is allowed, say, $\epsilon = 0.1\%$ or 1%. At the expense of ϵ error in the correctness, we can reduce the searching time at some rate δ. There are some ANN algorithms that run in $O(\log n)$ in theory. In addition,

the locality-sensitive hashing algorithms theoretically run in a sublinear order in n. While our algorithm runs in $O(n\delta)$ with $\delta \leq 1$, but we will show $\delta = o(1)$ in n. That is, the algorithm is also sublinear in n.

Any PACNN algorithm can be used as a PCNN algorithm with $\eta = 0$. Also we are interested in small ϵ only. However, they affect the order estimate. In [24], the authors describe that their PACNN 1-NN algorithm is not suitable for large data with small ϵ and η (the expected number of distance calculation is $O(n\epsilon^{-1}(1+\eta)^{-m})$ in a uniform distribution with L_∞ norm). For this reason, this algorithm in PCNN framework with $\eta = 0$ seems to perform poorly for small ϵ. In the locality-sensitive hashing algorithms, the theoretical order shows that application to PCNN framework with $\eta = 0$ makes the query time linear in n. In our algorithm, however, there is no such a factor strongly affected by ϵ (always $\eta = 0$). The effectiveness mainly depends on only the nature of the dataset, so that easier problems are solved much faster in the searching. Indeed, in some datasets, the rate δ of full-distance calculation will be shown to be very close to zero even for sufficiently small ϵ. One more attractive point is that we can know the degree of time reduction in advance with sufficient accuracy. After extracting several statistics of a given dataset and before the searching phase, the value of δ for a specified value of ϵ is shown for helping the user's decision.

The actual searching time is also affected by the other factors such as the data structure and the memory access way. Typically several additional calculations are required in the search processing other than the pairwise distance calculation. For example, the algorithm in [21] has a coefficient growing exponentially in m. Therefore, we need to compare algorithms in some concrete large-scale problems.

2 Key idea

Our key idea is illustrated in Fig. 1. Simply speaking, our algorithm relies on the fact that "a very close pair in the original m dimensions is also close in a few l dimensions in high probability" ($m = 2$ and $l = 1$ in this figure). This is easily understood by supposing that sufficiently many samples are given so that any query point and its nearest neighbor are close enough. In Fig. 1, a Gaussian with mean $\mathbf{0}$, variance 1.0 and covariance 0.8 is shown. It shows that the nearest neighbor is between the two lines surrounding the query point (two points in the figure) with high probability. Therefore, it suffices to probe the points in this range. The first principal axis is used for the two lines and the width is determined from the empirical marginal distribution to make the probability higher than 99.9%. Note that the exception will occur in the area where the nearest neighbor is relatively far from the query point (point $(2.0, 2.0)$ in the figure). Also note that the surrounding region becomes narrower as the sample number n increases, so that this algorithm runs in a sublinear order of n. In the following, we describe this strategy formally for large m and for small but more than one l. The point is that l is fairly less than m, so that l can be regarded as a constant.

For simplicity, we consider the case of $k = 1$, that is, the nearest neighbor search. Indeed, when the dimensionality m is large, the following holds for any small k, as will be shown later. We firstly notice that the ranking of points changes if some coordinates are dropped and that 1-NN changes as well. Therefore, for probabilistic evaluation, we have to analyze how the following two probabilities change with a threshold θ_l: $P(D_l^2(X, X_{1NN}) < \theta_l)$ of the squared distance between a point X and its nearest neighbor X_{1NN} and $P(D_l^2(X, Y) < \theta_l)$ of the squared distance between X and any other point

Y. Here, $D_l^2(X,Y)$ is the squared Euclidean distance measured in the first l dimensions, that is, $D_l^2(X,Y) = \sum_{j=1}^{l}(X_j - Y_j)^2$ for $X = (X_1, X_2, \ldots, X_m)$ and $Y = (Y_1, Y_2, \ldots, Y_m)$. When the suffix is omitted, it means D_m^2.

When we adopt $\theta_l = D_m^2(X, \hat{X}_{1NN})$ regardless of the value of l, where $\hat{X}_{1NN}$ is the current candidate of the nearest neighbor in the middle of the search, then we come to the *partial distance strategy* (PDS) algorithm. In this case, in the stored sample set, $P(D_l^2(X, X_{1NN}) \leq \theta_l) = 1$, that is, $\epsilon = 0$ because $D_l^2(X, X_{1NN}) \leq D_m^2(X, X_{1NN}) \leq D_m^2(X, \hat{X}_{1NN}) = \theta_l$ holds. In PDS algorithm, the distance calculation is stopped for a sample Y when $D_l^2(X,Y) > D_m^2(X, \hat{X}_{1NN})$ for some l. In this paper, we consider its probabilistic version. In this version, we establish the value of θ_l so as to make the *misjudgement error* $\epsilon = P(D_l^2(X, X_{1NN}) > \theta_l)$ sufficiently small and to make *full-distance-calculation probability* $\delta = P(D_l^2(X,Y) < \theta_l)$ as small as possible.

The algorithm is very simple. First we find the best order of coordinates using principal component analysis. Next, we obtain the empirical distribution functions $\hat{F}_l(D_l^2(X, X_{1NN}))$ and $\hat{G}_l(D_l^2(X,Y))$ for $l = 1, 2, \ldots, l_{max}$ in the principal coordinates, where $\hat{F}_l$ is the empirical distribution function of the squared distance between any point and its nearest neighbor and $\hat{G}_l$ is the empirical distribution function of the squared distance between any two points generated according to the underlying distribution. Then for a specified value of ϵ, we determine the value of θ_l as the minimum value satisfying $1 - \hat{F}_l(\theta_l) < \epsilon$ (an empirical percentile approach). At the same time, we can know the estimate of δ as $\hat{G}_l(\theta_l)$. In the searching phase, we exempt sample Y from the full-dimensional calculation if $D_l^2(X,Y) > \theta_l$ for some l. The metric considered in this paper is Euclidean for which we can analyze the performance of the proposed algorithm theoretically.

3 Algorithm

Now, we describe our *marginal distance strategy* (*MDS*) algorithm [1]. It is a probabilistic variant of partial distance searching.

3.1 Preprocessing

First, for a prespecified value of l_{max}, we find l_{max} eigen vectors of the covariance matrix $\hat{\Sigma}$ estimated from all of the training samples. Next, we randomly choose n' samples (n' is fixed at 1000 in the following experiments) from all n samples. For these n' samples, we find their kth nearest neighbors. We then project all of the training samples on the l_{max}-dimensional subspace spanned by the l_{max} principal (eigen) vectors. In each $l(= 1, 2, \ldots, l_{max})$ dimension, we obtain two empirical densities of $D_l^2(X, X_{kNN})$ and $D_l^2(X, Y)$, where X and Y are taken only from n' samples, but X_{kNN} is found from all n samples. In this way, we obtain two empirical distributions $\hat{F}_l(D_l^2(X, X_{kNN}))$ and $\hat{G}_l(D_l^2(X, Y))$.

3.2 User choice of parameters

After we obtain two empirical distributions $\hat{F}_l$ and $\hat{G}_l$ ($l = 1, 2, \ldots, l_{max}$), we prompt the user to specify the value of confidence (error) parameter ϵ and the value of marginal dimension l. For helping the user, two estimated ratios are presented for several candidate values of ϵ and l: the expected full-distance-calculation ratio δ_l and the expected time reduction ratio δ_l^*. Here, δ_l is directly obtained from the empirical distribution $\hat{G}_l(D_l^2(X, Y) = \theta_l)$, where θ_l is taken

[1] The code is available upon request.

so as to satisfy $\hat{F}_l(\theta_l) \simeq 1 - \epsilon$, while δ_l^* is calculated from δ_l by Eq. (1) which will be given later. In the implementation, the user is required to specify the values of ϵ and l by looking at the displayed table in which δ_l and δ_l^* are shown corresponding to some values of ϵ and $l(= 1, 2, \ldots, l_{max})$. For the value of l, the user is allowed to use the estimated optimal value l_{opt} which will also be given later in Eq.(2). So, the user may choose only the value of ϵ. Then the user knows the values of δ_l and δ_l^* before searching.

3.3 Searching algorithm

The searching algorithm is shown in Fig. 2. Steps 1.1.s-e show the marginal strategy using the projected l-dimensional vectors. Here we notice that it is apparently better to apply the marginal strategy (thresholding) in every dimension from 1 to l for reducing the searching time. However, this results in a larger error than the specified value of ϵ. Since our error calculation is based on a specified one value of l, the error has to be accumulated if we use such a continuous application of marginal strategy. We therefore do not adopt such a strategy. In Steps 1.2.s-e, we restart the distance calculation in support of the partial distance strategy, from the first dimension using the original m-dimensional vectors instead of the projected vectors. This is obviously inefficient, since we can continue the distance calculation from the $(l + 1)$th dimension. However, we intentionally do this because the cost of obtaining all of the m projected values of the query point is high. To obtain the m projected values, we have to carry out inner product operations m times using m eigen vectors of $\hat{\Sigma}$ or m orthonormal vectors of which the first l vectors are identical to the l principal (eigen) vectors. Therefore, we find only l projected values of the query point for the marginal strategy and discard away

the calculated partial distance if the marginal criterion does not perform. Steps 1.3.s-e are the ordinal update procedure of the current solution.

3.4 Analysis of algorithm

First, let us examine the space complexity necessary both for preprocessing and for searching. When we use l dimensions for MDS, in addition to keeping all n data, we need $O(ln)$ for storing the projected n vectors. In addition, we need $O(lm)$ for keeping l principal (eigen) vectors of length m. Here, l is usually small enough. As a result, the memory storage is still $O(nm)$, which is a minimum requirement for keeping all n training data.

Next, let us clarify the time complexity. In the preprocessing, we have to obtain l principal vectors from the estimated covariance matrix $\hat{\Sigma}$. Roughly $O(nm^2)$ is needed for calculation of $\hat{\Sigma}$ and $O(ml)$ is needed for calculation of l principal vectors. We estimate two distributions $F_l(D_l^2(X, X_{kNN}))$ and $G_l(D_l^2(X, Y))$ $(l = 1, 2, \ldots, l_{max})$ from $n'(< n)$ sampled points. We need $O(n'n)$ for the former distribution and $O(n'^2)$ for the latter distribution. Thus, we can regard it as $O(n)$ for $n' \ll n$.

In the searching process, it is better to derive the detailed estimation. So we divide the calculation cost into three parts. Here let us ignore the effect of the partial distance strategy. Then, for processing a query point, we need:

(1) $\delta_l n \cdot m$ (distance) operations with cost c_1 (one subtraction, one multiplication, and one addition). Here, at rate δ_l, we execute this full distance calculation.

(2) $l \cdot m$ (inner product) operations with cost c_2 (one multiplication plus one addition). This is to obtain the projected l values of the query point.

(3) $n \cdot l$ (distance) operations with cost c_1 for obtaining the squared distance between the projected query point and every data point in l dimensions.

The overall operation cost is estimated by

$$\begin{aligned} T_l &= c_1 \delta_l n m + c_2 l m + c_1 n l \\ &= \left\{ \delta_l + \frac{c_2 l}{c_1 n} + \frac{l}{m} \right\} T_0 \\ &= \delta_l^* T_0, \\ &\quad \text{where} \quad T_0 = c_1 n m, \quad \delta_l^* = \delta_l + \frac{c_2 l}{c_1 n} + \frac{l}{m}. \end{aligned} \tag{1}$$

Here, T_0 is the cost consumed in the brute-force algorithm and δ_l^* is the estimated time reduction rate (δ_l being the ratio at which full distance calculation is executed). The order is $O(\delta_l n m)$ for $l \ll m$ and $l \ll n$.

3.5 Setting of optimal marginal dimension

For optimal setting of the value of marginal dimension l, we can use Eq. (1) as

$$l_{opt} = \arg\max_l \delta_l^* \left(= \delta_l + \frac{c_2 l}{c_1 n} + \frac{l}{m} \right). \tag{2}$$

Here, $\delta_l = \hat{G}_l(\theta_l)$ and θ_l is determined by a specified ϵ as $\theta_l = \hat{F}_l^{-1}(1-\epsilon)$. In the following experiments, we regarded c_1 and c_2 as the same to obtain l_{opt}. Once l_{opt} is determined, we can estimate the searching time by $\delta_{l_{opt}}^* T_0$.

4 Experiments

4.1 Compared algorithms

Comparison was made with one ANN algorithm and three PACNN algorithms. The ANN algorithm [21] [2] (version 0.2) belongs to ANN framework, and a PACNN algorithm using an M-tree [24] [3], shortly M-PAC, and E^{2}LSH [4] [26] belong to PACNN framework. MDS algorithm belongs to PCNN framework. We also implemented a sequential PACNN algorithm, shortly seq-PAC, given in [24].

The main problem in comparison of these algorithms is the difference among frameworks. Note that in ANN approach we cannot know how many answers are correct, while all the answers are close to the correct nearest neighbors up to $(1+\eta)$ times. On the other hand, in PCNN approach, we cannot know to what degree the obtained nearest neighbor is apart from the correct nearest neighbor when the answer is not correct. In PACNN approach, both are unknown. In these respects, it is not easy to compare them fairly. Therefore, we chose as a common performance indicator the probability of correct nearest neighbors $P(\hat{X}_{1NN} = X_{1NN})$. This is included in the criterion of MDS, but it will be used only for comparing the other performance measures such as the actual approximation ratio η^* and the searching time. In the following, we will compare algorithms in a comparable value of this probability.

The ANN algorithm has an *approximation parameter* η to guarantee $D(X, \hat{X}_{1NN}) \leq (1+\eta)D(X, X_{1NN})$. The M-PAC algorithm and the seq-PAC algorithm need

[2] The code is available at http://www.cs.umd.edu/∼mount/ANN/.
[3] The code is available at http://www-db.deis.unibo.it/Mtree.
[4] The code is available at http://web.mit.edu/andoni/www/LSH/.

a confidence parameter ϵ in addition to η to attain $P\left(D(X, \hat{X}_{1NN}) \leq (1+\eta)D(X, X_{1NN})\right) \geq 1-\epsilon$. E^{2}LSH has a *radius parameter* R, instead of η, to guarantee $P(D(X, \hat{X}_{1NN}) \leq R) \geq 1-\epsilon$. MDS needs only ϵ to guarantee $P(D(X, \hat{X}_{1NN}) = D(X, X_{1NN})) \geq 1-\epsilon$. In the following experiments, several values of some parameters in each algorithm were tested. In MDS, we used $l_{max} = 10$.

4.2 Artificial datasets

First, we have examined the basic performance of MDS algorithm. Our primary concern is to what degree of efficiency is obtained by allowing a small amount of error ϵ. For evaluation, we compared it with the ANN algorithm ($\eta = 0$) and PDS algorithm. Both algorithms return the correct nearest neighbors. The ANN algorithm implements a k-d tree structure tuned for ANN. It also employs an incremental distance update technique. Even if the approximation parameter η is set to zero to find the correct k nearest neighbors, the algorithm is still one of the fastest algorithms to find the correct nearest neighbors in large-dimensional problems.

We used two kinds of normal distributions as follows .

Data A: $N(\mathbf{0}, I_m), m = 100$

Data B: $N(\mathbf{0}, \Sigma), \sigma_j^2 = m/(sj)$ $(j = 1, 2, \ldots, m(= 100))$, where $s = \sum_{i=1}^{m} 1/i$ such that $\mathrm{tr}\Sigma = m = 100$.

Nearest neighbor: $k = 1$

Sample size: $n = 1000, 100000$

Dimension $l = 1, 2, 3$

Procedure: Project all of the data on the first l dimensions (just using the first l elements). Then calculate $D_l^2(X, X_{1NN})$ for any X and calculate $D_l^2(X, Y)$ for any two points X and $Y (X \neq Y)$. In the case of $n = 100,000$, only 1,000 randomly chosen data are used for calculation of the mean and variance, but the nearest neighbors are found from all of the data in the original dimension m.

Results: Fig. 3 and Fig. 4.

The relationship between δ (the ratio of full-distance calculation) and ϵ (the error ratio) is shown in Fig. 3. From Fig. 3, we see 1) in the case of largely different variances (Data B) we can have a large amount of efficiency even for $l = 1$, 2) even for a small ϵ, say $\epsilon = 0.01$, some amount of reduction is obtained, and 3) the increase of sample number n contributes to the efficiency to some extent and the increase of marginal dimension l contributes more.

From Fig. 4 for comparison of three algorithms, we see

(1) The ANN with $\eta = 0$ only performs well in low dimensions. For larger dimensions, it runs almost linearly in m.

(2) The partial distance strategy performs well for any dimension.

(3) For a standard normal distribution (Data A with $n = 10^5$), as expected,

only a small amount of gain is obtained in MDS compared with the partial distance strategy.

(4) For a normal distribution with different variances (Data B, $n = 10^5$), the MDS approach performs well for all dimensions. In this case, we can obtain a large amount of reduction in searching time at the expense of a small degree of error, say, 0.1% or 1.0%.

4.3 Phoneme dataset

Next, we conducted an experiment on a Japanese phoneme dataset. There are 41 phonemes including five vowels. We used 52 400 words uttered by 10 males and equally divided the entire data into a training sample set and a testing sample set. The numbers of phonemes extracted from the words were 124 673 for training and 124 019 for testing. For each phoneme, we obtained 151 features including 14 LPC cepstrum coefficients plus the power in each frame of 10 frames and the original frame length before normalization of length.

The results are shown in Tables 1–5. From these tables, we notice that

(1) Among five algorithms, MDS and ANN are two of the fastest algorithms. They are comparable in any one of the searching speed, the precision of 1NN, the class label precision, and the degree of approximation. LSH is comparable to them only in a few settings.

(2) A wrong nearest neighbor does not necessarily mean a wrong class label. It is often the case that the found nearest neighbor is different from the correct one but the class label is still the same as the correct one. We can confirm this from the values of "Class Label Precision" that are larger than those of "Precision of 1NN." As a result, "Recognition Rate" is

almost kept to the same level as the original 1NN for $1-\epsilon = 0.999, 0.99$ in MDS and ANN.

(3) In MDS, the actual error rate of NNs is almost the same as the value of ϵ that was specified by the user (Compare the predicted value of $1-\epsilon$ in the top line and the value of "Precision of 1NN."). In addition, the predicted full-distance-calculation rate δ is very close to the actual value δ^*. Here δ^* is not the actual time reduction rate but the actual reduction rate of full-distance-calculation. This implies that we can know in advance to what degree we can reduce the searching cost and how much cost we have to pay for that.

In this comparison, there were many practical problems other than the essential difference of frameworks. The M-PAC algorithm requires an M-tree for searching. However, since there are many parameters that have to be chosen appropriately according to the given dataset, e.g., the node size, the number of histogram bins and the memory size, it was very hard to construct a good M-tree. We have tried several values of these parameters to create an appropriate M-tree, but no satisfactory setting was obtained. It also required a large amount memory that prevents us from examining large datasets. Therefore, we examined its naive version without indexing, seq-PAC, too. Unfortunately, the criterion used in seq-PAC is a "stop condition" [24] that is stronger than a "filter condition" used in MDS. Indeed, with $\eta = 0$, only at ratio of $\epsilon/2$ nearest neighbors satisfy this condition, assuming that the stop condition is satisfied just in the half of probing all training samples. As a result, the cost of full-distance calculation is reduced to $1-\epsilon/2$ of that of brute-force algorithm. Even if $\eta > 0$ is adopted, the efficiency is not so large. As a result, the computation time was higher than that of the brute-force algorithm optimized by a compiler. In E^2LSH, the setting of the radius parameter R is also not

easy, because R is an absolute value and an appropriate value of R depends on the problem. The class-label precision of E^{2}LSH is lower than those of the other algorithms, because it returned no answer when no point did not meet the criterion. In ANN algorithm, it was also hard to find the best value of η to attain a good searching time, while keeping an acceptable precision. There is no problem if the actual value of η is predicable, but it is usually far better than the use-specified value. Therefore, it was hard to find the best trade-off between the precision and the searching time. On the contrary, in M-PAC algorithm, the actual value of ϵ is rather large than the specified value.

4.4 MNIST dataset

Next, we used the MNIST dataset of ten handwritten digits [29]. It contains 60 000 points, each having dimension $m = 784 (= 28 \times 28)$. It provides another test set of 10 000 points. According to [26], the points were normalized so that each point has its norm equal to one.

The results are shown in Tables 6–8. Since M-PAC algorithm and seq-PAC algorithm were too slower than the others, the results are not shown. We can see that MDS is superior to the other two algorithms in searching time for comparable probabilities of correct nearest neighbors. It is also noted that, even in this dataset, the actual closeness $1 + \eta^*$ of ANN is much smaller than the specified value $1 + \eta$. In MDS, for both ϵ and δ, the estimated values are sufficiently close to the actual values, and the adopted marginal dimension l_{opt} $(= 6, \ldots, 9)$ is quite small compared with $m = 784$.

The relationship between the searching time and the data size n is shown in Fig. 5. For different values of n, we randomly chose n samples from 60 000

training samples. From Fig. 5, we see that the searching time of ANN and E^2LSH increases linearly, or a little worse, with n, while the searching time of MDS increases somewhat sublinearly with n.

5 Boosting of MDS algorithm

The proposed MDS algorithm is a filter approach (Step 1.1). It scans all samples even if many samples are only tested in the marginal dimension $l(< m)$. In this section, we describe a method to improve it to a pre-cutting approach. The idea is simple. As we have seen so far, the algorithm works even for $l = 1$ to some extent. Therefore, application of MDS in $l = 1$ is beneficial.

In the preprocessing, we sort all stored samples in the first principal coordinate. Then we check only samples Y with the first coordinate value $Y_1 \in [X_1-\theta_1, X_1+\theta_1]$ for a query sample X with $\theta_1 = \hat{F}_1^{-1}(1-\epsilon_1)$ for a given acceptable error ϵ_1. This range is easily found by a binary search in $O(\log n)$ or by a table look-up in $O(1)$ with a set of sufficiently fine cells on the coordinate. The table look-up (or hashing) manner is easily implemented even for $l > 1$ with space of $O(B^l)$, where B is the number of cells in each coordinate, but $l = 1$ is better in the practical sense. So we implemented it in our boosted version of MDS. The algorithm is shown in Fig. 6 as a replacement of Step 1.s of the original algorithm. When we use this boosting, the error is a little increased. The error is upper-bounded by the sum of the originally-specified error ϵ and the error ϵ_1 in this pre-cutting process. The latter ϵ_1 is also specified by a user.

The results for phoneme and MNIST datasets show that this boosting works effectively (Tables 9 and 10). In these cases, θ_1 is determined with $\epsilon_1 = 0.1\%$ so that the total error estimate has to be added by 0.1%.

6 Distribution of Pair Distances and kNN Distances

So far, from the experimental results, we have seen that the proposed algorithm works well for large datasets. Here, we analyze the method from a theoretical viewpoint.

Let us start with investigating the relationship between $\epsilon_l = P(D_l^2(X, X_{1NN}) > \theta)$ and $\delta_l = P(D_l^2(X, Y) < \theta)$ for a threshold θ. Here, ϵ_l is the probability that a wrong 1-NN is returned and δ_l is the probability that a full-distance calculation is carried out in MDS. Let F_l be the distribution function of $D_l^2(X, X_{1NN})$ and G_l be the distribution function of $D_l^2(X, Y)$. Then we have

$$\delta_l = G_l\left(F_l^{-1}(1-\epsilon_l)\right). \tag{3}$$

This equation implies that a larger difference between F_l and G_l brings a smaller value of δ_l. It is usually difficult to have an explicit form of Eq. (3) for F_l and G_l, so we have to solve this numerically for general distributions. However, for some special distributions, we can further analyze. Let us analyze uniform distributions and normal distributions.

Let $r = D(X, X_{kNN})$ be the distance between X and its kth nearest neighbor X_{kNN} in an m-dimensional space. Given a sufficient number of samples, we may consider that X_{kNN} is generated according to the uniform distribution on the surface S^{m-1} of the hyper-sphere of radius r centered at X. Then, the expected radius $E\{r_l\} = E\{D_l(X, X_{kNN})\}$ is obtained from the projection of S^{m-1} onto the l-dimensional subspace. Indeed, we can show that the expected

second and fourth moments of r_l are given by

$$\begin{aligned} E\{r_l^2\} &= \frac{l}{m} r^2, \\ E\{r_l^4\} &= \frac{l(l+2)}{m(m+2)} r^4. \end{aligned} \tag{4}$$

Therefore, it suffices to analyze $r^s (s = 2, 4)$ in the original m-dimensional space for obtaining the mean and variance of r_l^2 over r and X.

In general, we have a conditional expectation of r^s (for example, see Fukunaga [28], pp. 277–280) as

$$E\{r^s|X\} = (p(X)V_m)^{-\frac{s}{m}} \beta_{k,m,s} \alpha_{n,m,s}, \tag{5}$$

where $V_m = \pi^{m/2}/\Gamma(1 + m/2)$ (the volume of the unit hyper-sphere in m-dimensional space), $\beta_{k,m,s} = \frac{\Gamma(k+s/m)}{\Gamma(k)} \simeq k^{s/m}$, and $\alpha_{n,m,s} = \frac{\Gamma(n+1)}{\Gamma(n+1+s/m)} \simeq (n+1)^{-s/m}$. In the following, we use $\alpha_{n,m}$ for $\alpha_{n,m,2}$. Note that k is included only in $\beta_{k,m,s}$ and n is included only in $\alpha_{n,m,s}$. Also note that $\beta_{k,m,s}$ can be regarded as one for any small k and large m. Fortunately, we can derive the expected values of r^s over X for some special distributions $p(X)$. For example, for a normal distribution with mean zero and covariance matrix Σ, we have

$$E\{p(X)^{\frac{s}{m}}\} = (2\pi)^{\frac{s}{2}} |\Sigma|^{\frac{s}{2m}} (1 - \frac{s}{m})^{-\frac{m}{2}}.$$

Therefore, with Eq. (4) and Eq. (5), we can derive the mean and covariance of r_l^2. The results for normal distributions with a diagonal covariance matrix and uniform distributions are shown in Table 11. The concrete derivation is given in Appendices A and B.

From Table 11, we see that both of the mean and variance of $D_l^2(X, X_{NN})$

increase linearly in the marginal dimension l. Note that the standard deviation increases linearly in $\sqrt{l}$. Another important thing is that there can be a large difference between $F_l(D_l^2(X, X_{NN}))$ and $G_l(D_l^2(X, Y))$ even for $l = 1$ if either the (diagonalized) covariance matrix Σ is different from the identity matrix, except for a constant factor, or the sample number n is fairly large to m, e.g., $\log n > m$. The first situation is often observed in real-life datasets and the difference of distributions comes from the fact that $\mathrm{tr}\Sigma_l \geq l|\Sigma_l|^{\frac{1}{l}} \geq l|\Sigma_m|^{\frac{1}{m}}$ holds in general (the harmonic mean is less than or equal to the arithmetic mean.).

7 Order analysis of MDS

Last, let us show that the full-distance calculation ratio δ in MDS algorithm is $o(1)$ in n. This is intuitively clear from the fact that F_l reduces to zero along with $\alpha_{n,m}(= (n+1)^{-2/m})$ as n goes to infinity (Table 11). Here we give a general evaluation.

Let us imagine two uni-variate densities with mean μ_i and variance σ_i^2 $(i = 1, 2)$ (Fig. 7). In MDS, the first distribution is of $Z_1 = D_l^2(X, X_{NN})$ and the second is of $Z_2 = D_l^2(X, Y)$. For a specified t such as $\mu_1 \leq t \leq \mu_2$, we consider two errors of $P\{Z_1 \geq t\}$ and $P\{Z_2 \leq t\}$. By one-sided Chebyshev's inequality, we have, for $t \geq 0$,

$$P\{Z \leq \mu - t\}, P\{Z \geq \mu + t\} \leq \frac{\sigma^2}{\sigma^2 + t^2}.$$

Hence,

$$\epsilon = P\{Z_1 \geq t\} = P\{Z_1 \geq \mu_1 + (t - \mu_1)\} \leq \frac{\sigma_1^2}{\sigma_1^2 + (t - \mu_1)^2},$$
$$\delta = P\{Z_2 \leq t\} = P\{Z_2 \leq \mu_2 - (\mu_2 - t)\} \leq \frac{\sigma_2^2}{\sigma_2^2 + (\mu_2 - t)^2}.$$

Hence, for a given acceptable error ϵ_0, by taking t as

$$t = \mu_1 + \sigma_1 \sqrt{\frac{1}{\epsilon_0} - 1}, \tag{6}$$

we can upper-bound the both errors as

$$\epsilon \leq \epsilon_0,$$
$$\delta \leq \frac{\sigma_2^2}{\sigma_2^2 + (\mu_2 - t)^2} = \frac{\sigma_2^2}{\sigma_2^2 + \left(\mu_2 - \mu_1 - \sigma_1 \sqrt{\frac{1}{\epsilon_0} - 1}\right)^2}. \tag{7}$$

The second inequation connects ϵ (exactly speaking, ϵ_0) to δ. This relationship holds independently of underlying distributions, but the evaluation is quite loose. In general, the value of t is less than the estimation Eq. (6). Indeed, for normal distributions, we know that $t = \mu_1 + 3.09\sigma_1$ for $\epsilon_0 = 0.1\%$ and $t = \mu_1 + 2.33\sigma_1$ for $\epsilon_0 = 1\%$ that are far less than the coefficient in Eq. (6).

Returning to the original problem, we assume that the first density is for the k-NN squared distance and the second is for the squared distance of any pair. Then, as the sample number n increases, the second density does not change, while the first density converges to zero. The convergence is guaranteed by the convergence of mean and variance as

$$\mu_1 = \alpha_{n,m} M \quad \text{and} \quad \sigma_1^2 = \alpha_{n,m}^2 S^2 \tag{8}$$

Here M and S are constants depending on the underlying distribution. It

should be noted that the convergence ratio is governed only by $\alpha_{n,m}$ that is independent of underlying distributions.

From (7) and (8), we have

$$\delta \leq \frac{\sigma_2^2}{\sigma_2^2 + \left(\mu_2 - \alpha_{n,m}\left(M + S\sqrt{\frac{1}{\epsilon_0} - 1}\right)\right)^2} \rightarrow \frac{\sigma_2^2}{\sigma_2^2 + \mu_2^2} \quad \text{as} \quad n \rightarrow \infty. \tag{9}$$

The last convergence comes from that fact that $\alpha_{n,m} \rightarrow 0$ as $n \rightarrow \infty$ for fixed m.

Inequation (9) shows that δ converges to a constant. However, in general, it converges to zero. Inequation (9) is derived from Chebyshev's inequality that holds for any distribution in which the random variable can take a negative value. However, for the (squared) distance variable under consideration takes only a non-negative value, so that F_l convergences to zero in probability one and the left tail of G_l reduced to zero quickly. Hence, we have confirmed that for a fixed ϵ_0, $\delta = o(1)$. It is also interesting to notice that $\frac{\sigma_2^2}{\sigma_2^2+\mu_2^2} \rightarrow 0$ as $l \rightarrow \infty$ with $\mu_2^2 = O(l^2)$ and $\sigma_2^2 = O(l)$ (for example, $\mu_2^2 = (2l)^2$ and $\sigma_2^2 = 8l$ for standard normal distributions (Table 11)). This means that a larger value of l brings a larger difference between these two distributions. Indeed, we can see the effectiveness of l in the previous two datasets both of which have many classes. The curve of δ is shown in Fig. 8. It is clear that δ is small even for $l = 1$ and decreases according to the increase of the value of l. Note that the estimated optimal l_{opt} was around 5 for the phoneme dataset and around 8 for the MNIST dataset. This value reflects the trade-off between δ and l in Eq. (1).

8 Discussion

Some ideas used in the proposed MDS algorithm are already seen in previous studies. Especially, the usage of empirical distributions and the usage of principle component analysis have been widely seen. For example, M-PAC [24] exploits the empirical distribution in the original m dimensions. However, as seen in Table 11 for comparison between $D_m^2(X, X_{NN})$ and $D_l^2(X, Y)$, we see that we cannot expect much gain using a threshold derived from $D_m^2(X, X_{NN})$. For example, PDS does not work until $l \sim m/2$, since after this value of l both distances become comparable for the first time (see $2l$ vs. m in the mean of a standard normal distribution and l vs. $0.35m (= 0.058 \times 6m)$ for a uniform distribution in Table 11). Therefore, as seen in MDS, lower-dimensional empirical distributions are more beneficial.

In this study, we have considered only the Euclidean measure, that is, L_2 norm. It is difficult to analyze the effect for other norms, but the proposed algorithm can be applied to any Minkowski L_p norm. This is because we determine everything necessary for the algorithm from the empirical marginal distributions. As long as the closeness in the original space remains to some extent in a lower-dimensional space, the algorithm performs depending on the extent. For principle component analysis, some studies have been devoted for L_1 norms [30,31]. Such a study would be appropriate if we want to apply MDS in L_1 norm.

The marginal distance strategy is generally more effective in a classification task than in a simple searching task without class labels. This is because in classification problems we can expect that the nearest neighbors are mostly generated according to the density function of the class that the query sample

belongs to. In general, $\Sigma_T = \Sigma_W + \Sigma_B$, where Σ_T is the total covariance matrix, Σ_W is the within-class covariance matrix, and Σ_B is the between-class covariance matrix. It is clear that $\Sigma_T > \Sigma_W$ (in eigen values). For a query sample X, its k-nearest neighbors can be expected to being generated from the same class as X, and their distance is therefore estimated from Σ_W, while for an arbitrary sample Y, the distance between X and Y is estimated from Σ_T. Therefore, we can expect a large difference in the two distributions of $G(D^2(X,Y))$ and $F(D^2(X,X_{kNN}))$. As a result, we can expect a large full-distance-calculation value of δ even for a small error value of ϵ. In fact, we can see this from Fig. 8.

The proposed algorithm is beneficial especially in the controllable nature. In the preprocessing stage, we know both the degree of reduction in searching time and the degree of correctness of nearest neighbors. If a user thinks it worth, the user can use the marginal strategy. Otherwise, the user may just ignore such an option. Then the partial distance strategy is invoked without error.

It is easy to modify our PCNN algorithm to a PACNN algorithm. It can be made by replacing $r^2 > \theta_l$ with $r^2 > \theta_l/(1+\eta)^2$ (in Step 1.1) and $r^2 > (r_k^*)^2$ with $r^2 > (r_k^*)^2/(1+\eta)^2$ (in Step 1.2) for given η. These are allowed from the fact that if no updating has been made after $\hat{X}_{kNN}$ was found, then every sample Y satisfies $D(X,Y) \geq D(X,\hat{X}_{kNN})/(1+\eta)$. It is also possible to add a termination process, as seen in [24], just before Step 1.1.e. such as " **if** $(r^2 < (1+\eta)^2\phi_l)$ **then** terminate the process," where $\phi_l = \hat{F}_l(\epsilon)$ that is different from $\theta_l = \hat{F}_l(1-\epsilon)$. Then, with probability $1-\epsilon$, approximate nearest neighbors are found.

Finally, let us consider about the actual searching time. In recent CPUs, for

some reasons, it is difficult to estimate the exact searching time beforehand even in the brute-force algorithm. The actual time depends on CPU, OS and the other practical factors. It also depends on the compiler and the optimization technique. In general, programs with a simpler structure are optimized better. In this sense, the brute-force algorithm gains the most benefit from the optimization and our MDS algorithm follows. Especially, recent compilers, e.g, Intel C compiler, can pull out the maximal performance of multi-core systems using multithreading, vectorization and loop unrolling techniques. With these techniques, simple filtering approach as seen in our MDS algorithm would be most promising, compared with the other algorithms using a complicated data structure. A *memory cache* also affects the actual time, especially when the amount of data is large. It is therefore almost impossible to estimate the actual time from a theoretical analysis.

In the proposed MDS algorithm, we can store in memory only the l principal vectors and the projected n training samples in $O(l(m+n))$. This is clearly better than $O(mn)$ with $l \ll m$ for keeping all of the raw data. When it is necessary, the full vector can be called from an extra storage device.

9 Conclusion

We have proposed a simple k-nearest neighbor search algorithm that performs well for high dimensional cases. This algorithm is advantageous compared to other algorithms only when data size and dimensionality are both large. It exploits the empirical marginal distribution of kth nearest neighbors in a lower dimension. As a result, with a predictable ratio, we can decrease the searching time by only checking the first few values of every projected data

point. The algorithm looses the correctness of the nearest neighbors, but the loss of correctness can be controlled to be arbitrarily small by the user. In some simulated data and two large-scale real-life datasets, it was confirmed that this probably-correct approach is superior in searching time to three typical approximately-correct approaches.

Acknowledgment

The authors would like to thank the anonymous reviewers for their critical comments to improve the quality of this paper.

Table 1
Results of MDS for the phoneme dataset.

	1-NN	MDS algorithm $(1-\epsilon)$			
		(0.999)	(0.99)	(0.95)	(0.90)
Marginal dimension l_{opt}	—	6	5	4	3
Precision of 1NN (%)	100.00	99.89	99.77	99.06	90.70
Class Label Precision (%)	100.00	99.96	99.94	99.80	98.69
Recognition Rate (%)	96.06	96.03	96.01	95.94	95.40
Approximation $(1+\eta^*)$	1.000	1.000	1.000	1.001	1.007
Estimated Reduction Rate (δ) (%)	—	1.407	1.714	0.852	1.187
Actual Reduction Rate (δ^*) (%)	—	1.234	1.680	0.856	1.195
Searching Time (s)	12035	618	608	666	533

Table 2
Results of ANN for the phoneme dataset.

	ANN Algorithm (η)					
	(0.0)	(1.0)	(1.4)	(1.6)	(2.0)	(4.0)
Precision of 1NN (%)	100.00	99.77	99.24	98.85	97.74	84.44
Class Label Precision (%)	100.00	99.98	99.94	99.91	99.82	98.71
Recognition Rate (%)	96.06	96.04	96.01	95.99	95.96	95.61
Approximation $(1+\eta^*)$	1.000	1.000	1.000	1.001	1.001	1.011
Searching Time (s)	2571	964	683	581	435	147

Table 3
Results of M-PAC for the phoneme dataset. The data size is 5000 (about 5% of the original dataset). In below, "Converted Searching Time" is multiplied by a ratio of 124019/5000.

	M-PAC Algorithm (η, $\epsilon = 0.0001$)		
	($\eta = 0.0$)	(0.001)	(0.01)
Precision of 1NN (%)	82.08	82.06	80.84
Class Label Precision (%)	99.70	99.70	99.64
Recognition Rate (%)	95.58	95.68	95.52
Searching Time (s)	1025	1026	991
Approximation ($1 + \eta^*$)	1.249	1.249	1.292
Converted Searching Time (s)	25429	25448	24580

Table 4
Results of sequential PAC for the phoneme dataset.

	seq-PAC Algorithm					
	($\epsilon = 0.1$)			($\epsilon = 0.5$)		
	($\eta = 0.0$)	(0.1)	(0.5)	(0.0)	(0.1)	(0.5)
Precision of 1NN (%)	99.85	99.44	77.93	87.02	70.12	7.84
Class Label Precision (%)	100.00	99.99	99.51	99.79	99.10	59.72
Recognition Rate (%)	96.06	96.05	97.75	96.95	95.46	68.78
Searching Time (s)	21315	21198	16348	17755	15393	8211
Approximation ($1 + \eta^*$)	1.000	1.001	1.031	1.017	1.046	1.346

Table 5
Results of LSH for the phoneme dataset.

	LSH Algorithm (R, ϵ)					
	$(R = 20)$		$(R = 18)$		$(R = 15)$	
	$(1 - \epsilon = 0.99)$	(0.90)	(0.99)	(0.90)	(0.99)	(0.90)
Precision of 1NN (%)	99.93	99.83	99.48	99.16	91.79	90.49
Class Label Precision (%)	99.94	99.91	99.50	99.41	91.85	91.44
Recognition Rate (%)	96.02	96.01	95.71	95.68	89.23	88.95
Approximation $(1 + \eta^*)$	1.000	1.000	1.000	1.000	1.000	1.000
Searching Time (s)	2290	1085	1350	570	482	229

Table 6
Results of MDS for the MNIST dataset.

	1-NN	MDS algorithm $(1 - \epsilon)$			
		(0.999)	(0.99)	(0.95)	(0.90)
Marginal dimension l_{opt}	—	9	8	8	6
Precision of 1NN (%)	100.00	99.73	98.80	97.98	86.75
Class Label Precision (%)	100.00	99.86	99.47	99.20	95.77
Recognition Rate (%)	88.74	88.72	88.6	88.55	88.38
Approximation $(1 + \eta^*)$	1.000	1.000	1.000	1.001	1.009
Estimated Reduction Rate (δ) (%)	—	2.094	1.039	0.613	0.460
Actual Reduction Rate (δ^*) (%)	—	1.962	1.034	0.664	0.501
Searching Time (s)	2437	52	50	46	38

Table 7
Results of ANN for the MNIST dataset.

	ANN Algorithm (η)					
	(0.0)	(1.0)	(2.0)	(3.0)	(4.0)	(5.0)
Precision of 1NN (%)	100.00	99.99	99.65	97.52	92.13	85.18
Class Label Precision (%)	100.00	100.00	99.97	99.95	98.53	97.18
Recognition Rate (%)	88.87	88.74	88.73	88.61	88.39	88.33
Approximation $(1+\eta^*)$	1.000	1.000	1.000	1.001	1.004	1.009
Searching Time (s)	1046	879	558	299	170	111

Table 8
Results of LSH for the MNIST dataset.

	LSH Algorithm (R, ϵ)					
	$(R=0.8)$		$(R=0.7)$		$(R=0.6)$	
	$(1-\epsilon=0.99)$	(0.90)	(0.99)	(0.90)	(0.99)	(0.90)
Precision of 1NN (%)	99.89	99.82	98.92	98.64	92.19	91.29
Class Label Precision (%)	99.89	99.86	98.92	98.80	92.22	91.69
Recognition Rate (%)	88.66	88.66	87.92	87.68	82.73	82.53
Approximation $(1+\eta^*)$	1.000	1.000	1.000	1.000	1.000	1.000
Searching Time (s)	1063	646	615	296	275	114

Table 9
Results of boosted MDS for the phoneme dataset.

	boosted MDS $(1-\epsilon)$			
	(0.998)	(0.994)	(0.989)	(0.899)
Marginal dimension l_{opt}	3	5	4	4
Precision of 1NN (%)	99.85	99.76	99.06	90.71
Class Label Precision (%)	99.96	99.94	99.80	98.69
Recognition Rate (%)	96.03	96.02	95.94	95.40
Approximation $(1+\eta^*)$	1.000	1.000	1.001	1.007
Searching Time (s)	297	305	329	269

Table 10
Results of boosted MDS for the MNIST dataset.

	boosted-MDS $(1-\epsilon)$			
	(0.998)	(0.989)	(0.949)	(0.899)
Marginal dimension l_{opt}	9	8	8	6
Precision of 1NN (%)	99.54	98.63	97.89	86.75
Class Label Precision (%)	99.80	99.42	99.17	95.77
Recognition Rate (%)	88.72	88.61	88.56	88.38
Approximation $(1+\eta^*)$	1.001	1.001	1.004	1.009
Searching Time (s)	41	30	29	30

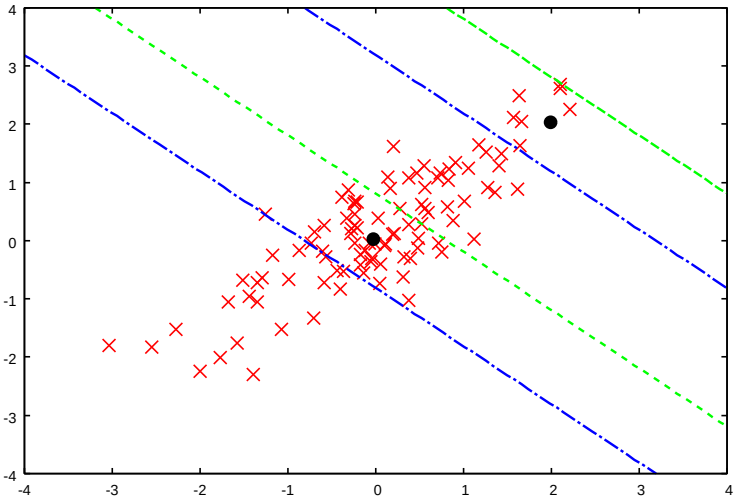
Table 11
Mean and variance of squared distances in the first $l(=1,2,\ldots,m)$ dimensions for normal distributions and uniform distributions. It is assumed that $k=1$ and m is large enough. For normal distributions, $\Sigma_l = \mathrm{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_l^2)$. In below $\alpha_{n,m} = (n+1)^{-2/m}$.

Normal distributions

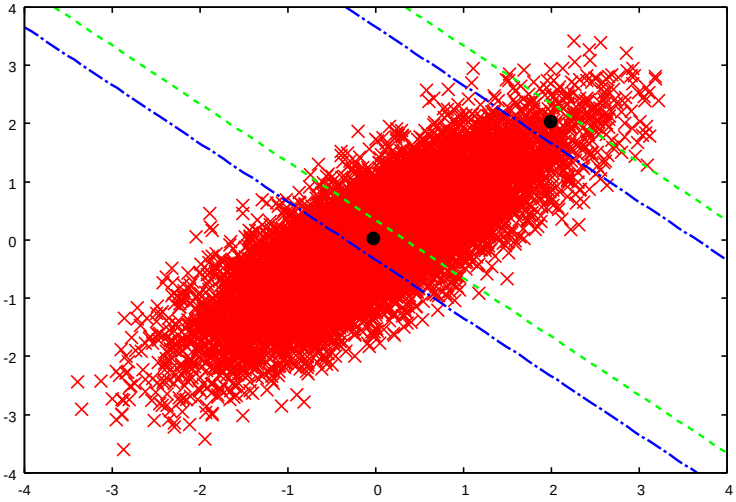
Statistics	$D_l^2(X,Y)$	$D_l^2(X, X_{NN})$	$D^2(X, X_{NN})$
$\Sigma = \mathrm{diag}$			
Mean	$2\mathrm{tr}\Sigma_l$	$l\lvert\Sigma\rvert^{\frac{1}{m}}\alpha_{n,m}$	$m\lvert\Sigma\rvert^{\frac{1}{m}}\alpha_{n,m}$
Variance	$8\mathrm{tr}\Sigma_l^2$	$2l\lvert\Sigma\rvert^{\frac{2}{m}}\alpha_{n,m}^2$	$2m\lvert\Sigma\rvert^{\frac{2}{m}}\alpha_{n,m}^2$
$\Sigma = I$			
Mean	$2l$	$l\alpha_{n,m}$	$m\alpha_{n,m} \simeq m$
Variance	$8l$	$2l\alpha_{n,m}^2$	$2m\alpha_{n,m}^2 \simeq 2m$

Uniform distributions

Statistics	$D_l^2(X,Y)$	$D_l^2(X, X_{NN})$	$D^2(X, X_{NN})$
Mean	$l/6 \sim 0.167l$	$\frac{l}{2\pi e}\alpha_{n,m} \sim 0.058l\alpha_{n,m}$	$\frac{m}{2\pi e}\alpha_{n,m} \sim 0.058m\alpha_{n,m}$
Variance	$7l/180 \sim 0.039l$	$\frac{l}{2\pi^2 e^2}\alpha_{n,m}^2 \sim 0.0068l\alpha_{n,m}^2$	$\frac{m}{2\pi^2 e^2}\alpha_{n,m}^2 \sim 0.0068m\alpha_{n,m}^2$



(a) $n = 10^2$



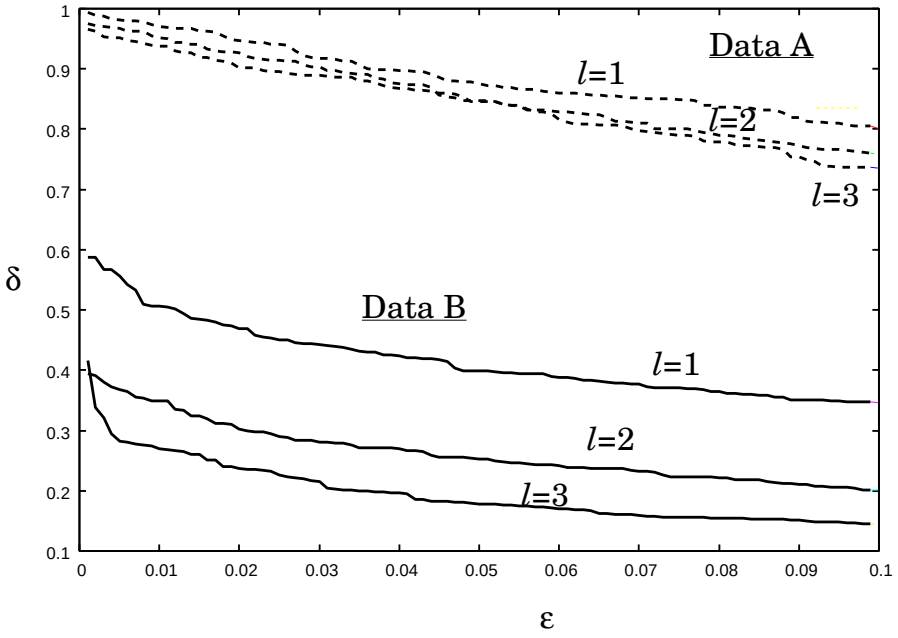
(b) $n = 10^4$

Fig. 1. Search region obtained by the proposed algorithm for a Gaussian with mean $(0, 0)$ and covariance matrix $(1.0, 0.8, 0.8, 1.0)$. Two query points are $(0, 0)$ and $(2, 2)$. Within the search region, 1-NN is included with probability 99.9% regardless of the location of the query point.

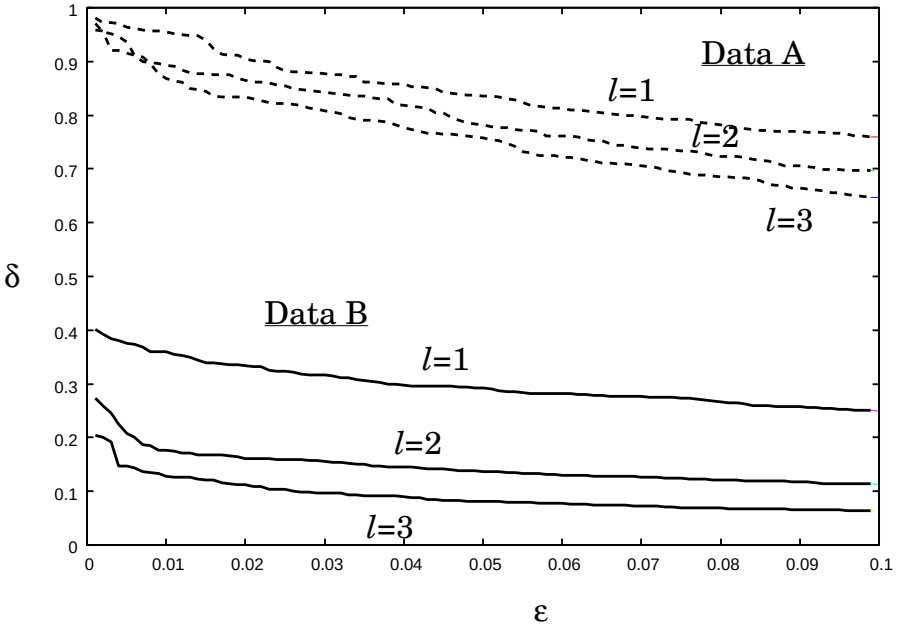
0. *Preparation:*
   $q \leftarrow$ query point
   $q^p \leftarrow \Phi^t q$ /* projection to the l-dimensional space */
   $r_k^* \leftarrow +\infty$
   $1NN, \ldots, kNN \leftarrow 0$

Search:
1.s **for** $i = 1$ to n
1.1.s $r^2 \leftarrow 0$
   for $j = 1$ to l
   $r^2 \leftarrow r^2 + (x_{ij}^p - q_j^p)^2$
   endfor
   if $(r^2 > \theta_l)$ /* marginal distance thresholding */
   Break the process of the ith sample and proceed to $(i+1)$th
1.1.e **endif**
1.2.s $r^2 \leftarrow 0$
   for $j = 1$ to m
   $r^2 \leftarrow r^2 + (x_{ij} - q_j)^2$
   if $(r^2 > (r_k^*)^2)$ /* partial distance thresholding */
   Break the process of the ith sample and proceed to $(i+1)$th
   endif
1.2.e **endfor**
1.3.s **if** $(r^2 < (r_k^*)^2)$ /* update of records */
   Update $1NN, \ldots, kNN$ with distances $r_1, r_2, \ldots, r_k$
   $(r_k^*)^2 \leftarrow r_k^2$ /* update of the kth NN distance */
1.3.e **endif**
1.e **endfor**
2. *Recovery process:*
2.1.s **if** $(kNN = 0)$ /* recovery process */
   Search all of the n data to find $1NN, \ldots, kNN$
2.1.e **endif**
3. output $1NN, \ldots, kNN$

Fig. 2. Searching algorithm MDS.

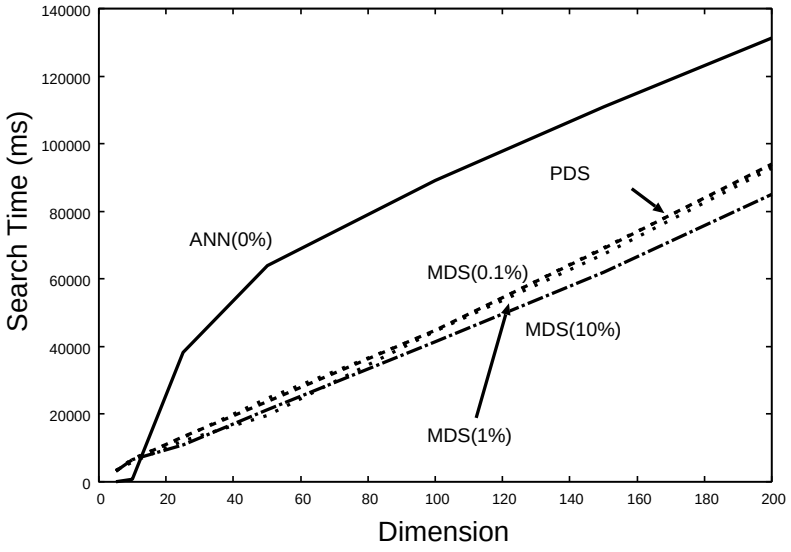


(a) $n = 10^3$

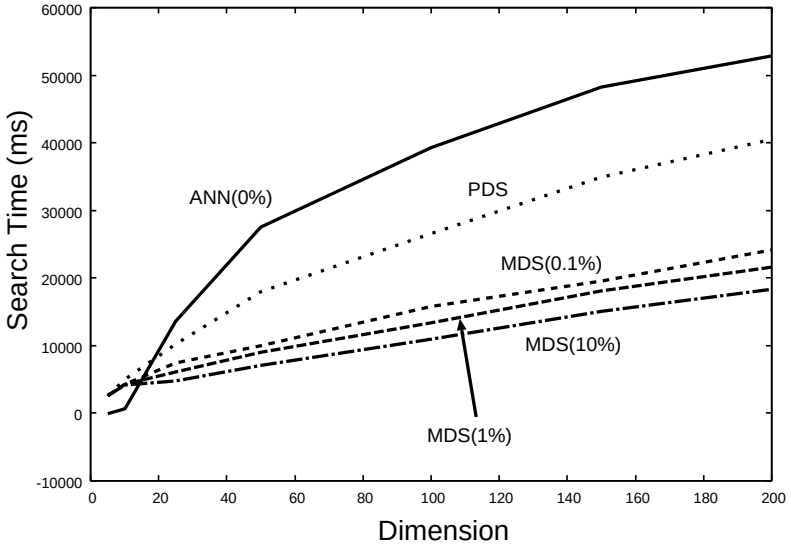


(b) $n = 10^5$

Fig. 3. ROC curve of $\epsilon = Pr(D_l^2(X, X_{kNN}) > \theta)$ and $\delta = Pr(D_l^2(X, Y) < \theta)$ as functions of θ. The marginal distance is $l = 1, 2, 3$.



(a) Data A



(b) Data B

Fig. 4. Searching time of the nearest neighbor ($k = 1$) as a function of dimensionality. The number of stored data is $n = 10^5$ and the number of queries is 10^3. In Data A, the marginal dimension l is fixed at 1. In Data B, $l = 1$ for $m \leq 20$, 2 for $m \leq 50$, and 3 for others.

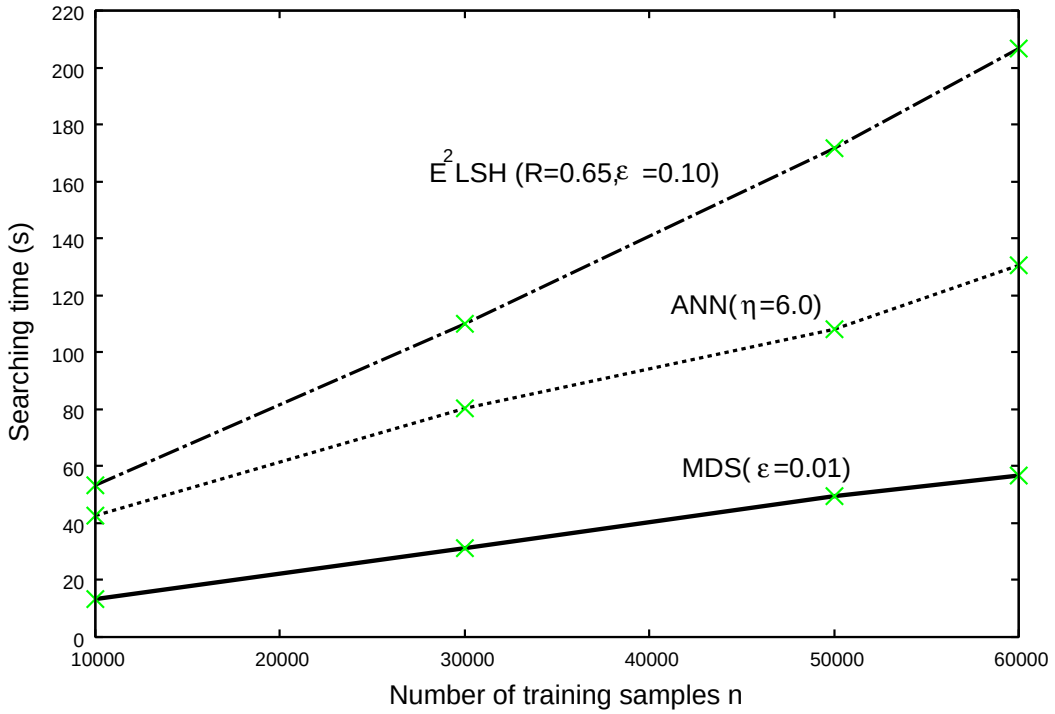


Fig. 5. Time complexity in data size.

Preprocessing:
All data are sorted in the increasing order of the 1st principal coordinate.

Search:
1.s Find the search region with the start index s and end index t
by a binary search or by a table look-up.
for $i = s$ to t

Fig. 6. Boosting process in MDS.

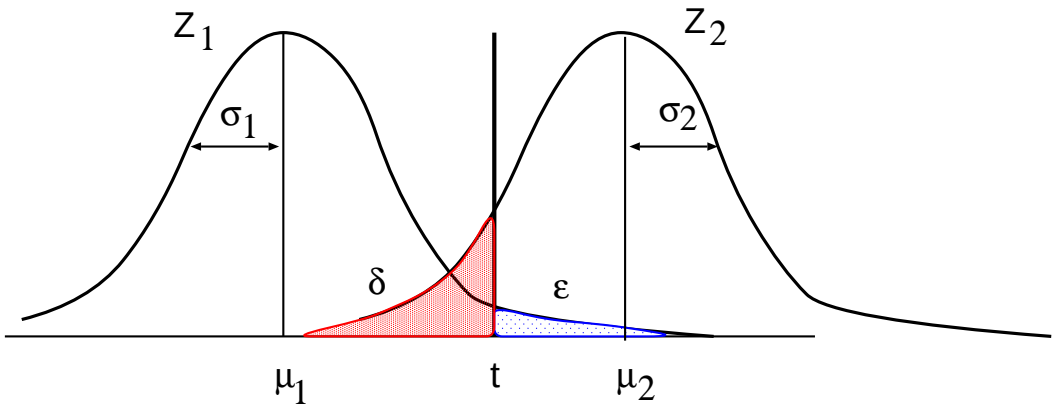
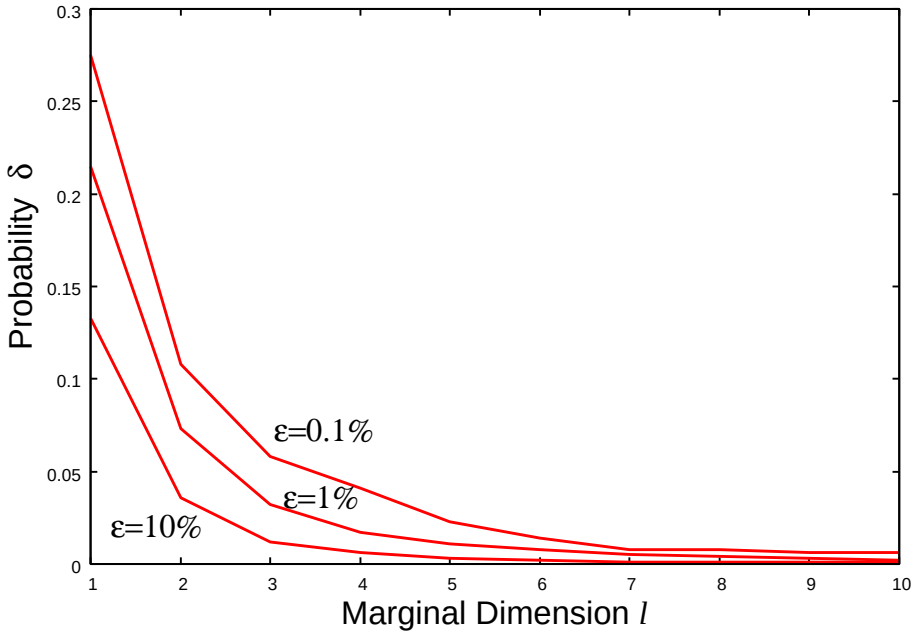
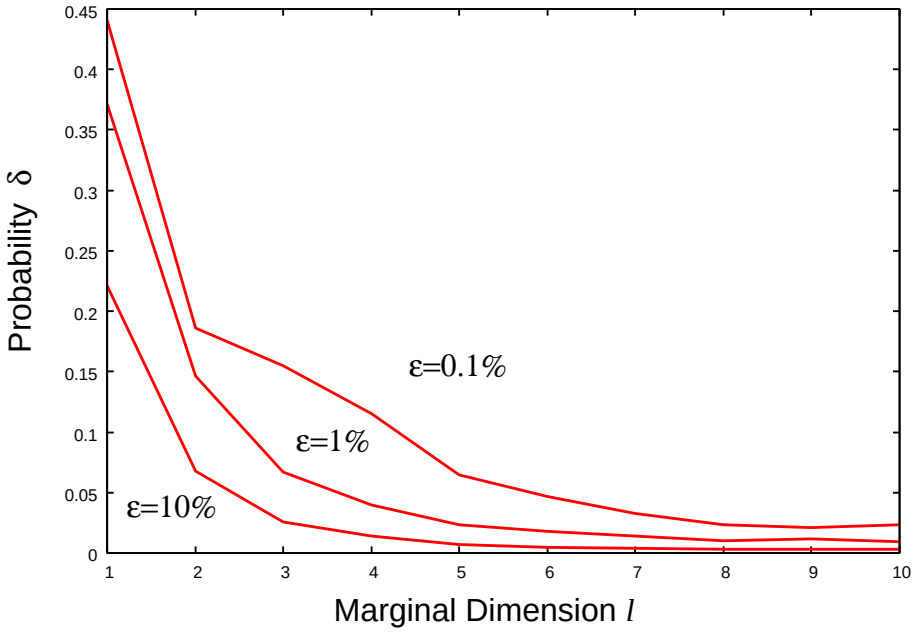


Fig. 7. Two densities causing two kinds of error.



(a) Phoneme dataset



(b) MNIST dataset

Fig. 8. Estimated full-distance calculation probability δ vs. marginal dimension l.

A Distance of pairs in Gaussians

Here, the mean and variance of $D^2(X,Y)$ are given for m-dimensional normal distributions. For uniform distributions, they are easily obtained. As for the marginal version with l, replacing m with l is sufficient. Without loss of generality, we may assume that the covariance matrix is diagonal.

Let $X = (X_1, X_2, \ldots, X_m)', Y = (Y_1, Y_2, \ldots, Y_m)' \sim N(\mathbf{0}, \Sigma)$, where $\Sigma = \mathrm{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_m^2)$ and $\sigma_1^2 \geq \sigma_2^2 \geq \cdots \geq \sigma_m^2 > 0$. With $Z_i = (X_i - Y_i)/\sigma_i$, Z_i is a random variable according to $N(0,2)$. Then, $(Z_i/\sqrt{2})^2$ obeys $\chi_2(1)$. This means

$$\begin{aligned} E(Z_i^2/2) &= 1 & V(Z_i^2/2) &= 2 \\ E(Z_i^2) &= 2 & V(Z_i^2) &= 8 \\ E((X_i - Y_i)^2/\sigma_i^2) &= 2 & V((X_i - Y_i)^2/\sigma_i^2) &= 8 \\ E((X_i - Y_i)^2) &= 2\sigma_i^2 & V((X_i - Y_i)^2) &= 8\sigma_i^4, \end{aligned}$$

where $V(\cdot)$ means the variance. Since every dimension is independent of others, we have

$$E(\sum_{i=1}^{m}(X_i - Y_i)^2) = 2\sum_{i=1}^{m}\sigma_i^2, \quad V(\sum_{i=1}^{m}(X_i - Y_i)^2) = 8\sum_{i=1}^{m}\sigma_i^4.$$

B Distance of kNN

Let u be the probability of the hyper-sphere centered at a fixed point X_0 with radius $r = D(X_0, X_{kNN})$ that is the distance to the kth nearest neighbor of X_0. Thus,

$$u = \int_{D(X_0,Y)<D(X_0,X_{kNN})} p(Y)dY.$$

By considering the kth order statistics $X_{(k)} = X_{kNN}$ in distance $D(X_0, X_i)(i = 1, 2, \ldots, n)$, the following is easily derived:

$$p(u) = \frac{n!}{(k-1)!(n-k)!} u^{k-1}(1-u)^{n-k}.$$

That is, $p(u)$ is a Beta distribution $Be(k, n-k+1)$. Note that this holds independently of the underlying distribution $p(X_0)$.

We assume, with a sufficiently large number of samples, that $r = D(X_0, X_{kNN})$ is small enough to allow the first order approximation

$$u \simeq p(X_0) V_m r^m,$$

where $V_m = \pi^{m/2}/\Gamma(1 + m/2)$, the volume of the unit hyper-sphere in m-dimensional space. Then, we have

$$r^s = \left(\frac{u}{p(X_0) V_m} \right)^{\frac{s}{m}}.$$

By taking the expectation over $u \in [0, 1]$, we have

$$\begin{aligned}
E_u\{r^s | X_0\} &= \int_0^1 \left(\frac{u}{p(X_0)V_m} \right)^{\frac{s}{m}} p(u) du \\
&= (p(X_0)V_m)^{-\frac{s}{m}} \frac{n!}{(k-1)!(n-k)!} \int_0^1 u^{k-1+s/m}(1-u)^{n-k} du \\
&= (p(X_0)V_m)^{-\frac{s}{m}} \frac{n!}{(k-1)!(n-k)!} \frac{\Gamma(k+s/m)\Gamma(n-k+1)}{\Gamma(n+1+s/m)} \\
&= (p(X_0)V_m)^{-\frac{s}{m}} \frac{\Gamma(k+s/m)}{\Gamma(k)} \frac{\Gamma(n+1)}{\Gamma(n+1+s/m)} \\
&= (p(X_0)V_m)^{-\frac{s}{m}} \beta_{k,m,s} \alpha_{n,m,s}.
\end{aligned} \tag{B.1}$$

Note that k is included only in $\beta_{k,m,s}$ and n is included only in $\alpha_{n,m,s}$. Here, by applying $\Gamma(n+\delta)/\Gamma(n) = n^{\delta}$ for small δ, we have the following approximations for a large m:

$$\begin{aligned}
\beta_{k,m,s} &= \frac{\Gamma(k+s/m)}{\Gamma(k)} \simeq k^{s/m} \simeq 1 \\
\alpha_{n,m,s} &= \frac{\Gamma(n+1)}{\Gamma(n+1+s/m)} \simeq (n+1)^{-s/m} \\
V_m^{-\frac{s}{m}} &\simeq \pi^{\frac{s}{2}(\frac{1}{m}-1)} m^{\frac{s}{2}(\frac{1}{m}+1)} \left(\frac{1}{2e}\right)^{\frac{s}{2}} \simeq \left(\frac{m}{2\pi e}\right)^{s/2}.
\end{aligned}$$

Next, let us examine the distance $r_l = D_l(X_0, X_{kNN})$, that is, the distance between X_0 and X_{kNN} in the l-dimensional space, although X_{NN} is closest to X_0 in the original m-dimensional space. Given a sufficient number of samples, we may consider that X_{kNN} is generated according to the uniform distribution on the surface S^{m-1} of the hyper-sphere with radius r centered at X. Thus, the expected radius of r_l is obtained by the projection of S^{m-1} on the l-dimensional subspace. We can show that the expected second and fourth moments of r_l are given by

$$\begin{aligned}
E_X\{r_l^2\} &= \frac{l}{m} r^2, \\
E_X\{r_l^4\} &= \frac{l(l+2)}{m(m+2)} r^4.
\end{aligned}$$

Then from (B.1) and above equations, we obtain

$$\begin{aligned}
E_u\{r_l^2|X_0\} &= \frac{l}{m}(p(X_0)V_m)^{-\frac{2}{m}} \beta_{k,m,2}\alpha_{n,m,2} \\
&\simeq \frac{l}{m} p(X_0)^{-\frac{2}{m}} \left(\frac{m}{2\pi e}\right)(n+1)^{-\frac{2}{m}} \\
&\simeq l p(X_0)^{-\frac{2}{m}} \left(\frac{1}{2\pi e}\right)(n+1)^{-\frac{2}{m}}
\end{aligned} \tag{B.2}$$

$$
\begin{aligned}
E_u\{r_l^4|X_0\} &= \frac{l(l+2)}{m(m+2)}(p(X_0)V_m)^{-\frac{4}{m}}\beta_{k,m,4}\alpha_{n,m,4} \\
&\simeq \frac{l(l+2)}{m(m+2)}p(X_0)^{-\frac{4}{m}}\left(\frac{m}{2\pi e}\right)^2(n+1)^{-4/m} \\
&\simeq l(l+2)p(X_0)^{-\frac{4}{m}}\left(\frac{1}{2\pi e}\right)^2(n+1)^{-4/m}
\end{aligned} \tag{B.3}
$$

Note that these equations are independent of $p(X_0)$. It is also noted that the expected mean and variance of r_l^2 are both linear in l (up to m).

We can derive the expected values of r_l^2 over X_0 for some special distributions $p(X_0)$.

For the uniform distribution on $[0,1]^m$, since $p(X_0)=1$, using $\alpha_{n,m}=\alpha_{n,m,2}$,

$$
\begin{aligned}
E_{X_0}E_u\{r_l^2|X_0\} &\simeq l\left(\frac{1}{2\pi e}\right)(n+1)^{-\frac{2}{m}} \simeq \frac{l}{2\pi e}\alpha_{n,m} \\
E_{X_0}E_u\{r_l^4|X_0\} &\simeq l(l+2)\left(\frac{1}{2\pi e}\right)^2(n+1)^{-4/m} \simeq \frac{l(l+2)}{4\pi^2e^2}\alpha_{n,m}^2 \\
V_{X_0,u}\{r_l^2\} &= E_{X_0}E_u\{r_l^4|X_0\} - (E_{X_0}E_u\{r_l^2|X_0\})^2 \simeq \frac{l}{2\pi^2e^2}\alpha_{n,m}^2
\end{aligned}
$$

For a normal distribution with mean zero and covariance matrix Σ,

$$
E_{X_0}\{p(X_0)^{\frac{s}{m}}\} = (2\pi)^{\frac{s}{2}}|\Sigma|^{\frac{s}{2m}}(1-\frac{s}{m})^{-\frac{m}{2}}.
$$

Hence, from (B.2) and (B.3),

$$
\begin{aligned}
E_{X_0}E_u\{r_l^2|X_0\} &\simeq 2\pi|\Sigma|^{\frac{1}{m}}(1-\frac{2}{m})^{-\frac{m}{2}}l\left(\frac{1}{2\pi e}\right)(n+1)^{-\frac{2}{m}} \\
&= \frac{l}{e}|\Sigma|^{\frac{1}{m}}(1-\frac{2}{m})^{-\frac{m}{2}}(n+1)^{-\frac{2}{m}} \\
&\left(\simeq l|\Sigma|^{\frac{1}{m}}(n+1)^{-\frac{2}{m}} = l|\Sigma|^{\frac{1}{m}}\alpha_{n,m}\right)
\end{aligned}
$$

$$
\begin{aligned}
E_{X_0}E_u\{r_l^4|X_0\} &\simeq (2\pi)^2|\Sigma|^{\frac{2}{m}}(1-\frac{4}{m})^{-\frac{m}{2}}l(l+2)\left(\frac{1}{2\pi e}\right)^2(n+1)^{-4/m} \\
&= \frac{l(l+2)}{e^2}|\Sigma|^{\frac{1}{m}}(1-\frac{4}{m})^{-\frac{m}{2}}(n+1)^{-4/m} \\
&\left(\simeq l(l+2)|\Sigma|^{\frac{1}{m}}(n+1)^{-\frac{2}{m}} = l(l+2)|\Sigma|^{\frac{1}{m}}\alpha_{n,m}^2\right)
\end{aligned}
$$

Thus, we have

$$
\begin{aligned}
V_{X_0,u}\{r_l^2\} &= E_{X_0}E_u\{r_l^4|X_0\} - (E_{X_0}E_u\{r_l^2|X_0\})^2 \\
&= \frac{1}{e^2}|\Sigma|^{\frac{2}{m}}(n+1)^{-\frac{4}{m}}\left(\frac{l(l+2)}{m(m+2)}(1-\frac{4}{m})^{-\frac{m}{2}} - \frac{l^2}{m^2}(1-\frac{2}{m})^{-m}\right) \\
&\simeq \frac{1}{e^2}|\Sigma|^{\frac{2}{m}}(n+1)^{-\frac{4}{m}}\left(\frac{l(l+2)}{m(m+2)}e^2(1+\frac{2}{m})^2 - \frac{l^2}{m^2}e^2(1+\frac{1}{m})^2\right) \\
&\simeq \frac{1}{e^2}|\Sigma|^{\frac{2}{m}}(n+1)^{-\frac{4}{m}}\left(\frac{l(l+2)}{m(m+2)}e^2(1+\frac{4}{m}) - \frac{l^2}{m^2}e^2(1+\frac{2}{m})\right) \\
&\simeq |\Sigma|^{\frac{2}{m}}(n+1)^{-\frac{4}{m}}\left(\frac{l(l+2)}{m(m+2)} - \frac{l^2}{m^2} + \frac{2l(l+1)}{m^3}\right) \\
&\simeq 2l|\Sigma|^{\frac{2}{m}}\alpha_{n,m}^2.
\end{aligned}
$$

References

[1] T.M. Cover and P.E. Hart, Nearest neighbor pattern classification, IEEE Trans. Information Theory, vol.IT-13, no.1, pp.21-27, Jan. 1967.

[2] C.-C. Chang and T.-C. Wu, A hashing-oriented nearest neighbor searching scheme. Pattern Recognition Letters, vol.14, pp. 625–630, August 1993.

[3] B. V. Dasarathy, Minimal consistent set(MCS) identification for optimal neighbor decision systems design, IEEE Trans. Systems, Man, & Cybernetics, vol.SMC-24, no.3, pp.511-517, March 1994.

[4] K. Fukunaga and P.M. Narendra, A branch-and-bound algorithm for computing k-nearest neighbors, IEEE Trans. Computers, vol.C-24, no.7, pp.750-753, July 1975.

[5] J. H. Friedman, F. Naskett and L. J. Shustek, An algorithm for finding nearest neighbors. IEEE Trans. Computer, pp. 1000–1006, October 1975.

[6] J. H. Friedman, J. L. Bentley and A. Finkel, An algorithm for finding best matches in logarithm expected time. ACM Trans. Math. Software, vol. 3, no. 3, pp. 209–226. 1977.

[7] G.W. Gates, The reduced nearest neighbor rule, IEEE Trans. Information Theory, vol.IT-18, no.3, pp.431-433, May 1972.

[8] P. E. Hart, The condensed nearest neighbor rule, IEEE Trans. Information Theory, vol.IT-14, no.3, pp.515-516, May 1968.

[9] A. Califano and R. Mohan, Multidimensional indexing for recognizing visual shapes. Proc. IEEE Conf. on Computer Vision and Pattern Recognition, pp. 28–34, June 1991.

[10] H. J. Wolfson, Model-based object recognition by geometric hashing. Proc. 1st European Conf. Computer Vision, pp. 526–536, April 1990.

[11] J. L. Bentley, Multidimensional binary search trees used for associative searching. Comm. of the ACM, vol. 18, no. 9, pp. 509–517, September 1975.

[12] J. L. Bentley, B. W. Weide and A. C. Yao, Optimal expected-time algorithms for closest-point problems. ACM Trans. Math. Software, vol. 6, no. 4, pp. 563–580, 1980.

[13] A. Guttman, R-trees: A dynamic index structure for spatial searching. ACM SIGMOD, pp. 47–57, June 1984.

[14] T. Sellis, N. Roussopoulos and C. Faloutos, The R^+-tree: A dynamic index for multi-dimensional objects. Proc. 13th VLDB Conference, pp. 507–517, 1987.

[15] S. Berchtold, D. A. Keim and H.-P. Kriegel, The X-tree: An index structure for high-dimensional data, Proc. 22nd VLDB Conference, pp. 28–39, 1996.

[16] A. Djouadi and E. Boutache, A fast algorithm for the nearest-neighbor classifier, IEEE Trans. Pattern Analysis and Machine Intelligence, vol.19, no.3, pp.277-282, March 1997.

[17] S. Berchtold, B. Ertl, D. A. Keim, H.-P. Kriegel and T. Seidl, Fast Nearest Neighbor Search in High-Dimensional Spaces, Proc. 14th IEEE Conf. Data Engineering, ICDE, pp. 23–27, 1998.

[18] 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, vol. 19, pp. 989-1003, 1997.

[19] D. Y. Cheng *et al.*, Fast search algorithms for vector quantization and pattern matching. Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, 1984, 372–375.

[20] J. McNames, A fast nearest-neighbor algorithm based on a principal axis search tree. IEEE Transactions on Pattern Analysis and Machine Intelligence, 23-9(2001), 964–976.

[21] S. Arya *et al.* An optimal algorithm for approximate nearest neighbor

searching fixed dimensions. Journal of the ACM, 45-6(1998), pp. 891–923. URL:http://www.cs.umd.edu/ mount/ANN/.

[22] J. M. Kleinberg, Two algorithms for nearest-neighbor search in high dimension, Proc. 29th Annu. ACM sympos. Theory Comput., pp.599–608, 1997.

[23] S. Maneewongvatana and D. M. Mount, An Empirical Study of a New Approach to Nearest Neighbor Searching. Lecture Notes in Computer Science, 2513 (2001), 172–187, Springer-Verlag.

[24] P. Ciaccia and M. Patella, PAC nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. Proceedings of the 16th International Conference on Data Engineering, 2000, 244–255.

[25] P. Indyk and R. Motwani, Approximate nearest neighbors: towards removing the curse of dimensionality. Proceedings of the 30th Annual ACM Symposium on Theory of Computing, pp. 604–613, 1998.

[26] A. Andoni, et al., Locality-Sensitive Hashing Using Stable Distributions (Chap. 3), in "Nearest-Neighbor Methods in Learning and Vision: Theory and Practice." G. Shakhnarovich, T. Darrell and P. Indyk (eds.), MIT Press, 2006.

[27] Alexandr Andoni and Piotr Indyk, Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions, Communications of the ACM, 51(1), pp.117–122, 2008.

[28] K. Fukunaga, Introduction to Statistical Pattern Recognition (2nd ed.), Academic Press, New York, 1990.

[29] Y. Le. Cunn, The mnist dataset of handwritten digits. Available at http://yann.lecun.com/exdb/mnist/.

[30] A. Baccini, P. Besse, A. Falguerolles, A L1-norm PCA and a heuristic approach. Proceedings of Ordinal and Symbolic Data Analysis. Springer, 359–368, 1996.

[31] N. Kwak, Principal Component Analysis Based on L1-norm Maximization. IEEE Transactions on Pattern Analysis and Machine Intelligence, 30-9(2008), 1672–1680.