The IOC algorithm: Efficient Many-Class Non-parametric Classification for High-Dimensional Data

Ting Liu
Computer Science Dept.
Carnegie Mellon University
Pittsburgh, PA 15213
tingliu@cs.cmu.edu

Ke Yang
Computer Science Dept.
Carnegie Mellon University
Pittsburgh, PA 15213
yangke@cs.cmu.edu

Andrew W. Moore
Computer Science Dept.
Carnegie Mellon University
Pittsburgh, PA 15213
awm@cs.cmu.edu

ABSTRACT

This paper is about a variant of k nearest neighbor classification on large many-class high dimensional datasets.

K nearest neighbor remains a popular classification technique, especially in areas such as computer vision, drug activity prediction and astrophysics. Furthermore, many more modern classifiers, such as kernel-based Bayes classifiers or the prediction phase of SVMs, require computational regimes similar to k-NN. We believe that tractable k-NN algorithms therefore continue to be important.

This paper relies on the insight that even with many classes, the task of finding the majority class among the k nearest neighbors of a query *need not require us to explicitly find those k nearest neighbors*. This insight was previously used in (Liu *et al.*, 2003) in two algorithms called KNS2 and KNS3 which dealt with fast classification in the case of two classes. In this paper we show how a different approach, IOC (standing for the International Olympic Committee) can apply to the case of n classes where $n > 2$.

IOC assumes a slightly different processing of the datapoints in the neighborhood of the query. This allows it to search a set of metric trees, one for each class. During the searches it is possible to quickly prune away classes that cannot possibly be the majority.

We give experimental results on datasets of up to 5.8×10^5 records and 1.5×10^3 attributes, frequently showing an order of magnitude acceleration compared with each of (i) conventional linear scan, (ii) a well-known independent SR-tree implementation of conventional k-NN and (iii) a highly optimized conventional k-NN metric tree search.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Design, Performance

Keywords

k nearest neighbor, classification, high dimension, metric tree

1. INTRODUCTION

KDD'04, August 22–25, 2004, Seattle, Washington, USA.

This paper is about new approaches to fast k-NN classification. Spatial data structures such as kd-trees [7, 17] and metric trees [21, 16, 5] have often been proposed and used for this computation, but the benefits of finding the k-NN using such data structures have often been observed to degrade substantially compared with linear search when the dimensionality of the data gets high [22]. This paper attempts to take advantage of one extra leverage point that *k-NN classification* has that the more general problem of k-NN does not have: all we need to do is to find the majority class of the k nearest neighbors—not the neighbors themselves.

During this paper, we show why it is hard to exploit this leverage point for the case of conventional k-NN. We therefore introduce a modified form of k-NN called IOC (standing for International Olympic Committee, explained later in the paper) that selects the predicted class by a kind of elimination tournament instead of a direct majority vote. Interestingly, this alternative voting scheme exhibits no general degradation in empirical performance, and we also prove that the asymptotic behavior of IOC must be very close to that of conventional k-NN.

IOC allows us to take advantage of the above leverage point, and we describe the new metric tree search algorithms that result. Our empirical results show excellent computational acceleration of IOC-with-metric-trees on real-world datasets compared with our own implementation (KNS1), and a well-known standard implementation of classical k-NN search (SR-tree). Our empirical results also show only slight predictive performance loss with typically an order of magnitude acceleration over classical k-NN for predictions on streams of queries.

2. METRIC TREES

The metric tree [21, 16, 5] is a data structure that supports efficient nearest neighbor search. We briefly describe the data structure and its usage.

First, some notation. Assume that the data are points in a d-dimensional Euclidean space. Let $\mathcal{T} = \{x_1, x_2, ..., x_n\}$ be the set of training data. Each x_i has a *label* $y_i \in \mathcal{L}$, where we define $m = |\mathcal{L}|$ to be the total number of classes. We use q to denote the *query point* on which the classification algorithm is to make a prediction.

For k-NN classification, one finds the k nearest neighbors of q from $\mathcal{T}$ — we denote them by $\mathcal{N} = \mathsf{kNN}(\mathsf{q}, k, \mathcal{T}) = \{w_1, w_2, ..., w_k\}$. Then k-NN labels q with the class that appears most frequently in $\mathcal{N}$. As a notational convention, we call the most frequent class the "winner" and all other classes the "losers." Thus, the k-NN classification amounts to finding the winner class and assigning it to the query point q.

We use $\mathbb{B}(x, r)$ to denote a ball centered at point x with radius r. In other words, we have $\mathbb{B}(x, r) = \{y : ||x - y|| \le r\}$.

A metric tree data structure organizes a set of points in a spatial hierarchical manner. It is a binary tree whose nodes represent a set of points. The root node represents all points, and the points represented by an internal node v are partitioned into two subsets, represented by the two children of $node$. More formally, if we use $N(\mathsf{v})$ to denote the set of points represented by node v, and use v.lc and v.rc to denote the left child and the right child of node v, then we have

$$\begin{aligned} N(\mathsf{v}) &= N(\mathsf{v.lc}) \cup N(\mathsf{v.rc}) \quad (1)\\ \emptyset &= N(\mathsf{v.lc}) \cap N(\mathsf{v.rc}) \quad (2) \end{aligned}$$

for all the non-leaf nodes. At the lowest level, each leaf node contains very few points (typically from 1 to 5).

Each node v contains a *pivot* point, denoted by v.pvt, and a *radius*, denoted by v.r, such that all points represented by v fall in the ball centered at v.pvt with radius v.r. Mathematically, we have

$$N(\mathsf{v}) \subseteq \mathbb{B}(\mathsf{v.pvt}, \mathsf{v.r}). \quad (3)$$

We stress that although $N(\mathsf{v.lc})$ and $N(\mathsf{v.rc})$ are disjoint, the balls representing these two pointsets are not necessarily so. Furthermore, v.r is chosen to be the minimal value satisfying (3). As a consequence, we know that the radius of any node is always strictly greater than the radius of its child nodes. The leaf nodes have very small radii.

The pivot points also serve as the criterion for partitioning the nodes: every point x in $N(\mathsf{v})$ is assigned to one child of v whose pivot point is closer to x. More specifically, for all non-leaf nodes v, we have

$$\begin{aligned} x \in N(\mathsf{v.lc}) &\iff ||x - \mathsf{v.lc.pvt}|| \leq ||x - \mathsf{v.rc.pvt}|| \quad (4)\\ x \in N(\mathsf{v.rc}) &\iff ||x - \mathsf{v.lc.pvt}|| > ||x - \mathsf{v.rc.pvt}|| \quad (5) \end{aligned}$$

We do not discuss the construction of metric trees except that a metric tree can be constructed from the points efficiently, for example, using methods from [21, 5, 15].

We focus our attention to nearest neighbor (NN) search using metric trees. Intuitively, metric trees can speed up the search by using the triangle inequality. Given any query point q and an arbitrary $x \in N(\mathsf{v})$, we know that

$$||\mathsf{q} - \mathsf{v.pvt}|| - \mathsf{v.r} \leq ||\mathsf{q} - x|| \leq ||\mathsf{q} - \mathsf{v.pvt}|| + \mathsf{v.r}. \quad (6)$$

Therefore, by doing only one distance computation (namely, the distance between q and v.pvt), we can bound the distance between q and any point in $N(\mathsf{v})$, both from above and from below. This information can help us estimate the number of points that are at most distance t from q, as well as the number of points that are at least distance t' away from q. In many cases, this insight can help prune away many nodes in the k-NN search.

3. THE IOC ALGORITHM

In this section we discuss the IOC algorithm for approximating the k-NN classification for the many-class setting. We first describe the problem with existing solutions using metric trees.

3.1 Previous solutions and their problems

A naïve implementation of the standard k-NN algorithm finds the exact k nearest neighbors using linear search, which needs N invocations of distance computation. When the dimension d of the data is large, these distance computations have time complexity $\Omega(dN)$, which can be unrealistically expensive. Classical solutions such as metric trees can be used to speed up the search, but this technique does not scale well to high dimensions.

Liu *et al.* [14] proposed several techniques to speed up the k-NN binary classification problem. Their techniques rely on the insight that in k-NN classification, one does not need to find the actual k nearest neighbors. Rather, it is often sufficient to answer simpler, counting-related problems. Examples of these questions are: 1) "How many points of class i are in the k nearest neighbors of q?" and 2) "Does class i contains at most ℓ points in the k nearest neighbors of q?" As demonstrated in [14], these questions can often be answered much more efficiently.

To illustrate this point more clearly, we introduce a new concept, namely the "threshold nearest neighbor" function.

DEFINITION 1. *The* threshold nearest neighbor *function, denoted by* tNN*, is defined as follows.*

$$\mathsf{tNN}(\mathsf{q}, \mathcal{T}, T, k, \ell) := \begin{cases} 1 & \text{if } |T \cap \mathsf{kNN}(\mathsf{q}, k, \mathcal{T})| \leq \ell \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

Intuitively, $\mathsf{tNN}(\mathsf{q}, \mathcal{T}, T, k, \ell)$ checks whether of the k nearest neighbors of q in $\mathcal{T}$, the subset T contains at most ℓ points.

Roughly speaking, the evaluation of $\mathsf{tNN}(\mathsf{q}, \mathcal{T}, T, k, \ell)$ is done by finding a "threshold bound" t, such that either

1. $\mathbb{B}(\mathsf{q}, t)$ contains at most ℓ points in T and at least $(k - \ell)$ points in $\mathcal{T} \setminus T$, or
2. $\mathbb{B}(\mathsf{q}, t)$ contains more than ℓ points in T and less than $(k-\ell)$ points in $\mathcal{T} \setminus T$.

In the first case, we have $\mathsf{tNN}(\mathsf{q}, \mathcal{T}, T, k, \ell) = 1$; in the second case, we have $\mathsf{tNN}(\mathsf{q}, \mathcal{T}, T, k, \ell) = 0$. In Section 3.4, we review the evaluation of tNN in more details.

Unfortunately, the insight in [14] does not work in the many-class case. Consider a query point q and its k nearest neighbor set $\mathcal{N}$. For binary classification, q is classified as class i, if and only if $\mathcal{N}$ contains at least $\lceil k/2 \rceil$ points of class i (assuming that k is odd). Thus the task of finding the winner is reduced to a counting problem, or more specifically, evaluating the function $\mathsf{tNN}(\mathsf{q}, \mathcal{T}, T_i, k, \lfloor k/2 \rfloor)$. In the case of many-classes, the situation is very different. We no longer have a fixed threshold that allows us to reduce the search-for-winner problem to a counting problem. We know that for a class to be the winner, it must necessarily contain more than $\lfloor k/m \rfloor$ points in $\mathcal{N}$ (assuming that k is not a multiple of m), and it is sufficient that it contains at least $\lceil k/2 \rceil$ points. However, for numbers between $\lfloor k/m \rfloor$ and $\lceil k/2 \rceil$, we cannot prove anything. Therefore, we cannot reduce the k-NN search problem to a simple counting problem. This is the reason why the techniques in [14] do not extend to the many-class case.

3.2 IOC: high-level descriptions

The IOC algorithm is a variant of the k-NN algorithm that allows speed-up using metric trees. The motivation behind IOC is to modify k-NN in such a way that it can be reduced to a *sequence* of counting problems. One important observation is that despite the fact that the necessary condition and the sufficient condition combined cannot determine if an *arbitrary* class is the winner in general, one can always use the necessary condition to find *some* class that is not a winner. This is simply because that by the pigeonhole principle, there exists at least one class containing at most $\lfloor k/m \rfloor$ points, and this class is not the winner.

This algorithm is inspired by the procedure used by the International Olympic Committee [11] to select the host city for summer Olympic games (which also explains its name). In the procedure, instead of having a single round of ballots and selecting the favorite city as the winner (which would correspond to the "standard" k-NN algorithm), multiple rounds of ballots are cast. In each round, if a city gets a majority of the votes, then it is declared the winner and the procedure finishes. Otherwise, the city that gets the fewest votes is eliminated and a new round of ballots is cast. This continues until only one city is left, and this city is declared the winner.

We now describe the IOC algorithm at a high level. IOC starts by building a metric tree for each class respectively, and then proceeds in *rounds*. In each round, either a winner is selected, or some losers are eliminated. More precisely, in each round, if a class i contains at least $\lceil k/2 \rceil$ points in the k nearest neighbors of q, which can be answered by evaluating the threshold nearest neighbor function $\mathsf{tNN}(\mathsf{q}, \mathcal{T}, T_i, k, \lfloor k/2 \rfloor)$, then this class is declared a winner and the algorithm terminates, labeling q with class i. Otherwise, the algorithm finds all the classes that contains at most $\lfloor k/m \rfloor$ points in the k nearest neighbors of q, and declare these classes the losers, All the "loser" classes will be removed from consideration. The number of classes, m, is reduced accordingly. This process continues until a winner is selected or there is only one class remaining, in which case the only remaining class is declared a winner.

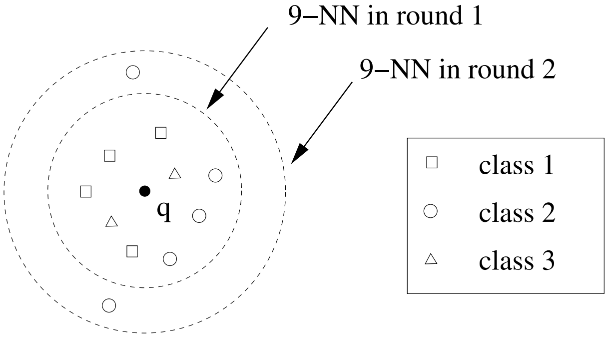

Figure 1: different predictions by IOC and k-NN.

We notice that the IOC algorithm does not always behave identically to the standard k-NN algorithms, and in particular, the prediction made by the IOC algorithm may differ from that by the standard k-NN. As an example shown in Figure 1, there are 3 classes and $k = 9$. The 9 nearest neighbors of the query point q contain 4 points of class 1, 3 points of class 2, and 2 points of class 3. Therefore, standard k-NN algorithm would select class 1 as the winner. However, in the IOC algorithm, class 3 would be identified as a loser and removed in the first round. In the second round, the 9 nearest neighbors of q includes two additional points of class 2. Now we have 4 points of class 1 and 5 points of class 2 in this round, and IOC will choose class 2 as the winner.

Incidentally, a similar example occurred in the procedure for picking the host city for the 2000 Olympics game by IOC. The process proceeded in multiple rounds, and Beijing was the favorite city in all but the last round, but never won more than half of the votes. In the last round, Beijing lost to Sydney, and the IOC chose Sydney as the winner. If the standard k-NN algorithm had been used, Beijing would have been chosen.

3.3 The Threshold Nearest Neighbor Function

Recall that the idea of IOC hinges on the ability to evaluate the threshold nearest neighbor function tNN efficiently. Therefore, we first describe an algorithm that evaluates tNN using metric trees. The algorithm, denoted by MTtNN thereafter, is adapted from [14].

To begin with, MTtNN builds one metric tree for T_i, the set of training points of class i. Then, to evaluate function $\mathsf{tNN}(\mathsf{q}, \mathcal{T}, T_i, k, \ell)$, MTtNN needs to:

1. Find an appropriate threshold t, and
2. Prove that either:
   (a) $\mathbb{B}(\mathsf{q}, t)$ contains at most ℓ points in T_i and at least $(k - \ell)$ points in $\mathcal{T} \backslash T_i$ (so that $\mathsf{tNN}(\mathsf{q}, \mathcal{T}, T_i, k, \ell) = 1$), or
   (b) $\mathbb{B}(\mathsf{q}, t)$ contains more than ℓ points in T_i and less than $(k - \ell)$ points in $\mathcal{T} \backslash T_i$ (so that $\mathsf{tNN}(\mathsf{q}, \mathcal{T}, T_i, k, \ell) = 0$).

First, let us assume that t is known. We see how one can prove statement (2.a) or (2.b) using the metric trees. Consider a node v in the metric tree for class j. Suppose v represents s points, and the distance between $\mathsf{v.pvt}$ and q is x. By the triangle inequality, we know that if $t < x - \mathsf{v.r}$, then none of the s points in $N(\mathsf{v})$ is in $\mathbb{B}(\mathsf{q}, t)$; if $t > x + \mathsf{v.r}$, then all the s points are in $\mathbb{B}(\mathsf{q}, t)$. In both cases, node v contributes information about the number of points in $\mathbb{B}(\mathsf{q}, t)$ and we say v is "useful." However, if $t \in [x - \mathsf{v.r}, x + \mathsf{v.r}]$, node v does not tell us anything, and we say node v is "useless." Then MTtNN sums up all the information from the useful nodes and checks if this information can be used to prove (2.a) or (2.b).

In case the information is insufficient, MTtNN selects a useless node v to *split*, i.e., to replace node v by its two children $\mathsf{v.lc}$ and $\mathsf{v.rc}$. Since child nodes have smaller radii, they provide more "refined" information that might be useful. Ultimately, the leaf nodes provide very accurate information since they have very small radii.[1] However, splitting a node is an expensive operation, as one needs to compute the distance between q and the pivots of the children nodes, and distance computations are the dominant operations in terms of time complexity. Therefore, to achieve optimal efficiency, one needs to minimize the number of splits.

Next, if we drop the assumption that t is known, MTtNN needs to search for t as well. To do so, it maintains a list of "known" nodes from the metric trees, i.e., the nodes where the distance between q and their pivots are computed and known, and searches for an appropriate t. If no such t is found due to insufficient information, the algorithm selects a node to split according to a certain splitting policy and tries again. As demonstrated in [14], with a carefully designed policy, one can indeed minimize the number of splits and make the algorithm very efficient.

3.4 The IOC Algorithm

With an efficient implementation of the tNN function, we can implement the IOC algorithm directly, as in Section 3.2. However, this is not very efficient, since MTtNN may need to do a lot of splits in order to find the answer. In fact, observe that in each round, many instances of the tNN functions are evaluated — for each class i, we need to evaluate both $\mathsf{tNN}(\mathsf{q}, \mathcal{T}, T_i, k, \lfloor k/2 \rfloor)$ and $\mathsf{tNN}(\mathsf{q}, \mathcal{T}, T_i, k, \lfloor k/m \rfloor)$. We can make progress whenever we find one winner or one loser. This observation allows us to improve the efficiency by *dove-tailing*, i.e., evaluating all the tNN functions simultaneously, and terminates whenever a winner or a loser is found. More precisely, we modify the MTtNN algorithm so that it may also output $\perp$, standing for "unknown." Then we only do a split when all evaluations return $\perp$.

The algorithm, IOC, is described in Figure 2. Here MTtNN' is the revised version of MTtNN that partially computes tNN. In other words, MTtNN' may return $\perp$ when it does not have sufficient information, but it never splits any node. The splitting of the trees is handled by the procedure $\mathsf{do\_split}$, which picks a particular class i and performs one split on the metric tree of T_i. Effectively, the IOC algorithm minimizes the number of splits by aggressively attempting to evaluate all the tNN functions after each split.

We emphasize that the IOC algorithm is presented in a way to maximize clarity. In particular, we omit all optimizations, some of

[1] As a matter of fact, it is often the case that a leaf node contains a single point, in which case a leaf node has radius 0.

Procedure IOC(k, q, $\mathcal{T}, T_1, T_2, ..., T_m$)
begin
  $A \leftarrow \{1, 2, ..., m\}$
  begin repeat
  /* partially evaluate the tNN functions */
    foreach $i \in A$ **do**
      $x_i \leftarrow \mathsf{MTtNN}'(q, \mathcal{T}, T_i, k, \lfloor k/2 \rfloor)$
      $y_i \leftarrow \mathsf{MTtNN}'(q, \mathcal{T}, T_i, k, \lfloor k/m \rfloor)$
    end foreach
    progress $\leftarrow 0$
    /* check for winners and losers */
    if $\exists i \in A$, s.t., $x_i = 0$, **then return** i
    else $\forall i \in A$,
      if $y_i = 1$ **then**
        /* found a loser, remove it */
        $A \leftarrow A \backslash \{i\}, \mathcal{T} \leftarrow \mathcal{T} \backslash T_i, m \leftarrow m - 1;$
        progress $\leftarrow 1;$
      end if
      /* terminate if only one class remaining */
      if $m = 1$ and $A = \{i\}$, **then return** i
    end if
    /* need to split if no winner/loser is found */
    if progress $= 0$ **then** do_split($\mathcal{T}, T_1, T_2, ..., T_m$)
  end repeat
end

Figure 2: The IOC algorithm.

which are obvious. For example, after splitting class j, one only needs to update the information related to class j and there is no need to re-compute all x_i and y_i for all i's. Additionally, many invocations of the MTtNN can be merged to improve efficiency. Furthermore, some other techniques are used in the algorithm to ensure its robustness. Due to space limitation, we do not discuss them in this extended abstract.

3.5 Theoretical analysis

We analyze the behavior of the IOC algorithm from the theoretical perspective. Due to space limitation, the proofs are omitted.

Theorem 1 IOC behaves identically to the standard k-NN algorithm when $k = 1$ and when $m = 2$.

Theorem 2 If class i is not chosen by the IOC algorithm as the winner, then there exists a k' such that class i is not the majority class in $\mathsf{kNN}(\mathsf{q}, k', \mathcal{T})$.

Theorem 3 Let q be the query point, m be the number of classes, n be the size of training set. Let $p_1 \geq p_2 \geq \cdots \geq p_m$ be the Bayes conditional probability for class i at q, and let $\delta = p_1 - p_2$. Then, with probability at least $1 - \epsilon$, the behavior of the IOC algorithm with

$$k \geq \frac{12}{\delta^2} \log \left(\frac{2m}{\epsilon} \right)$$

is identical to the behavior of k-NN as $n \to \infty$.

Remarks Theorem 1 establishes the fact that IOC and k-NN are identical in many cases. Theorem 2 indicates that even in the many-class case where IOC and k-NN differ, the difference isn't significant: if a class is not chosen as the winner by IOC, then it will be a loser in k'-NN, for a properly chosen k'. Theorem 3 implies that if k is large enough, then the asymptotic behavior of IOC and k-NN are identical with very high probability, and in particular, both are approximations of the optimal Bayes prediction.

4. EXPERIMENTAL RESULTS

In this section, we tested the IOC algorithm on both artificial and real-world datasets and compared the results with three other algorithms:

1. Naïve: a conventional linear-scan k-NN algorithm.
2. SR-tree: an implementation by Katayama and Satoh [12].
3. KNS1: an optimized k-NN search based on metric trees [21].

We estimate two performance measures:

Speed This is the primary concern of this paper. We consider accelerations both in terms of number of distance computations and CPU time. For all the experiments below, we first show the computational cost of naïve k-NN. We then examine the speed-ups of SR-tree, KNS1 and IOC. (Notice that for SR-tree, we omit the distance computations speedup, since the SR-tree implementation does not report this measure.)

Accuracy We compare the (empirical) classification accuracy between k-NN and IOC. We emphasize that since our goal is to accelerate multi-class classification in high dimensions, we do not try to improve accuracy (though we should expect no decline). We consider it acceptable to have both k-NN and IOC perform badly on some datasets as long as their performance is comparable.

We tested our algorithm on a variety of real world datasets (listed in Table 1) with multi-class classification tasks. The datasets are all publicly available.

Table 1: The datasets.

Dataset	Train size	Test size	Num. Dimensions	Num. classes
Letter	16000	4000	16	26
Isolet	6238	1555	617	26
CovType	58101	522911	54	7
Video	35049	3894	62	3
Internet_ads	2952	327	1555	2

1. *Letter* (Letter Recognition Database [20]) It is from the UCI Machine Learning repository, containing 20,000 instances with 26 classes. Each instance represents a bitmap image of a character as one of the 26 Roman letters. The objective is to identify the letter category from the images.
2. *Isolet* (Isolet Spoken Letter Recognition Database [6]) It contains 6238+1559 instances with 26 classes. The dataset was derived from 150 people speaking the name of each letter of the alphabet twice (3 examples are missing). Each instance has 617 attributes. The goal is to predict which letter is spoken.
3. *CovType* (Forest CoverType Database) If is originally from UCI/KDD Archive. The dataset contains 581012 datapoints with 7 classes. See [4].
4. *Video* (TREC-2001 Video Dataset [10]) It contains 5.8 hours of MPEG-1 video files. The task is to detect the shot boundaries within the video files. The corpus contain 2 types of transition frames: cuts and gradual transitions, so we can see this problem as a 3 class classification problem: no transition, cut and gradual transition. After preprocessing, the final dataset contains 38,943 frames, each frame has 62 attributes [18].
5. *Internet_ads* (Internet Advertisements [13]) This dataset represents a set of possible advertisements on Internet pages. The task is to predict whether an image is an advertisement ("ad") or not ("non-ad"). After we remove the three continuous attributes, the final dataset contains 3,279 instances, and 1,555 attributes for each instance.

For each dataset, we manually partitioned them into a training set and a test set, and we ran our experiments with $k = 1$, 5, and 9.

For all algorithms, we report the pre-processing time and the error rates (see Table 2), as well as the average prediction time per query (see Table 3) We also plot the speed-up of various algorithms over naïve k-NN (CPU time) for the case $k = 5$ in Figure 3. Furthermore, we report how the CPU time of various algorithms scales with the size of the training data (see Figure 4 for the case $k = 1$ and $k = 9$).

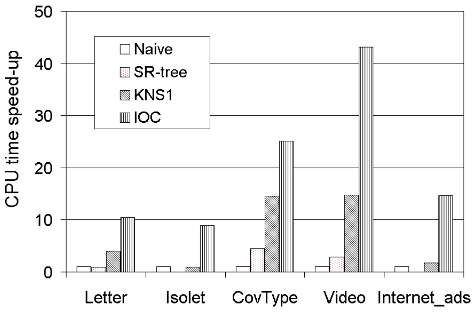

Figure 3: CPU time speed up over naïve k-NN ($k = 5$).

Table 2: Pre-processing time and error rates [time(s) : error].

Dataset	k	Naïve	SR-tree	KNS1	IOC
Letter	1	0 : 0.043	54 : 0.043	0.46 : 0.043	6.7 : 0.112
	5	0 : 0.054	54 : 0.054	0.46 : 0.054	6.7 : 0.088
	9	0 : 0.056	54 : 0.056	0.46 : 0.056	6.7 : 0.077
Isolet	1	0 : 0.11	n/a : n/a	4.43 : 0.11	68 : 0.12
	5	0 : 0.077	— : —	4.4 : 0.077	69 : 0.085
	9	0 : 0.08	— : —	4.4 : 0.08	69 : 0.08
CovType	1	0 : 0.14	311 : 0.14	5.1 : 0.14	101 : 0.12
	5	0 : 0.17	311 : 0.17	5.1 : 0.17	101 : 0.17
	9	0 : 0.18	311 : 0.18	5.1 : 0.18	101 : 0.17
Video	1	0 : 0.15	240 : 0.15	3.5 : 0.15	52 : 0.16
	5	0 : 0.13	240 : 0.13	3.5 : 0.13	52 : 0.13
	9	0 : 0.13	240 : 0.13	3.5 : 0.13	52 : 0.13
Internet	1	0 : 0.040	n/a : n/a	6.4 : 0.040	80 : 0.049
	5	0 : 0.052	— : —	6.4 : 0.052	80 : 0.052
	9	0 : 0.062	— : —	6.4 : 0.062	80 : 0.064

Error rate The error rate of Naïve k-NN, SR-tree, and KNS1 are the same, since they are all exact k-NN algorithms. For the IOC algorithm, the error rate is slightly different. For *Letter* with k=1, the accuracy for IOC is worse than k-NN, while for all the other datasets and other settings of k, the error rates are comparable. In some cases IOC has even better accuracy than k-NN. This validates our claim that both k-NN and IOC are approximate versions of the optimal Bayes prediction, and none generally outperforms the other.

Speed-up The SR-tree algorithm typically does not show a significant speedup compared with the naïve linear scan in these datasets. We emphasize that the IOC algorithm maintains very robust speed-up in the face of high-dimensional data. Consider the datasets *Isolet* (617 dimensions) and *Internet_ads* (1,555 dimensions). The SR-tree implementation is unable to run on these two sets, and the KNS1 algorithm, which is highly optimized using metric trees, ach-ieves very mediocre speed-up, and sometimes we even observe a slow-down. Nevertheless IOC consistently exhibits from 10-fold to 50-fold speed-up. The dataset *Internet_ads* is particularly interesting and merits special mention. Notice that it is a two-class dataset, and techniques from [14] can be directly applied here. In particular, one can use the KNS3 algorithm in [14] to speed up the k-NN prediction. However, our IOC algorithm, which is designed for many-class prediction, shows about 2-fold speed-up over KNS3, and has about the same accuracy. This fact suggests that the pre-pruning technique used by IOC might have much wider applicability.

Table 3: Number of distance computations and CPU time. The Naïve column shows [number : time(s)], all other columns show speedup over Naïve.

Dataset	k	Naïve	SR-tree	KNS1	IOC
Letter	1	16000 : 27	n/a : 1.3	14 : 6.5	47 : 16
	5	— : —	n/a : 0.88	8.9 : 3.4	38 : 10
	9	— : —	n/a : 0.79	7.4 : 3.3	32 : 7
Isolet	1	6238 : 112	n/a : n/a	1.1 : 1.1	20 : 17
	5	— : —	— : —	0.98 : 0.93	11 : 9
	9	— : —	— : —	0.92 : 0.88	8.8 : 6.3
CovType	1	58101 : 40776	n/a : 7.3	36 : 26	79 : 38
	5	— : —	n/a : 4.46	20 : 15	34 : 25
	9	— : —	n/a : 3.62	16 : 12	42 : 14
Video	1	35049 : 177	n/a : 3.7	28 : 20	555 : 664
	5	— : —	n/a : 2.9	21 : 15	96 : 43
	9	— : —	n/a : 2.6	19 : 13	70 : 30
Internet	1	2952 : 28	n/a : n/a	2.4 : 2.4	49 : 58
	5	— : —	— : —	1.7 : 1.7	15 : 15
	9	— : —	— : —	1.5 : 1.4	10 : 9.8

Scalability We performed the simulations for scaling over dataset *Video*. We fixed 3500 points as a test set and trained on 5 training sets with sizes 7000, 14000, 21000 and 28000. To achieve better understanding of the scalability of our algorithms, we ran the experiments for both $k = 1$ and $k = 9$. The results are presented on Figure 4. Notice that IOC scales much better than all other algorithms.

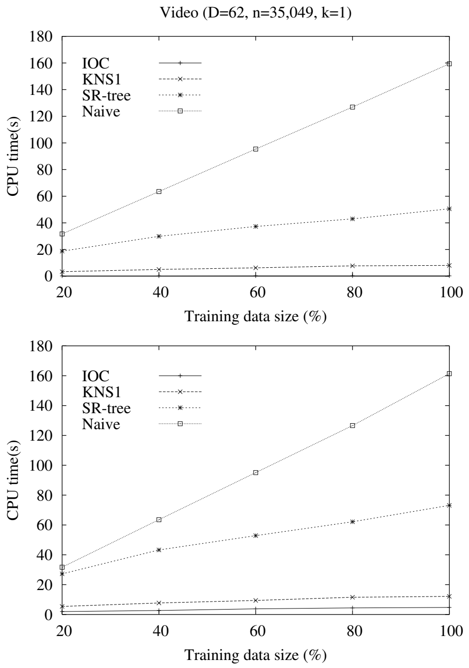

Figure 4: CPU time vs. data size (Video, $k = 1$ and $k = 9$).

5. RELATED WORK

For the problem of finding the k nearest datapoints (as opposed to k-NN classification) in high dimensions, the frequent failure of traditional metric trees to beat naïve has lead to some very ingenious and innovative alternatives, based on random projections, hashing discretized cubes, and acceptance of approximate answers. For example, Gionis *et al.* [8] gives a hashing method that was demonstrated to provide speedups over a SR-tree-based approach in 64 dimensions by a factor of 2–10 depending on how much error in the approximate answer was permitted. Another approximate k-NN idea is in [3], one of the first k-NN approaches to use a priority queue of nodes, in this case achieving a 3-fold speedup with an approximation to the true k-NN. However, these approaches are based on the notion that any points falling within a factor of $(1+\epsilon)$ times the true nearest neighbor distance are acceptable substitutes for the true nearest neighbor. Noting in particular that distances in high-dimensional spaces tend to occupy a decreasing range of continuous values [9], it remains unclear whether schemes based upon the absolute values of the distances rather than their *ranks* are relevant to the classification task (indeed, in the extreme of uniform data in very high dimensions, a randomly chosen data point would be expected to lie in the $(1+\epsilon)$ ball[2]). In contrast, the IOC algorithm finds an exact answer, though to a modified version of k-NN classification. Although both approaches have theoretical underpinnings, an important piece of future work will be a thorough empirical comparison (which was beyond the scope of the current paper).

Another solution to the cost of k-NN-type queries is *editing* (or *prototypes*): most training points are forgotten and only particularly representative ones are used (e.g. [2, 1, 19]). Kibler and Aha extended this further by allowing datapoints to represent local consensuses of sets of previously observed datapoints. This can be effective, but requires in advance the choice of the right degree of "smoothing" of the data (i.e. choosing the number of points to be included in the consensus). KNS1 (traditional metric tree kNN search) and IOC can both adaptively be called with varying values of k. A more serious problem with prototypes is that inevitably, very local predictions have detail lost and eventually (as the amount of data increases) the very advantage of non-parametric classifiers (their ability to adapt to local data) is lost if the number of retained datapoints remains fixed.

6. REFERENCES

[1] D. W. Aha. A Study of Instance-Based Algorithms for Supervised Learning Tasks: Mathematical, Empirical and Psychological Evaluations. PhD. Thesis; Technical Report No. 90-42, University of California, Irvine, November 1990.

[2] D. W. Aha, D. Kibler, and M. K. Albert. Instance-Based Learning Algorithms. *Machine Learning*, 6:37–66, 1991.

[3] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[4] Jock A. Blackard. Forest covertype database. `http://kdd.ics.uci.edu/databases/covertype/covertype.data.html`.

[5] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd VLDB International Conference*, September 1997.

[2] This was illustrated by Jonathan Goldstein in a presentation at the NIPS 2003 workshops.

[6] Ron Cole and Mark Fanty. Isolet spoken letter recognition database. `ftp://ftp.ics.uci.edu/pub/machine-learning-databases/isolet/`.

[7] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

[8] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. In *Proc 25th VLDB Conference*, 1999.

[9] J. M. Hammersley. The Distribution of Distances in a Hypersphere. *Annals of Mathematical Statistics*, 21:447–452, 1950.

[10] CMU informedia digital video library project. The trec-2001 video trackorganized by nist shot boundary task. 2001.

[11] IOC. International olympic committee: Candidature acceptance procedure. `http://multimedia.olympic.org/pdf/en_report_711.pdf`, 1999.

[12] Norio Katayama and Shin'ichi Satoh. The SR-tree: an index structure for high-dimensional nearest neighbor queries. pages 369–380, 1997.

[13] Nicholas Kushmerick. Internet advertisements. `ftp://ftp.ics.uci.edu/pub/machine-learning-databases/internet_ads/`.

[14] Ting Liu, Andrew Moore, and Alexander Gray. Efficient exact k-nn and nonparametric classification in high dimensions. In *Proceedings of Neural Information Processing Systems*, 2003.

[15] A. W. Moore. The Anchors Hierarchy: Using the Triangle Inequality to Survive High-Dimensional Data. In *Twelfth Conference on Uncertainty in Artificial Intelligence*. AAAI Press, 2000.

[16] S. M. Omohundro. Bumptrees for Efficient Function, Constraint, and Classification Learning. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*. Morgan Kaufmann, 1991.

[17] F. P. Preparata and M. Shamos. *Computational Geometry*. Springer-Verlag, 1985.

[18] Yanjun Qi, Alexander G. Hauptmann, and Ting Liu. Supervised classification for video shot segmentation. In *proceedings of 2003 IEEE International Conference on Multimedia & Expo*, 2003.

[19] D. B. Skalak. Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms. In W. W. Cohen and H. Hirsh, editors, *Machine Learning: Proceedings of the Eleventh International Conference*. Morgan Kaufmann, 1994.

[20] David J. Slate. Letter recognition database. `ftp://ftp.ics.uci.edu/pub/machine-learning-databases/letter-recognition/`.

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

[22] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 194–205, 24–27 1998.