Journal of Machine Learning Research 25 (2024) 1-51 Submitted 3/23; Revised 12/23; Published 2/24

A Multilabel Classification Framework for Approximate Nearest Neighbor Search

Ville Hyvönen VILLE.2.HYVONEN@AALTO.FI
Department of Computer Science
Aalto University

Elias Jääsaari EJAEAESA@ANDREW.CMU.EDU
Machine Learning Department
Carnegie Mellon University

Teemu Roos TEEMU.ROOS@HELSINKI.FI
Department of Computer Science
University of Helsinki

Editor: Sanjiv Kumar

Abstract

To learn partition-based index structures for approximate nearest neighbor (ANN) search, both supervised and unsupervised machine learning algorithms have been used. Existing supervised algorithms select all the points that belong to the same partition element as the query point as nearest neighbor candidates. Consequently, they formulate the learning task as finding a partition in which the nearest neighbors of a query point belong to the same partition element with it as often as possible. In contrast, we formulate the candidate set selection in ANN search directly as a multilabel classification problem where the labels correspond to the nearest neighbors of the query point. In the proposed framework, partition-based index structures are interpreted as partitioning classifiers for solving this classification problem. Empirical results suggest that, when combined with any partitioning strategy, the natural classifier based on the proposed framework leads to a strictly improved performance compared to the earlier candidate set selection methods. We also prove a sufficient condition for the consistency of a partitioning classifier for ANN search, and illustrate the result by verifying this condition for chronological k-d trees and (both dense and sparse) random projection trees.

Keywords: approximate nearest neighbor search, multilabel classification, partitioning models, statistical learning theory

1. Introduction

Approximate nearest neighbor (ANN) search is a fundamental algorithmic problem. The task is to build an index structure that enables finding the approximate k nearest neighbors of a *query point* from the set of *corpus points* in sub-linear time. There is a large body of literature on ANN search spanning several research communities, including the machine learning community. Specifically, space-partitioning index structures—such as space-partitioning trees (Friedman et al., 1976; Muja and Lowe, 2014; Dasgupta and Sinha, 2015) and data-dependent hash tables (Indyk and Motwani, 1998; Datar et al., 2004; Weiss et al., 2009)—are machine learning methods commonly used for ANN search. These partition-

based ANN methods use a partition (or a collection of partitions) of the feature space to select a smaller *candidate set* from the corpus points. They then return the k nearest neighbors of the query point among the points of the candidate set as the approximate nearest neighbors.

In this article[1] we propose a theoretical framework for ANN search. In particular, we formulate candidate set selection directly as a multilabel classification problem where the labels represent the indices of the nearest neighbors of the query point. The proposed framework has two immediate practical implications. First, it suggests that the performance of space-partitioning index structures can be improved by using them in a theoretically justified fashion as *partitioning classifiers* (see Devroye et al., 1996, Chapter 21), instead of applying the earlier candidate set selection methods (see Fig. 1 for an illustration of the difference between our approach and the earlier candidate set selection methods). Second, the proposed framework enables applying a general purpose classifier directly as an index structure for ANN search; we demonstrate this by using a multilabel random forest as an index structure.

We begin by reviewing the relevant background on ANN search and multilabel classification (Sec. 2). We then formulate the candidate set selection in ANN search as a multilabel classification task (Sec. 3.1), characterize the optimal decision boundaries for this classification problem (Sec. 3.2), and define the natural (partitioning) classifier for the general multilabel classification task (Sec. 4). In Sec. 5, we show how the natural classifier defined in Sec.4 can be used as a candidate set selection for ANN search. In Sec. 6, we review the earlier supervised partitioning methods for ANN search and elaborate the difference between the proposed framework and the earlier supervised methods. The proposed multilabel formulation also enables considering asymptotics in the standard statistical learning framework: we establish a sufficient condition for the consistency of a partitioning classifier for ANN search (Sec. 7.1). We then verify this condition for the chronological k-d tree (Bentley, 1975), the random projection tree (Dasgupta and Freund, 2008; Dasgupta and Sinha, 2015), and the sparse variant of random projection tree (Hyvönen et al., 2016); see Secs. 7.2, 7.3.1, and 7.3.2, respectively. To empirically validate the proposed framework, we show that the natural classifier suggested by it leads to a strictly improved performance compared to the earlier candidate set selection methods when combined with three types of space-partitioning trees (Sec. 8).

2. Background and notation

In this section, we review the conventional formulations of ANN search and multilabel classification.

2.1 Approximate nearest neighbor search

Let the *corpus* $\{c_j\}_{j=1}^m$ be a set of vectors in $\mathbb{R}^d$. Any metric, or, more generally, any dissimilarity measure, can be used to define the nearest neighbors of a *query point* $x \in \mathbb{R}^d$ among the set of corpus points. In this article, we consider nearest neighbors defined by

1. This article extends our earlier work (Hyvönen et al., 2022) published in the Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS 2022).

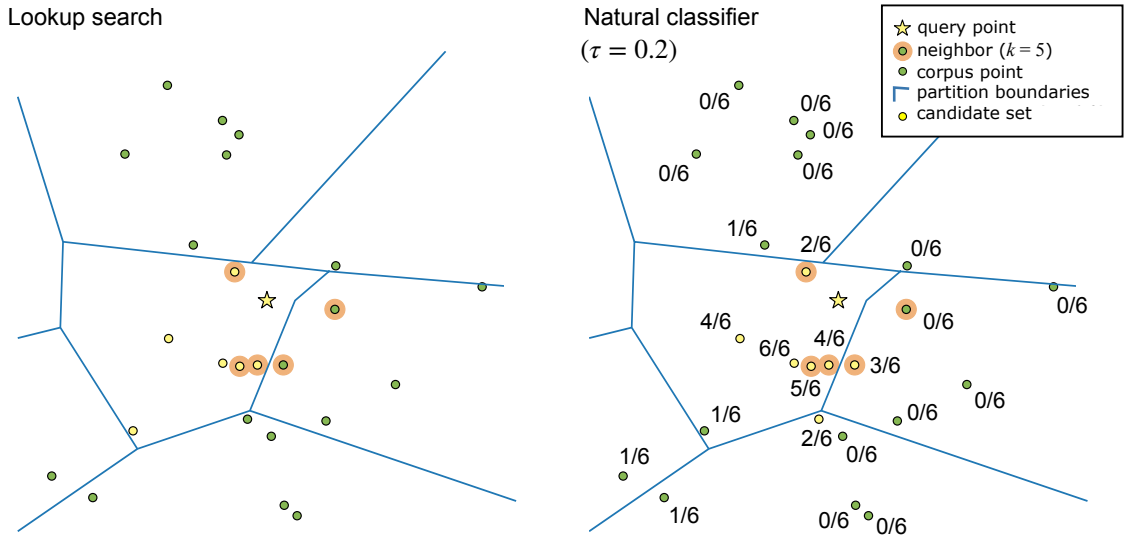

Figure 1: A comparison of two candidate set selection methods. *Left:* traditional lookup search; *Right:* the natural classifier suggested by the proposed framework. Lookup search selects all the corpus points that belong to the same partition element as the query point into the candidate set. In contrast, the natural classifier uses the observed label proportions among these points as probability estimates. The fractions displayed in the right panel present these estimates for each label (i.e., corpus point): the denominator is the number of points in the partition element the query point belongs to, and the numerator is the number of these points that have the corresponding corpus point among their $k = 5$ nearest neighbors. The natural classifier selects the candidate set by thresholding these probability estimates (in the figure the value of the threshold parameter is $\tau = 0.2$). Observe that unlike lookup search, the natural classifier may select points that are outside of the partition element containing the query point into the candidate set.

the Euclidean distance $||\cdot||$. Denote the corpus points that are ordered according to their distance to the query point by $c_{(1)}, \ldots, c_{(m)}$. The k corpus points that are closest to the query point are called its k *nearest neighbors* and the set of their indices is denoted by

$$\text{NN}_k(x) := \{j \in \{1, \ldots, m\} \,:\, ||x - c_j|| \leq ||x - c_{(k)}||\}. \tag{1}$$

Observe that in the corner case where there are other corpus points with the same distance to the query point as $c_{(k)}$, $\text{NN}_k(x)$ contains more than k points[2]. However, when the distribution of the query point x is continuous, there are no ties almost surely, and thus a.s. $|\text{NN}_k(x)| = k$.

2. Another option for defining the set $\text{NN}_k(x)$ is to limit its size to k by breaking ties arbitrarily. However, since this would lead to artefacts when measuring recall of ANN algorithms when ties are present, we prefer definition (1). Observe that Aumüller et al. (2019a) define $\text{NN}_k(x)$ as always having exactly k points, but circumvent this problem by handling ties in their definition of *recall*.

The trivial solution to the problem of finding the nearest neighbors $\text{NN}_k(x)$ of the query point x is to compute the distance from x to all the corpus points and then sort these distances. However, when the dimensionality of the data is high and the corpus is large, this brute force solution is often too slow if the downstream application requires fast response times, which is often the case in, e.g., recommender systems and image recognition. The first data structure proposed for speeding up nearest neighbor search was the k-d tree (Bentley, 1975). However, for high-dimensional data, a k-d tree is not faster than the brute force approach for exact nearest neighbor search. This is because of the well-known *curse of dimensionality* that affects the non-parametric statistical methods—including the partition-based methods for nearest neighbor search—in general (Lee and Wong, 1977). Although the query speed of index structures for *exact* nearest neighbor search degrades to the extent that they are not an improvement on the brute force approach when the dimensionality of the data increases, this problem can be mitigated by allowing an approximate solution. This is why in modern high-dimensional applications *approximate nearest neighbor* (ANN) search is typically used when a fast solution to the nearest neighbor problem is required.

Algorithms for ANN search can be divided into three categories: graphs (Malkov et al., 2014; Malkov and Yashunin, 2018; Iwasaki and Miyazaki, 2018; Baranchuk et al., 2019), quantization (Jegou et al., 2010; Johnson et al., 2019; Guo et al., 2020), and space-partitioning methods. In this article, we consider space-partitioning methods that can be further divided into tree-based (Muja and Lowe, 2014; Dasgupta and Sinha, 2015; Jääsaari et al., 2019) and hashing-based (Datar et al., 2004; Aumüller et al., 2019b; Gong et al., 2020) algorithms that use trees and hash tables, respectively, as index structures.

Space-partitioning algorithms for ANN search use an index structure to select a *candidate set* $S(x) \subset \{1, \ldots, m\}$ of potential nearest neighbors. They then calculate the exact distances between the points of the candidate set and the query point, and return the k nearest points as the approximate nearest neighbors. These algorithms will correctly retrieve a nearest neighbor $j \in \text{NN}_k(x)$ if and only if it belongs to the candidate set. Thus, the *recall* of a space-partitioning algorithm can be written as

$$\text{Rec}(S(x)) := \frac{|\text{NN}_k(x) \cap S(x)|}{|\text{NN}_k(x)|}, \tag{2}$$

where we denote the number of elements of the set A by $|A|$. The performance of an approximate nearest neighbor algorithm is typically measured by its average *recall-query time tradeoff* (see, e.g., Aumüller et al., 2019a or Li et al., 2019)—i.e., the average query time required to reach a certain average recall level on a set of test queries.

2.2 Multilabel classification

Consider a standard multi-label classification problem with m labels. Let $X \in \mathbb{R}^d$ be a random variable and let $L(X) \subseteq \{1, \ldots, m\}$ be the corresponding label set. Equivalently, the output variable can be defined with a binary encoding by letting $Y \in \{0, 1\}^m$ be an m-bit random vector, where

$$Y_j = \begin{cases} 1, & \text{if } j \in L(X), \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

A multilabel classifier is an m-component function $g = (g_1, \ldots, g_m) : \mathbb{R}^d \to \{0,1\}^m$ that attaches a label set to the value of the input variable X. Denote the training set that is assumed to be an i.i.d. sample from the distribution of the pair (X, Y) by $D_n := \{(X_i, Y_i)\}_{i=1}^n$. When the classifier $g : \mathbb{R}^d \times \{\mathbb{R}^d \times \{0,1\}^m\}^n \to \{0,1\}^m$ is learned from the training set of size n, we denote it by $g^{(n)}(x) := g(x, D_n)$. When the training set D_n is considered a random variable, the classifier $g^{(n)}$ also becomes a random function.

The performance of a classifier is measured by a loss function $L : \{0,1\}^m \times \{0,1\}^m \to \mathbb{R}$, and the objective is to minimize the risk $\mathcal{R}(g) := E[L(g(X), Y)]$. This risk is lower-bounded by the *Bayes risk* $\mathcal{R}^* = \inf_g \mathcal{R}(g)$, the minimizer of which is called the *Bayes classifier*.

The Bayes classifier for many common multilabel loss functions—such as Hamming loss, ranking loss, precision, recall, and F-measures—is obtained by thresholding the conditional label probabilities $\eta_j(x) := P\{Y_j = 1 \,|\, X = x\}$ (Dembczynski et al., 2010; Koyejo et al., 2015). This justifies the standard plug-in approach of first estimating the conditional label probabilities[3] by $\hat{\eta}_1(x), \ldots, \hat{\eta}_m(x)$, and then defining the *plug-in classifier* as

$$g_j^{(n)}(x) := \begin{cases} 1, & \text{if } \hat{\eta}_j(x) > \tau \\ 0, & \text{otherwise,} \end{cases} \tag{4}$$

where $\tau \in [0,1]$. Equivalently, the plug-in classifier can be written as an estimate of the label set $L(x)$ as $\hat{L}(x) := \{j \in \{1, \ldots, m\} : \hat{\eta}_j(x) > \tau\}$.

The multilabel classification problem is often solved by reducing it to a series of binary or multiclass classification problems and estimating the conditional label probabilities $\eta_j(x)$ under this model. (see, e.g., Menon et al. (2019) for a discussion of different reduction methods). In what follows, we will employ the *pick-all-labels* (PAL) reduction (Reddi et al., 2019) where we separate each label $l \in L(x_i)$ of the training set point x_i into a multi*class* (but single-label) training instance (x_i, l), and fit the classifier to this modified training set by minimizing a multiclass loss function.

3. Candidate set selection as a multilabel classification problem

Equipped with the above definitions, we first formalize the candidate set selection in ANN search described in Sec. 2.1 as an instance of the multilabel classification problem described in Sec. 2.2. Then, we characterize the optimal decision boundaries of this classification problem.

3.1 Formulation as a multilabel classification problem

In the classical formulation of ANN search, the input–output pair is defined as $(x, \textsc{nn}_k(x))$. It is straightforward to observe that this is an instance of the multilabel classification problem where the label set $L(x)$ is $\textsc{nn}_k(x)$—i.e., the set of indices of the k nearest neighbors of the query point. Assuming that the values of x are i.i.d. draws from the distribution of

3. More generally, instead of the conditional label probability estimates $\hat{\eta}_1(x), \ldots, \hat{\eta}_m(x)$, any score function values $s_1(x), \ldots, s_m(x)$ for the labels can be learned and thresholded to make the classification decision. While we will present all the results for only the version of the plug-in classifier that uses the probability estimates, they readily generalize to the version of the plug-in classifier that uses the score function values.

the random variable X (the *query distribution*), the objective is to predict the value of the random variable Y (defined by (3) with $\text{NN}_k(X)$ as a label set) given the value of the random variable X. Since the labels $\{1, \ldots, m\}$ correspond to the indices of the corpus points, the classification decision (4) where the probability estimates are thresholded corresponds to the candidate set selection, and the estimated label set $\hat{L}(x)$ corresponds to the candidate set $S(x)$. Observe that it always holds for the true label set that $|L(x)| = \sum_{j=1}^{m} y_j \geq k$, and when the query distribution is continuous, it holds almost surely that $|L(x)| = k$.

If no additional training data is available, the corpus itself can be used as a training set. More precisely, in this case we interpret $\{c_j\}_{j=1}^m$ as a sample from the query distribution, compute the k nearest neighbors of the corpus points, and then use $\{(c_j, y_j)\}_{j=1}^m$ as a training set. Note that in this case $y_{jj} = 1$ for each $j = 1, \ldots, m$ since each corpus point is the nearest neighbor of itself.

3.2 Characterization of the optimal decision boundaries

Next, we examine the form of the *optimal decision boundary*—i.e., the decision boundary of the Bayes classifier—of the classification problem defined above. The components of the Bayes classifier $g^* = (g_1^*, \ldots, g_m^*)$ for the multilabel 0-1 loss and, by extension, for other common multilabel loss functions, such as F-measures, can be written as

$$g_j^*(x) = \begin{cases} 1, & \text{if } j \in \text{NN}_k(x), \\ 0, & \text{otherwise,} \end{cases}$$

for each $j = 1, \ldots, m$. Equivalently, the optimal decisions are described by the *optimal decision regions* $E_1^{(k)}, \ldots, E_m^{(k)}$, where

$$E_j^{(k)} := \{x \in \mathbb{R}^d : g_j^*(x) = 1\} = \{x \in \mathbb{R}^d : j \in \text{NN}_k(x)\}$$

for each $j = 1, \ldots, m$. When $k = 1$,

$$\begin{aligned} E_j^{(1)} &= \{x \in \mathbb{R}^d : j \in \text{NN}_1(x)\} \\ &= \{x \in \mathbb{R}^d : ||x - c_j|| \leq ||x - c_{j'}|| \text{ for all } j' \neq j\}, \end{aligned} \tag{5}$$

for each $j = 1, \ldots, m$, i.e., the sets $E_1^{(1)}, \ldots, E_m^{(1)}$ define the standard *Voronoi tessellation* of the feature space. The cells of the Voronoi tessellation are convex polytopes. When $k > 1$, the sets $E_1^{(k)}, \ldots, E_m^{(k)}$ are not necessarily convex. However, we can show that they are *star-convex polytopes* (see Fig. 2 for an illustration). In particular, we will show that, first, $E_1^{(k)}, \ldots, E_m^{(k)}$ are star-convex; and, second, they can be written as finite unions of finite intersections of half-spaces.

Definition 1 *(Star-convexity) The set $S \in \mathbb{R}^d$ is called star-convex if there exists $x_0 \in S$ s.t. for every point $x \in S$ the line segment connecting x to x_0 is in S.*

Theorem 2 *For any $j, k \in \{1, \ldots, m\}$, the set*

$$E_j^{(k)} = \{x \in \mathbb{R}^d : j \in \text{NN}_k(x)\}$$

is star-convex.

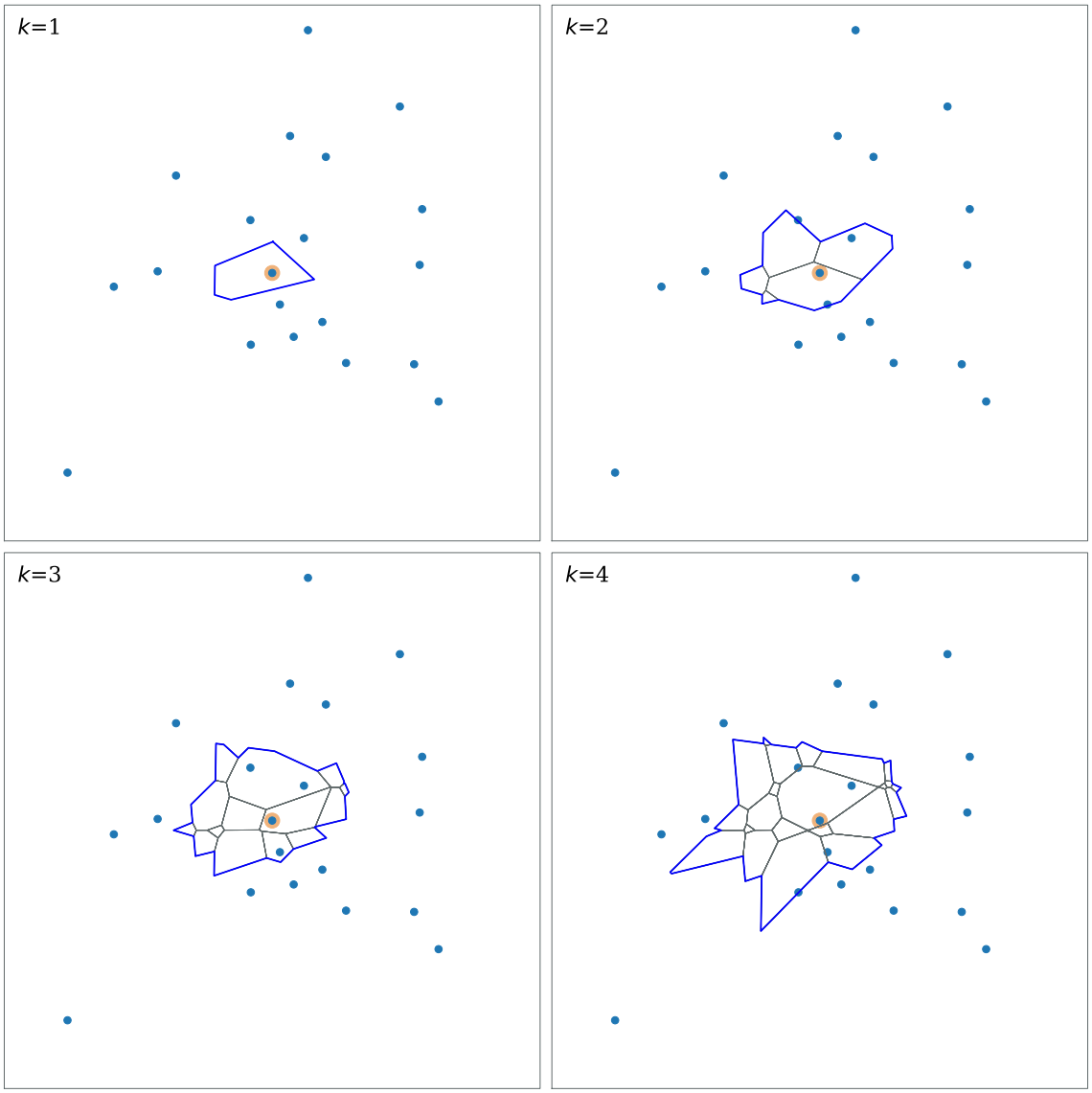

Figure 2: An example of the star-convex optimal decision regions $E_j^{(k)}$ related to the corpus point c_j (the highlighted point in the middle) for $k = 1, 2, 3, 4$. Dots represent the corpus points. The outer boundary of the indicated region is the decision boundary: the highlighted corpus point is among the k nearest neighbors of a query point if and only if the query point falls within the boundary. The smaller regions $E_I^{(k)}$ inside the decision boundary correspond to the cells of a kth order Voronoi tessellation. In other words, each region $E_I^{(k)}$ represents the region where $\text{NN}_k(x) = I$, i.e. the k distinct corpus points defined by the set I are the k nearest neighbors of any $x \in I$.

Proof We prove that c_j is the center-point x_0 of Definition 1 by assuming that there exists $x_1 \in E_j^{(k)}$ and $x_2 \notin E_j^{(k)}$ s.t. x_2 lies on the line segment connecting x_1 to c_j, and showing that this leads to a contradiction. Denote by $c_{j'}$ any corpus point that belongs to $\text{NN}_k(x_2)$. Since $c_j \notin \text{NN}_k(x_2)$, it holds that $||x_2 - c_{j'}|| < ||x_2 - c_j||$. Using the triangle inequality and the assumption that x_2 lies on the line segment connecting x_1 to c_j we have

$$\begin{aligned} ||x_1 - c_{j'}|| &\leq ||x_1 - x_2|| + ||x_2 - c_{j'}|| \\ &< ||x_1 - x_2|| + ||x_2 - c_j|| \\ &= ||x_1 - c_j||. \end{aligned}$$

This means that all the corpus points that belong to $\text{NN}_k(x_2)$ are closer to x_1 than c_j. But this contradicts the assumption that $j \in \text{NN}_k(x_1)$. Since $c_j \in E_j^{(k)}$, we have proven the claim. ■

By definition, each cell of the Voronoi tessellation $E_1^{(1)}, \ldots, E_m^{(1)}$ can be written as a finite intersection of half-spaces. Denote by

$$E_{j,j'} := \{x \in \mathbb{R}^d \,:\, ||x - c_j|| \leq ||x - c_{j'}||\}$$

the half-space that contains the points that are closer to or as close to the jth than the j'th corpus point. The set $E_j^{(1)}$ is now obtained as

$$\begin{aligned} E_j^{(1)} &= \{x \in \mathbb{R}^d \,:\, \forall j' \neq j : ||x - c_j|| \leq ||x - c_{j'}||\} \\ &= \bigcap_{j' \neq j} \{x \in \mathbb{R}^d \,:\, ||x - c_j|| \leq ||x - c_{j'}||\} \\ &= \bigcap_{j' \neq j} E_{j,j'}. \end{aligned}$$

The cells of the *k-th order Voronoi tessellation* $\{E_I^{(k)}\}_{I \in \Pi^{(k)}([m])}$, where $\Pi^{(k)}(A)$ denotes the set of all k-combinations—i.e., subsets with k distinct elements—of the set A and $[m] := \{1, \ldots, m\}$, are defined as

$$E_I^{(k)} := \{x \in \mathbb{R}^d \,:\, I \subset \text{NN}_k(x)\},$$

i.e., they are sets that contain the points of the feature space that have the corpus points indexed by I as their k nearest neighbors. Like the cells of the standard Voronoi tessellation, the cells of k-th order Voronoi tessellation can also be written as finite intersections of the half-spaces as

$$\begin{aligned} E_I^{(k)} &= \{x \in \mathbb{R}^d \,:\, I \subset \text{NN}_k(x)\} \\ &= \{x \in \mathbb{R}^d \,:\, \forall j \in I, j' \notin I : ||x - c_j|| \leq ||x - c_{j'}||\} \\ &= \bigcap_{j \in I, j' \notin I} \{x \in \mathbb{R}^d \,:\, ||x - c_j|| \leq ||x - c_{j'}||\} \\ &= \bigcap_{j \in I, j' \notin I} E_{j,j'}, \end{aligned} \tag{6}$$

where the second equality holds because $|I| = k$.

Finally, we show that the sets $E_1^{(k)}, \ldots, E_m^{(k)}$ that define the optimal decision boundary can be obtained as finite unions of finite intersections of half-spaces, or, equivalently, as finite unions of the cells of the kth order Voronoi tessellation.

Theorem 3 *For any $j, k \in \{1, \ldots, m\}$, the set $E_j^{(k)}$ can be written as*

$$\begin{aligned} E_j^{(k)} &= \bigcup_{\{I \in \Pi^{(k)}([m]) \,:\, I \ni j\}} \; \bigcap_{j' \in I, j'' \notin I} E_{j',j''} \\ &= \bigcup_{\{I \in \Pi^{(k)}([m]) \,:\, I \ni j\}} E_I^{(k)}. \end{aligned}$$

Proof A point x has c_j among its k nearest neighbors if and only if there exists a set of k corpus points that contains c_j and the corpus points that do not belong into this set are not closer to x than any of the corpus points of this set. Thus, we have

$$\begin{aligned} E_j^{(k)} &= \{x \in \mathbb{R}^d \,:\, \exists I \in \Pi^{(k)}([m]) \text{ s.t. } I \ni j \text{ and } \forall j' \notin I, j'' \in I : ||x - c_{j''}|| \leq ||x - c_{j'}||\} \\ &= \bigcup_{\{I \in \Pi^{(k)}([m]) \,:\, I \ni j\}} \{x \in \mathbb{R}^d \,:\, \forall j' \notin I, j'' \in I : ||x - c_{j''}|| \leq ||x - c_{j'}||\} \\ &= \bigcup_{\{I \in \Pi^{(k)}([m]) \,:\, I \ni j\}} \; \bigcap_{j' \in I, j'' \notin I} E_{j',j''} \\ &= \bigcup_{\{I \in \Pi^{(k)}([m]) \,:\, I \ni j\}} E_I^{(k)}, \end{aligned}$$

where the last equality follows from (6). ■

4. Partitioning classifiers for multilabel classification

Partitioning classifier is a general term for a classifier that is based on a partition of the feature space and whose classification decision is based on the labels of the training set points that belong to the same partition element as the query point. Partitioning classifiers can be divided into two categories depending on whether they learn flat or recursive partitions. There is a vast literature on recursive partitioning classifiers (i.e., classification trees); specifically, gradient boosted trees (Friedman et al., 2000; Friedman, 2001) are one of the most widely used and efficient classifiers (Chen and Guestrin, 2016). Flat partitions are more typically used for unsupervised tasks, such as density estimation (Kontkanen and Myllymäki, 2007; López-Rubio, 2013; Cui et al., 2021) and clustering (Hartigan, 1975; Ester et al., 1996), but they have also been applied to classification (Lugosi and Nobel, 1996; McAllester and Ortiz, 2003).

4.1 Learning a partition

Denote by $\mathcal{P} = \{R_1, R_2, \ldots, R_L\}$ a partition of $\mathbb{R}^d$, i.e., a collection of disjoint sets for which $\bigcup_{l=1}^{L} R_l = \mathbb{R}^d$. Denote the structure function that maps the query point to the index of the

partition element it belongs into by $q : \mathbb{R}^d \to \{1, 2, \ldots, L\}$. When the partition is learned from the training data D_n, we denote it by $\mathcal{P}^{(n)} = \pi_n(D_n)$, where π_n is a partitioning rule, i.e., a function that associates the n-tuple $\{(x_i, y_i)\}_{i=1}^n$ with a (measurable) partition of $\mathbb{R}^d$ from the fixed family of partitions

$$\mathcal{F}_n := \{\pi_n((x_1, y_1), \ldots, (x_n, y_n)) \,:\, (x_i, y_i) \in \mathbb{R}^d \times \{0, 1\}^m \text{ for each } i = 1, \ldots, n\}.$$

When the the partitioning rule π_n is randomized, we define it as a function that associates the n-tuple $\{(x_i, y_i)\}_{i=1}^n$ and the observed value of the random vector $Z \in \mathcal{Z}$ with a partition of $\mathbb{R}^d$ from the fixed family of partitions

$$\mathcal{F}_n := \{\pi_n((x_1, y_1), \ldots, (x_n, y_n), z) \,:\, (x_i, y_i) \in \mathbb{R}^d \times \{0, 1\}^m \text{ for each } i = 1, \ldots, n, z \in \mathcal{Z}\}.$$

A randomized partitioning rule is typically used to learn a collection of partitions $\{\mathcal{P}_t^{(n)}\}_{t=1}^T$, where the tth partition is $\mathcal{P}_t^{(n)} = \pi_n(D_n, Z_t)$; we denote by $Z_1, \ldots, Z_T$ the i.i.d. random vectors that are used to construct T randomized partitions. Further, denote by $q_t : \mathbb{R}^d \to \{1, 2, \ldots, L_t\}$ the structure function corresponding to tth partition $\mathcal{P}_t = \{R_1^{(t)}, \ldots R_{L_t}^{(t)}\}$

4.2 Natural classifier

Partitioning classifiers use the training set twice: first, to learn the partition $\mathcal{P}^{(n)} = \pi(D_n)$, and second, to classify the query point x using the training set points that belong to the same partition element $R_{q(x)}$ with it. In the binary and multiclass classification the classification decision is typically done via majority voting among the classes of the training set points that belong to the same partition element as the query point. This partitioning classifier is called *the natural classifier* (Devroye et al., 1996, Chapter 21).

In multilabel classification, majority voting is not defined unambiguously, since the query point can have more than one correct label. Instead, a typical approach is to use a plug-in classifier (4) that assigns the labels to the query point x by thresholding the conditional label probability estimates $\hat{\eta}_1(x), \ldots, \hat{\eta}_j(x)$. The most straightforward way to obtain these estimates is to use the observed label proportions

$$\hat{\eta}_j(x) = \frac{1}{N_{q(x)}} \sum_{i \,:\, x_i \in R_{q(x)}} y_{ij}. \tag{7}$$

among the training set points that belong to the same partition element with the query point. We denote by $N_{q(x)} := |\{i \,:\, x_i \in R_{q(x)}\}|$ the number of training set points in that partition element. The labels of the ith training set point are denoted by $y_{i1}, \ldots y_{im}$, and encoded using binary encoding as in (3).

We call the plug-in classifier (4) where the conditional label probability estimates are defined by (7) the *natural classifier*. Estimating the conditional label probabilities by (7) can be motivated via the one-versus-all reduction[4] under which the observed label proportions among the training set points that belong to $R_{q(x)}$ are the maximum likelihood estimates of the piecewise constant binomial model. Observe that when all the data points have the same number of labels, the observed label proportions (7) are also proportional to the

4. See Menon et al. (2019) for a review of multilabel problem reductions.

maximum likelihood estimates of the piecewise constant multinomial model under the PAL reduction.

When a collection of partitions $\{\mathcal{P}_t^{(n)}\}_{t=1}^T$, where each partition $\mathcal{P}_t^{(n)} := \{R_1^{(t)}, \ldots, R_{L_t}^{(t)}\}$ is learned by a randomized partitioning rule $\pi_n(D_n, Z)$, is used as a classifier, the contributions of the partitions are aggregated. The conditional label probability estimates of an ensemble can be obtained in a straightforward fashion by averaging the conditional probability estimates of the individual partitions:

$$\hat{\eta}_j(x) = \frac{1}{T} \sum_{t=1}^{T} \hat{\eta}_j^{(t)}(x). \tag{8}$$

The estimates $\hat{\eta}_j^{(1)}(x), \ldots, \hat{\eta}_j^{(T)}(x)$ are defined as the observed label proportions as in (7). We also refer to the plug-in classifier (4) where the conditional label probabilities are estimated by (8) as the *natural classifier*.

5. Natural classifier for candidate set selection

Partition-based index structures can be interpreted as partitioning classifiers for the multilabel classification problem introduced in Sec. 3.1. In particular, it follows immediately that the natural classifier defined in the previous section can be used as a candidate set selection method for ANN search. To this end, consider an algorithm for learning a partition, such as Algorithm 3 that grows a RP tree. Formally, this algorithm is a (randomized) partitioning rule π_n that associates a training set $\{(x_i, y_i)\}_{i=1}^n$ and an observed value of the random vector Z with a partition $\mathcal{P}^{(n)} = \{R_1, \ldots, R_L\}$. Given this partition and the threshold parameter $\tau \in [0, 1]$, the label set estimated by the natural classifier is

$$\hat{L}(x) = \{j \in \{1, \ldots, m\} \,|\, \hat{\eta}_j(x) > \tau\}, \tag{9}$$

where the conditional label probability estimates $\hat{\eta}_j(x)$ are defined by (7) as the observed label proportions among the training set points in the partition element $R_{q(x)}$ that contains the query point. In other words, the natural classifier selects into the candidate set those corpus points that are among the k nearest neighbors of a sufficiently high proportion of the training set points that belong to $R_{q(x)}$. For clarity, Algorithm 1 describes the natural classifier for the candidate set selection in detail[5].

In contrast, in lookup search the candidate set selected as

$$S(x) = \{j \in \{1, \ldots, m\} \,|\, c_j \in R_{q(x)}\},$$

i.e., a corpus point c_j is selected into the candidate set if and only if it belongs to the same partition element as the query point. Also this candidate set can be interpreted as a classification decision of a multilabel classifier by writing it in a same form as a natural classifier (9); in this case the conditional label probability estimates are of the form $\hat{\eta}_j(x) = \mathbb{1}_{R_{q(x)}}(c_j)$. However, this is not a natural classifier for the multilabel classification problem

5. We observed that on some data sets it improves performance of the algorithm to compute the conditional label probability estimates $\hat{\eta}_j(x)$ using a value $k' > k$; this adds an additional tuning parameter to the algorithm.

Algorithm 1 Natural classifier for ANN search using a single partition

Input: query point x, corpus $\{c_j\}_{j=1}^m$, training set $\{x_i\}_{i=1}^n$ partition $\mathcal{P} = (R_1, \ldots, R_L)$, threshold parameter τ, number of neighbors searched for k, number of neighbors of the training set points used to compute the conditional label probability estimates k'

Output: indices of the approximate k nearest neighbors of the query point x among the corpus points $\{c_j\}_{j=1}^m$

procedure NATURAL-CLASSIFIER(x, $\{c_j\}_{j=1}^m$, $\{x_i\}_{i=1}^n$, $\mathcal{P}$, τ, k, k')

$N_{q(x)} \leftarrow |\{x_i \in R_{q(x)}\}|$

for $j \in 1, \ldots, m$ **do**

$\hat{\eta}_j(x) \leftarrow \frac{1}{N_{q(x)}} \sum_{x_i \in R_{q(x)}} \mathbb{1}\{j \in \text{NN}_{k'}(x_i)\}$

$\hat{L}(x) \leftarrow \{j \in \{1, \ldots, m\} \,|\, \hat{\eta}_j(x) > \tau\}$

for $j \in \hat{L}(x)$ **do**

compute $d(x, c_j)$

return indices of the k corpus points in $\hat{L}(x)$ for which $d(x, c_j)$ is smallest

we defined in Sec. 3.1. Instead, it is a natural classifier for a distinct multilabel classification problem in which the labels of the training set point x_i are defined as $\tilde{y}_{ij} = \mathbb{1}_{R_q(x_i)}(c_j)$. In other words, the labels represent the corpus points that belong to the same partition element with x_i. This suggests that the natural classifier would be a more efficient candidate set selection method than lookup search since the natural classifier is directly optimized (under the common OVA and PAL problem reductions) to solve the problem of interest.

The experimental results support the above analysis. Fig. 5 compares different candidate set methods on four benchmark data sets while keeping the partition type fixed. The natural classifier is significantly faster candidate set method compared to lookup search in all the cases.

Finally, consider an ensemble of partitions $\{\mathcal{P}_t^{(n)}\}_{t=1}^T$, that is obtained by a randomized partitioning rule π_n (such as Algorithm 3 for growing a RP tree) with T different random vectors. As in the case of a single partition, the natural classifier can be used to estimate the label set as in (9). The only difference is that the conditional label probability estimates are obtained by (8) as the averages of the estimates of the single partitions. The natural classifier for candidate set selection using a collection of partitions is described in detail in Algorithm 2.

6. Related work

The most directly relevant earlier literature consists of studies that learn space-partitioning index structures for ANN search using supervised information. The idea of optimising the index structure for the particular query distribution was first presented by Maneewongvatana and Mount (2001), and later extended by Cayton and Dasgupta (2007) who formulate ANN search as a supervised learning problem and propose a tree-based and a hashing-based algorithm for solving it. More recently, many supervised *learning to hash*-methods have been proposed. In this section, we first briefly discuss the supervised hashing methods, and then

Algorithm 2 Natural classifier for ANN search using multiple partitions

Input: query point x, corpus $\{c_j\}_{j=1}^m$, training set $\{x_i\}_{i=1}^n$ collection of partitions $\{\mathcal{P}_t\}_{t=1}^T$, threshold parameter τ, number of neighbors searched for k, number of neighbors of the training set points used to compute the conditional label probability estimates k'

Output: indices of the approximate k nearest neighbors of the query point x among the corpus points $\{c_j\}_{j=1}^m$

procedure NATURAL-CLASSIFIER(x, $\{c_j\}_{j=1}^m$, $\{x_i\}_{i=1}^n$, $\{\mathcal{P}_t\}_{t=1}^T$, τ, k, k')

  for $j = 1, \ldots, m$ **do**

    for t = 1, ..., T **do**

      $N \leftarrow |\{x_i \in R^{(t)}_{q_t(x)}\}|$

      $\hat{\eta}_j^{(t)}(x) \leftarrow \frac{1}{N} \sum_{x_i \in R^{(t)}_{q_t(x)}} \mathbb{1}\{j \in \text{NN}_{k'}(x_i)\}$

    $\hat{\eta}_j(x) \leftarrow \frac{1}{T} \sum_{t=1}^T \hat{\eta}_j^{(t)}(x)$

  $\hat{L}(x) \leftarrow \{j \in \{1, \ldots, m\} \,|\, \hat{\eta}_j(x) > \tau\}$

  for $j \in \hat{L}(x)$ **do**

    compute $d(x, c_j)$

  return indices of the k corpus points in $\hat{L}(x)$ for which $d(x, c_j)$ is smallest

discuss how the proposed framework differs from the earlier supervised methods for ANN search.

6.1 Data-dependent hashing methods

Data-dependent hashing methods partition the feature space by using binary hash functions $q_1, \ldots, q_L$, where $q_l : \mathbb{R}^d \to \{-1, 1\}$ for each $l = 1, \ldots, L$. Typically, these hash functions have the form of *generalized linear projection*

$$q_l(x) = \text{sign}(f(w_l \cdot x + b_l)), \tag{10}$$

where $f : \mathbb{R} \to \mathbb{R}$ is a non-linear link function that is applied to the linear term $w_l \cdot x + b_l$. The data-dependent hashing methods learn the weight term w_l and the bias term b_l from the training set.

There exists many different data-dependent hashing methods that differ on their choice of a link function f and the learning method used to obtain the weights and the bias term. Data-dependent hashing methods can be classified into *unsupervised* (see, e.g., Weiss et al., 2009; Gong et al., 2012; Liu et al., 2011; Kong and Li, 2012) and *supervised* methods. Among them, the most directly relevant to our work are supervised methods that use the true nearest neighbors of the training set points as the output variable when learning the hash functions. Representative examples of supervised hashing methods include *linear discriminant analysis hashing* (LDAH) (Strecha et al., 2011), *minimal loss hashing* (Norouzi and Fleet, 2011), *kernel-based supervised hashing* (KSH) (Liu et al., 2012), and *central similarity quantization* (CSQ) (Yuan et al., 2020). See, e.g., Wang et al. (2015) or Wang et al. (2017) for a more detailed survey on data-dependent hashing methods.

6.2 Difference to the earlier supervised methods

There are two key components in a partition-based ANN algorithm: the index structure and the candidate set selection method. A vast amount of different index structures have been proposed for ANN search. However, candidate set selection methods have received relatively little attention. Even the earlier *supervised* methods use lookup search, i.e., they select to the candidate set all the corpus point that belong to the same partition element as the query point (or to all the partition elements the query point is routed into if backtracking is used). Hence, the earlier supervised methods formulate the learning problem in an indirect fashion, typically as maximizing the number of nearest neighbors of the query point that belong to the same partition element as the query point, while simultaneously minimizing the number of non-neighbors in the same element.

For instance, Cayton and Dasgupta (2007) formulate the learning problem as maximizing the expected recall

$$E_X \mathrm{Rec}(X) = E_X \left[\frac{|\mathrm{NN}_k(X) \cap S(X)|}{k} \right]$$

while minimizing the expected candidate set size $E_X|S(X)|$ over the query distribution. However, in this formulation the candidate set $S(x)$ is explicitly defined as the set of corpus points that belong to the same partition element as the query point in an index structure, i.e.,

$$S(x) = \{j \in \{1, \ldots, m\} \, : \, c_j \in R_{q(x)}\}. \tag{11}$$

The same principle holds for supervised hashing methods. They select into the candidate set the corpus points whose hash code is within a certain Hamming radius of the hash code of the query point. Using the terminology of this article, their candidate set selection method is lookup search with backtracking. This leads to a formulation of the learning problem that differs from ours (Sec. 3.1). For instance, in supervised kernel hashing (Liu et al., 2012) the problem is formulated as learning a binary embedding that minimizes the Hamming distance between the query and its nearest neighbors, while maximizing the Hamming distance between the query and its non-neighbors.

To summarize, the key observation of this article that distinguishes the proposed framework from the earlier supervised methods for ANN search is that the lookup search is not a necessary component of a partition-based algorithm. This means that the learning problem does not have to be formulated in an indirect fashion as maximizing the number of nearest neighbors that belong to the partition elements that are near the query point. Instead, we formulate the candidate set selection *itself* directly as a multilabel classification problem and, consequently, interpret the points that belong to the same partition element with the query point as training set points whose labels—that represent their nearest neighbors—are used to make the classification decision. Our empirical results (see Fig. 5) indicate the utility of the proposed formulation: the natural classifier directly suggested by the proposed framework has superior performance compared to lookup search.

7. Theoretical results

In this section, we first provide a sufficient condition for the consistency of a partitioning classifier for ANN search (Sec. 7.1). We then verify this condition for chronological k-d trees

(Sec. 7.2), dense RP trees (Sec. 7.3.1), and sparse RP trees (Sec. 7.3.2). The consistency of ensembles of both sparse and dense RP trees also follow immediately from the results of this section. Table 1 provides a summary of results used to establish consistency of these trees and their ensembles, and the required assumptions on the distribution of the input variable X.

Table 1: Summary of the consistency results of this article.

Tree type	Proof	Assumptions on X
Chronological k-d tree	Theorem 5 + Theorem 11	Density
RP tree	Theorem 6 + Theorem 13	Density + bounded support
Sparse RP tree	Theorem 6 + Theorem 15	Density + bounded support
Ensemble of RP trees	Corollary 7 + Theorem 13	Density + bounded support
Ensemble of sparse RP trees	Corollary 7 + Theorem 15	Density + bounded support

7.1 A sufficient condition for consistency

We first provide a sufficient condition for the consistency of a partitioning classifier for ANN search (Theorem 5). Then we show that the result holds with a slightly weaker condition when the distribution of the input X variable has bounded support (Theorem 6). Finally, we show that if the conditions of either of these results hold for a randomized partitioning rule, then the partitioning classifier based on the collection of classifiers learned by this partitioning rule is also consistent (Corollary 7).

The classical theorem for proving the consistency of partitioning classifiers for binary classification is the following:

Theorem 4 *(Devroye et al. (1996), Theorem 6.1, p. 94–95) Assume that only the features $X_1, \ldots, X_n$ are used to learn the partition $\mathcal{P}^{(n)} = \pi(X_1, \ldots, X_n)$. The natural classifier[6] $g^{(n)}$ defined by $\mathcal{P}^{(n)}$ is consistent (under 0-1 loss) for binary classification, if*

(i) $N_{q(X)} \to \infty$ *in probability, and*

(ii) $\operatorname{diam}\left(R_{q(X)}\right) \to 0$ *in probability,*

when $n \to \infty$.

The number of the training set points in the partition element the query point x belongs to is denoted by $N_q(x) := |\{i : X_i \in R_{q(x)}\}|$, and the diameter of a set A is defined as the maximum distance between any two points of this set, which we denote by

$$\operatorname{diam}(A) := \sup_{a,\, b \in A} \|a - b\|.$$

While the above result is for binary classification, it can be readily extended to the multilabel case. However, as a multilabel classification problem ANN search has two distinctive

6. The natural classifier for binary classification is defined as the classifier that classifies the query point into the majority class of the training set points that belong to the same partition element with it.

properties: (i) the Bayes error $\mathcal{R}^*$ is zero; (ii) the decision boundaries of the Bayes classifier are defined by a finite set of hyperplanes (see Theorem 3). It turns out that in this case, the second condition of Theorem 4 is sufficient for the consistency of a partitioning classifier.

Theorem 5 *Let $g^{(n)}$ be a natural classifier for ANN search defined by the partition $\mathcal{P}^{(n)} = (R_1, \ldots, R_L)$ that is learned by the partitioning rule $\pi(D_n, Z)$, and the threshold parameter $\tau \in [0, 1)$ for ANN search. Assume that the distribution of X, denoted by μ, is continuous. If $\operatorname{diam}(R_{q(X)}) \to 0$ in probability—that is, if for every $\epsilon > 0$,*

$$P\{\operatorname{diam}(R_{q(X)}) > \epsilon\} \to 0$$

when $n \to \infty$, then the classifier $g^{(n)}$ is consistent (for 0-1 loss)—i.e., $E_{D_n}\mathcal{R}(g^{(n)}) \to 0$.

Proof If for all the pairs of corpus points $(c_j, c_{j'})$, $j' \neq j$, all the points of the partition element R_l are closer to c_j than $c_{j'}$ (or vice versa)—that is, if there is no such pair $(c_j, c_{j'})$ for which there exists $a, b \in R_l$ such that $\|a - c_j\| < \|a - c_{j'}\|$ and $\|b - c_j\| > \|b - c_{j'}\|$—then also $\hat{\eta}_j(x) = \eta_j(x)$ for each $x \in R_l$ and $j = 1, \ldots, m$; consequently, each $x \in R_l$ is classified correctly for any $\tau \in [0, 1)$. Now, since

$$\begin{aligned}
&P\{g^{(n)}(X) \neq \eta(X)\} \\
&\leq P\{\exists X_i, X_{i'} \in R_{q(X)} \text{ s.t. } Y_i \neq Y_{i'}\} \\
&\leq P\left(\exists j' \neq j : \exists a, b \in R_{q(X)} \text{ s.t. } \|a - c_j\| < \|a - c_{j'}\|,\, \|b - c_j\| > \|b - c_{j'}\|\right) \\
&\leq \sum_{j' \neq j} P\{\exists a, b \in R_{q(X)} \text{ s.t. } \|a - c_j\| < \|a - c_{j'}\|,\, \|b - c_j\| > \|b - c_{j'}\|\},
\end{aligned} \tag{12}$$

to prove consistency of $g^{(n)}$ it is sufficient to show that for all $j, j' \in \{1, \ldots, m\}$, $j \neq j'$,

$$P\{\exists a, b \in R_{q(X)} \text{ s.t. } \|a - c_j\| < \|a - c_{j'}\|,\, \|b - c_j\| > \|b - c_{j'}\|\} \to 0$$

in probability when $n \to \infty$.

Choose any j, j', $j \neq j'$, and denote the hyperplane that is halfway in between the corpus points c_j and $c_{j'}$ by $H := \{x \in \mathbb{R}^d : \|x - c_j\| = \|x - c_{j'}\|\}$. For any $t = 1, 2, \ldots$, let H_t denote the set surrounding H by a margin of width $1/t$. Since $H_1 \supset H_2 \supset H_3 \ldots$, and $H = \cap_{t=1}^{\infty} H_t$, it follows from the upper continuity of the probability measure that $\lim_{t\to\infty} \mu(H_t) = \mu(H)$. Because the Lebesgue measure of the hyperplane H in $\mathbb{R}^d$ is zero and μ is absolutely continuous w.r.t. the Lebesgue measure by the assumption, then also $\lim_{t\to\infty} \mu(H_t) = \mu(H) = 0$.

Now, for any $t = 1, 2, \ldots$, if $R_{q(x)}$ crosses the hyperplane H, then either $x \in H_t$ or the diameter of the $R_{q(x)}$ is greater than $1/t$. Hence,

$$\begin{aligned}
&P\{\exists a, b \in R_{q(X)} \text{ s.t. } \|a - c_j\| < \|a - c_{j'}\|,\, \|b - c_j\| > \|b - c_{j'}\|\} \\
&\leq P\{X \in H_t \text{ or } \operatorname{diam}(R_{q(X)}) > 1/t\} \\
&\leq \mu(H_t) + P\{\operatorname{diam}(R_{q(X)}) > 1/t\}.
\end{aligned}$$

We can get $\mu(H_t)$ as small as desired by choosing a large enough t; and since by assumption the second term is arbitrarily small when n is large enough, the result follows. ■

The result holds with a weaker condition when the query distribution has bounded support.

Theorem 6 *Let $g^{(n)}$ be a natural classifier for ANN search defined by the partition $\mathcal{P}^{(n)} = (R_1, \ldots, R_L)$ learned by the partitioning rule $\pi_n(D_n, Z)$, and the threshold parameter $\tau \in [0, 1)$. Assume that the distribution of X, denoted by μ, is continuous. If there exists $M > 0$ for which $P\{X \in [-M, M]^d\} = 1$ and*

$$P\{\mathrm{diam}(R_{q(X)} \cap [-M, M]^d) > \epsilon\} \to 0$$

for every $\epsilon > 0$ when $n \to \infty$, then the classifier $g^{(n)}$ is consistent (for 0-1 loss)—i.e., $E_{D_n}\mathcal{R}(g^{(n)}) \to 0$.

Proof By assumption, it holds for any n that

$$P\left(\bigcup_{i=1}^{n}\{X_i \notin [-M, M]^d\}\right) \leq \sum_{i=1}^{n} P\{X_i \notin [-M, M]^d\} = 0,$$

and, hence,

$$\begin{aligned} P\{g^{(n)}(X) \neq \eta(X)\} &\leq P\{\exists X_i, X_{i'} \in R_{q(X)} \text{ s.t. } Y_i \neq Y_{i'}\} \\ &= P\{\exists X_i, X_{i'} \in R_{q(X)} \cap [-M, M]^d \text{ s.t. } Y_i \neq Y_{i'}\}. \end{aligned}$$

The rest of the proof now proceeds as in Theorem 5 with the set $R_{q(x)}$ replaced by its intersection with the hypercube $[-M, M]^d$.

■

As an immediate consequence of the above theorems, we have a sufficient condition for the consistency of the the natural classifier (8) that is based on a collection of randomized partitions.

Corollary 7 *If conditions of Theorem 5 or 6 hold for the distribution of X and a partition $\mathcal{P}^{(n)}$ learned by the randomized partitioning rule $\pi(D_n, Z)$, then the natural classifier (8) defined by the threshold parameter value $\tau \in [0, 1)$ and a collection of partitions $\{\mathcal{P}_t^{(n)}\}_{t=1}^{T}$ that are learned by $\pi(D_n, Z)$ is consistent.*

Proof Assume first that the partitioning rule $\pi_n(D_n, Z)$ and the distribution of X satisfy the conditions of Theorem 5. Now,

$$\begin{aligned} &P\{g(X) \neq \eta(X)\} \\ &\leq P\left\{\exists t : \exists j \neq j' : \exists a, b \in R_{q(X)}^{(t)} \text{ s.t. } ||a - c_j|| < ||a - c_{j'}||, ||b - c_j|| > ||b - c_{j'}||\right\} \\ &\leq \sum_{t=1}^{T} P\left\{\exists j \neq j' : \exists a, b \in R_{q(X)}^{(t)} \text{ s.t. } ||a - c_j|| < ||a - c_{j'}||, ||b - c_j|| > ||b - c_{j'}||\right\}. \end{aligned}$$

where the second inequality follows from the union bound. Now the result follows immediately by continuing the proof of Theorem 5 from (12). In case the partitioning rule $\pi_n(D_n, Z)$ and the distribution of X satisfy the conditions of Theorem 6 the proof proceeds in a similar fashion. ■

7.2 Consistency of chronological k-d trees

Next, we illustrate the utility of Theorem 5 by applying it to prove the consistency of the *chronological k-d tree* (Bentley, 1975) that rotates the split directions and uses the same split direction for all the nodes at one level of a tree. At the first level the training data is split at the median of the first coordinates of the data points. At the second level both nodes are split at the median of the second coordinates of the node points. At the $(d+1)$th level, the nodes are split again at the median of the first coordinates, and so on (see Appendix B.1).

More precisely, let $X, X_1, \ldots, X_n \in \mathbb{R}^d$ be i.i.d. random variables. A chronological k-d tree can be formalized as a partitioning rule π that returns the partition $\mathcal{P}^{(n)} = \pi(X_1, \ldots, X_n)$. When the tree height is ℓ, this partition has 2^ℓ elements (also called *leafs*). The leafs are hyperrectangles in $\mathbb{R}^d$. Some of the edges of these hyperrectangles may have an infinite length. To handle these leafs, we introduce the notation where for any $M > 0$ the hypercube $[-M, M]^d$ divides the partition elements $R_1, \ldots, R_{2^\ell}$ into three disjoint sets:

$$\begin{aligned} A &:= \{l \in \{1, \ldots, 2^\ell\} \,:\, R_l \subset [-M, M]^d\}, \\ C &:= \{l \in \{1, \ldots, 2^\ell\} \,:\, R_l \subset \mathbb{R}^d \setminus [-M, M]^d\}, \\ B &:= \{1, \ldots, 2^\ell\} \setminus (A \cup C). \end{aligned} \tag{13}$$

Here A is the set of indexes of the partition elements that are completely inside the hypercube $[-M, M]^d$, B is the set of indexes of the partition elements that cross its boundary, and C is the set of indexes of the partition elements that are completely outside of it.

First, we prove two auxiliary results that bound the number of nodes crossing the boundary of the box $[-M, M]^d$ and the combined length of the edges (in any fixed coordinate direction) of the nodes that reside completely inside $[-M, M]^d$, respectively. It is worth noting that these bounds are of purely combinatorial nature and thus do not depend on the training set.

Lemma 8 *For any training set D_n, it holds for the number of nodes of a chronological k-d tree, denoted by $N_B := |B|$, crossing the border of the hypercube $[-M, M]^d$ that*

$$N_B \leq 4d \cdot 2^{\ell - \frac{\ell}{d}}.$$

Proof The border of the hypercube consists of $(d-1)$-dimensional faces. Choose a coordinate direction $j \in \{1, \ldots, d\}$ and consider a $(d-1)$-face that is orthogonal to that coordinate axis. Denote the number of nodes crossing that $(d-1)$-face by $N_B^{(j)}$. Before any splits there is one node—the whole feature space $\mathbb{R}^d$—that crosses it. Splitting a node that crosses that $(d-1)$-face at a coordinate direction other than j creates two nodes crossing

it if the splitting hyperplane intersects with $[-M, M]^d$ (if the splitting hyperplane does not intersect with $[-M, M]^d$ then it does affect $N_B^{(j)}$). Splitting at the jth direction does not increase $N_B^{(j)}$ since the splitting hyperplane is perpendicular to the $(d-1)$-face we consider and so cannot cross it.

Therefore, if ℓ is a multiple of d, we have $N_B^{(j)} \leq 2^{\ell - \frac{\ell}{d}}$ since each full round of d splits contains $d-1$ splits orthogonal to the jth coordinate direction, each of which may double the number of nodes crossing the $(d-1)$-face, and one split parallel to the jth coordinate direction that doesn't increase the number. If ℓ is not a multiple of d, then

$$N_B^{(j)} \leq 2^{\ell - \lfloor \frac{\ell}{d} \rfloor} \leq 2^{\ell - \frac{\ell}{d} + 1}$$

because the last incomplete round of splits may not contain a split at the jth coordinate direction. Since for each coordinate direction a d-dimensional hypercube has two $(d-1)$-faces that are orthogonal to that coordinate axis, we have[7]

$$N_B \leq 2 \sum_{j=1}^{d} N_B^{(j)} \leq 4d \cdot 2^{\ell - \frac{\ell}{d}}.$$

■

Lemma 9 *Let $j \in \{1, \ldots, d\}$ be any coordinate direction. Denote the length of the node R_l in the jth coordinate direction by V_l. Then for any training set D_n,*

$$\sum_{l \in A} V_l \leq 4M \cdot 2^{\ell - \frac{\ell}{d}}.$$

Proof For each $l = 1, \ldots, 2^\ell$, denote the length of the hyperrectangle $R'_l := R_l \cap [-M, M]^d$ in the jth coordinate direction by V'_l. Clearly, $V_l = V'_l$ for each $l \in A$ by the definition of the set A. Thus,

$$\sum_{l \in A} V_l = \sum_{l \in A} V'_l \leq \sum_{l=1}^{2^\ell} V'_l.$$

Before any splits, there is one node with $V'_l = 2M$. Splitting a node in a coordinate direction other than j creates two child nodes with the same length in the jth coordinate direction as the parent node, and thus doubles the contribution of the parent node to the sum over the nodes[8]. When we split a node in the jth coordinate direction, the *sum* of the lengths of the child nodes in the jth direction equals the length of the parent node in that direction; thus, the split does not affect the sum over the nodes. Hence, we have

$$\sum_{l=1}^{2^\ell} V'_l \leq 2M \cdot 2^{\ell - \frac{\ell}{d}} \tag{14}$$

7. Since we are proving an upper bound it does not matter that we double count nodes that cross more than one $(d-1)$-face.

8. Here we assume that the splitting hyperplane intersects with $[-M, M]^d$. If the splitting hyperplane does not intersect with $[-M, M]^d$, then the split does not increase $\sum_{l=1}^{2^\ell} V'_l$. Thus, the inequality in (14) holds as an equality if and only if all the splitting hyperplanes that are not orthogonal to the jth coordinate direction intersect with $[-M, M]^d$.

when ℓ is a multiple of d. When ℓ is not a multiple of d, the last incomplete round may not contain a split in the jth coordinate direction, and thus

$$\sum_{l=1}^{2^\ell} V_l' \leq 2M \cdot 2^{\ell - \lfloor \frac{\ell}{d} \rfloor} \leq 4M \cdot 2^{\ell - \frac{\ell}{d}}.$$

■

Before proceeding to the main result of this section, we prove the following well-known variation of Markov's inequality for completeness.

Lemma 10 *Suppose H is an event, $a > 0$, and X is a non-negative random variable for which $E|X| < \infty$. Then,*

$$P\{X > a, H\} \leq \frac{E[\mathbb{1}_H X]}{a}.$$

Proof We can write

$$E[\mathbb{1}_H X] \geq E[\mathbb{1}_H X \mathbb{1}\{X > a\}] \geq aE[\mathbb{1}_H \mathbb{1}\{X > a\}] = aP\{X > a, H\},$$

and the result follows by dividing by a. ■

We are now in a position to establish the consistency of a chronological k-d tree for approximate nearest neighbor search. In view of Theorem 5 it suffices to prove that the leaf diameter converges to zero in probability when the query distribution is continuous:

Theorem 11 *If for the height of a chronological k-d tree holds that $\ell \to \infty$ when $n \to \infty$, then the leaf diameter* $\operatorname{diam}(R_{q(X)})$ *converges to zero in probability.*

Proof Choose a coordinate direction $j \in \{1, \ldots, d\}$ and denote the length of an edge of the hyperrectangle R_l in that direction by V_l. Since the coordinate direction was chosen arbitrarily, it suffices to show that $V_{q(X)}$ converges to zero in probability to prove that also the cell diameter converges to zero in probability.

For any training set size n, define $\ell' := \min(\ell, \lfloor \log_2 \log_2 n \rfloor)$. Since for any training set[9] D_n the probability $P\{V_{q(X)} > \delta \,|\, D_n\}$ is non-increasing w.r.t. to the tree height and $\ell' \leq \ell$, in order to prove that $P\{V_{q(X)} > \delta\} \to 0$ for the original tree height ℓ, it is sufficient to show that it goes to zero for the tree height ℓ'.

For any $\theta > 0$, there exists $M > 0$ s.t. $P\{X \notin [-M, M]^d\} < \theta$. The box $[-M, M]^d$ divides any partition of $\mathbb{R}^d$ into three disjoint sets A, B, and C as defined in (13) that correspond to the indexes of the nodes completely inside $[-M, M]^d$, the indexes of the nodes crossing its border and the indexes of the nodes completely outside of it, respectively[10]. We

9. To keep notation consistent throughout the article, we denote the training set also here by $D_n := \{(X_i, Y_i)\}_{i=1}^n$. However, it should be observed that the chronological k-d tree uses only the inputs $X_1, \ldots, X_n$ to learn the partition; thus, the learned partition does not depend on the labels $Y_1, \ldots, Y_n$.
10. We define the sets A, B, and C here for a tree of height ℓ'.

can now decompose the probability of the event $\{V_{q(X)} > \delta\}$ into three parts corresponding to the sets A, B, and C:

$$P\{V_{q(X)} > \delta\} \leq P\{V_{q(X)} > \delta, q(X) \in A\} + P\{q(X) \in B\} + P\{q(X) \in C\}. \tag{15}$$

Choose $\epsilon > 0$ and denote the event that no partition element has a probability mass larger than $(1+\epsilon)/2^{\ell'}$ by

$$G := \bigcap_{l=1}^{2^{\ell'}} \left\{ \mu(R_l) \leq \frac{1+\epsilon}{2^{\ell'}} \right\},$$

where $\mu(A) := P\{X \in A\}$ is the probability distribution of X for any measurable set A. Our strategy is to first handle this case where all the leafs contain an approximately equal probability mass, and then bound the probability of G^C by applying the Vapnik-Chervonenkis inequality to show the uniform convergence of the empirical distribution of X to its true distribution in the class of leafs of a chronological k-d tree, i.e., in the class of hyperrectangles in $\mathbb{R}^d$. To this end, we further partition the right hand side of (15) as

$$P\{V_{q(X)} > \delta\} \leq \underbrace{P\{V_{q(X)} > \delta, q(X) \in A, G\}}_{I} + \underbrace{P\{q(X) \in B, G\}}_{II} + \underbrace{P(G^C)}_{III} + \underbrace{P\{q(X) \in C\}}_{IV}$$

and bound these four terms.

Term IV: Since $P\{q(X) \in C\} \leq P\{X \notin [-M, M]^d\} < \theta$, we can get this term as small as desired by choosing a small enough θ.

Term I: By applying Lemma 10 (with the event $\{q(X) \in A\} \cap G$), we see that

$$P\{V_{q(X)} > \delta, q(X) \in A, G\} \leq \frac{E\left[\mathbb{1}_G V_{q(X)} \mathbb{1}\{q(X) \in A\}\right]}{\delta}.$$

Therefore, it suffices to bound $E\left[\mathbb{1}_G V_{q(X)} \mathbb{1}\{q(X) \in A\}\right]$, for which we have

$$\begin{aligned}
E\left[\mathbb{1}_G V_{q(X)} \mathbb{1}\{q(X) \in A\}\right] &= E\left[\mathbb{1}_G E\left[V_{q(X)} \mathbb{1}\{q(X) \in A\} \,|\, D_n\right]\right] \\
&= E\left[\mathbb{1}_G \sum_{l=1}^{2^{\ell'}} \mu(R_l) V_l \mathbb{1}\{l \in A\}\right] \\
&= E\left[\mathbb{1}_G \sum_{l \in A} \mu(R_l) V_l\right] \\
&\leq \frac{1+\epsilon}{2^{\ell'}} E\left[\sum_{l \in A} V_l\right] \\
&\leq \frac{1+\epsilon}{2^{\ell'}} \cdot 4M \cdot 2^{\ell' - \frac{\ell'}{d}} \\
&= 4M \cdot \frac{1+\epsilon}{2^{\ell'/d}},
\end{aligned} \tag{16}$$

where the outermost expectation on the right hand side is w.r.t. D_n, the first inequality follows from the definition of G, and the second inequality follows from Lemma 8. Since by assumption $\ell' \to \infty$ when $n \to \infty$, also $P\{V_{q(X)} > \delta, q(X) \in A, G\} \to 0$.

Term II: Applying a similar technique as in (16), we have

$$\begin{aligned} P\{q(X) \in B, G\} &= E\left[\mathbb{1}_G P\{q(X) \in B \mid D_n\}\right] \\ &= E\left[\mathbb{1}_G \sum_{l=1}^{2^{\ell'}} \mathbb{1}\{l \in B\}\mu(R_l)\right] \\ &\leq \frac{1+\epsilon}{2^{\ell'}} E\left[N_B\right] \\ &\leq \frac{1+\epsilon}{2^{\ell'}} \cdot 4d \cdot 2^{\ell' - \frac{\ell'}{d}} \\ &= 4d \cdot \frac{1+\epsilon}{2^{\ell'/d}}, \end{aligned}$$

where the expectation is w.r.t. D_n, the first inequality follows from the definition of G and the second inequality follows from Lemma 9. Hence, also $P\{q(X) \in B, G\} \to 0$ when $n \to \infty$.

Term III: Finally, we bound the probability of the event G^C. Let $\mathcal{R}$ be the class of all hyperrectangles in $\mathbb{R}^d$. The Vapnik-Chervonenkis dimension of $\mathcal{R}$ is $2d$ (see, e.g., Theorem 13.8. by Devroye et al. (1996, p. 220-221)), and hence we have $s(\mathcal{R}, n) \leq n^{2d}$ for its shatter coefficient (see, e.g., Theorem 13.3. by Devroye et al. (1996, p. 218)). If $n \geq 2 \cdot \frac{\log_2 n}{\epsilon} \geq 2 \cdot \frac{2^{\ell'}}{\epsilon}$, then $\frac{1}{n} \leq \frac{1}{2} \cdot \frac{\epsilon}{2^{\ell'}}$. This means that for large enough n, we have

$$\begin{aligned} P(G^C) &= P\left\{\exists l \text{ s.t. } \mu(R_l) > \frac{1+\epsilon}{2^{\ell'}}\right\} \\ &= P\left\{\exists l \text{ s.t. } \mu(R_l) - \left(\frac{1}{2^{\ell'}} + \frac{1}{n}\right) > \frac{\epsilon}{2^{\ell'}} - \frac{1}{n}\right\} \\ &\leq P\left\{\exists l \text{ s.t. } \mu(R_l) - \mu_n(R_l) > \frac{\epsilon}{2^{\ell'}} - \frac{1}{n}\right\} \\ &\leq P\left\{\exists l \text{ s.t. } \mu(R_l) - \mu_n(R_l) > \frac{\epsilon}{2^{\ell'+1}}\right\} \\ &\leq P\left\{\sup_{R \in \mathcal{R}} |\mu(R) - \mu_n(R)| > \frac{\epsilon}{2^{\ell'+1}}\right\} \\ &\leq 8 s(\mathcal{R}, n) \exp\left\{-\frac{n\epsilon^2}{128 \cdot 2^{2\ell'}}\right\} \\ &\leq 8 n^{2d} \exp\left\{-\frac{n\epsilon^2}{128 \cdot 2^{2\ell'}}\right\} \\ &\leq 8 n^{2d} \exp\left\{-\frac{n\epsilon^2}{128 (\log_2 n)^2}\right\} \to 0 \end{aligned} \tag{17}$$

when $n \to \infty$. The first inequality on the right hand side of (17) follows because for the empirical measure—denoted by $\mu_n(A) := \frac{1}{n}\sum_{i=1}^n \mathbb{1}_A(X_i)$ for any measurable set A—of any leaf R_l it holds that $\mu_n(R_l) \leq \frac{1}{2^{\ell'}} + \frac{1}{n}$. The fourth inequality follows from the Vapnik-Chervonenkis inequality (Vapnik and Chervonenkis, 1971); we use the version presented in Theorem 12.5. by Devroye et al. (1996, p. 197-198). The last inequality follows because $2^{2\ell'} \leq (\log_2 n)^2$ by the definition of ℓ'. ■

Algorithm 3 Grow a random projection (RP) tree (Dasgupta and Freund, 2008; Dasgupta and Sinha, 2015; Hyvönen et al., 2016)

Input: data X, current depth, maximum depth ℓ, random vectors $Z = (Z^{(1)}, \ldots, Z^{(\ell)})$
Output: subtree of depth $\ell -$ depth
procedure GROW-RP(X, depth, ℓ, Z)
  if depth $= \ell$ **then**
    return X as a leaf node
  proj $\leftarrow$ PROJECT($X, Z^{(\text{depth})}$)
  $\hat{s} \leftarrow$ MEDIAN(proj)
  left $\leftarrow$ GROW-RP(points in X for which proj $\leq \hat{s}$, depth + 1, ℓ, Z)
  right $\leftarrow$ GROW-RP(points in X for which proj $> \hat{s}$, depth + 1, ℓ, Z)
  return (left, right, directions[depth], $\hat{s}$) as an inner node

7.3 Consistency of random projection trees

The random projection (RP) tree (Dasgupta and Freund, 2008; Dasgupta and Sinha, 2015), is a data structure that is randomized by construction. At each level of an RP tree, the normal of the splitting hyperplanes is generated from the d-dimensional standard normal distribution $N(0, I)$ (for computational efficiency, the same split direction can be used at each of the sibling nodes on a given level of the tree, as suggested by Hyvönen et al. (2016)). The data in the node are projected onto this random vector and split into two subsets at the median of the projections. Algorithm 3 describes the method of growing an RP tree; in the dense RP tree the random vectors $Z = (Z^{(1)}, \ldots, Z^{(\ell)})$ on each level are generated from the d-dimensional standard normal distribution, i.e.,

$$Z_j^{(l)} \sim N(0, 1)$$

for each $l = 1, \ldots, \ell$, $j = 1, \ldots, d$.

We can formalize an RP tree as a randomized partitioning rule $\pi(D_n, Z)$. In addition to the training data D_n, this partitioning rule takes as an argument the random vectors Z. The partitioning rule $\pi(D_n, Z)$ returns a partition $\mathcal{P}^{(n)} = (R_1, \ldots, R_{2^\ell})$ with 2^ℓ partition elements which are called *leafs*. The leafs are (possibly unbounded) convex polytopes—i.e., areas constrained by the ℓ hyperplanes.

We will first prove consistency of the original (dense) RP tree described above. We will then describe its sparse variant(s) and prove their consistency in Sec. 7.3.2.

7.3.1 The dense case

By Theorem 6, assuming that the query distribution is continuous and has a bounded support, it suffices to show that the diameter of the intersection of the leaf that contains the query point and the hypercube $[-M, M]^d$ converges to zero in probability. For each $j = 1, \ldots, d$, denote by

$$V^{(j)}(A) := \max_{a,b \in A} |a_j - b_j|$$

the longest distance between the jth coordinates of any two points that belong to the set A, and denote by

$$j_{\max}(A) := \arg\max_{j=1,\ldots,d} V^{(j)}(A)$$

the coordinate direction in which this distance is longest.[11] Geometrically, $V^{(1)}, \ldots, V^{(d)}$ represent the lengths of the edges of the smallest axis-aligned hyperrectangle containing A, and $V^{(j_{\max})}$ is the longest edge of this hyperrectangle.

Since $\operatorname{diam}(A) \leq \sqrt{d}\, V^{(j_{\max}(A))}(A)$ for any set A, it is sufficient to show that $V^{(j_{\max})}$ of the intersection of the leaf node that contains the query point and the hypercube $[-M, M]^d$ converges to zero in probability. To this end, the following lemma gives a lower bound on the probability that for any node R, the length of this intersection in its longest coordinate direction $j_{\max}$ decreases at least by a factor of $3/4$ in at least one of the two child nodes when R is split by a randomly oriented hyperplane as described in Algorithm 3. Fig. 3 illustrates the lemma and its proof. Note that this lower bound does not depend on the training data but only on the i.i.d. random vectors.

Lemma 12 *Let R be a node of an RP tree. Denote its child nodes by R_{left} and R_{right}, and the intersections of these nodes with the hypercube $[-M, M]^d$ by R', R'_{left}, and R'_{right}, respectively. Further, for any $j = 1, \ldots, d$, denote $V^{(j)} := V^{(j)}(R')$, $V^{(j)}_{\text{left}} := V^{(j)}(R'_{\text{left}})$, and $V^{(j)}_{\text{right}} := V^{(j)}(R'_{\text{right}})$. It holds for the coordinate direction $j' := j_{\max}(R')$ that*

$$P\left\{V^{(j')}_{\text{left}} \leq \frac{3}{4}V^{(j')} \text{ or } V^{(j')}_{\text{right}} \leq \frac{3}{4}V^{(j')}\right\} \geq F\left(\frac{1}{4d-3}\right),$$

where F is the c.d.f. of a random variable following the beta distribution $\text{Beta}\left(\frac{d-1}{2}, \frac{1}{2}\right)$.

Proof If $R \cap [-M, M]^d = \emptyset$, then the claim holds trivially. When $R \cap [-M, M]^d \neq \emptyset$, by scale and translation invariance, we can assume w.l.o.g. that the set R' is located so that the smallest hyperrectangle containing it, denoted by U, is centered at the origin; that $j' = j_{\max}(R') = 1$, i.e., that the longest edge of U is aligned with the first coordinate axis which we call the x-axis or the *horizontal axis*; and that this edge has unit length, $V^{(1)} = 1$. The claim of the theorem now simplifies to

$$P\left\{V^{(1)}_{\text{left}} \leq 3/4 \text{ or } V^{(1)}_{\text{right}} \leq 3/4\right\} \geq F\left(\frac{1}{4d-3}\right).$$

The hyperplane that splits node R into the two child nodes is defined by its normal vector, given by the random vector $Z = (Z_1, \ldots, Z_d) \sim N_d(0, I)$, where we have dropped the index l for brevity, and the median of the projections of the training set points of R onto Z. Our strategy is to show that

$$P\left\{V^{(1)}_{\text{left}} \leq 3/4 \text{ and } V^{(1)}_{\text{right}} \leq 3/4\right\} \geq F\left(\frac{1}{4d-3}\right) \tag{18}$$

11. In what follows, we will drop the argument A and denote the lengths by $V^{(j)}$ whenever the set in question is clear from the context.

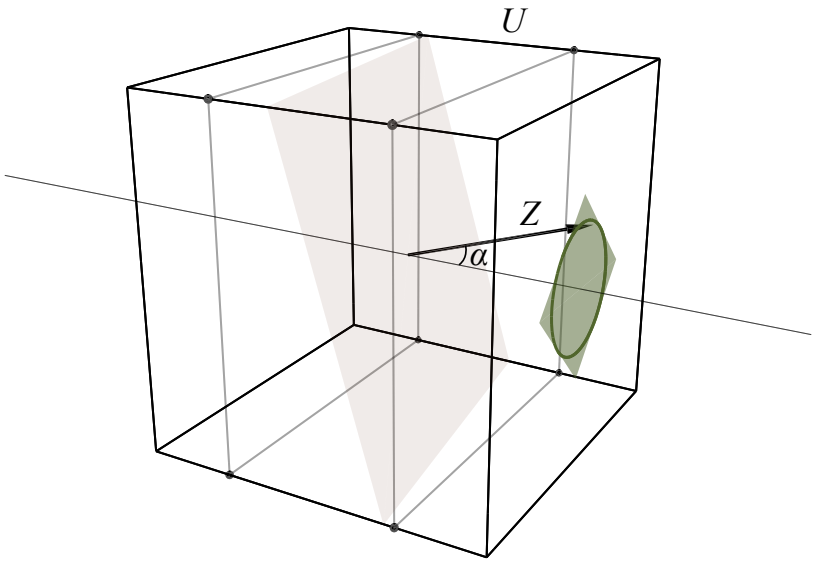

Figure 3: An illustration of the proof strategy of Lemma 12. Pictured is the worst-case scenario where all the edges of the smallest hyperrectangle U containing the set $R \cap [-M, M]^d$ have unit length, i.e., U is a hypercube. If the splitting hyperplane passes through the center of U and intersects with all the edges of U that are parallel to the x-axis in their middle half, then both $V_{\text{left}}^{(1)} \leq 3/4$ and $V_{\text{right}}^{(1)} \leq 3/4$. This event amounts to the random vector Z (that is the normal of the splitting hyperplane) intersecting with a side of U that is orthogonal to the x-axis inside of an L1 ball (in the 3D case, a diamond-shaped polygon), which can be further approximated by an L2 ball (in the 2D case, a circle).

in the special case where the splitting hyperplane passes through the origin. Since translating the splitting hyperplane away from the origin (while keeping its normal Z constant) reduces either $V_{\text{left}}^{(1)}$ or $V_{\text{right}}^{(1)}$, verifying (18) for the origin-centered hyperplane proves the original claim.

Let $x_{\min}$ and $x_{\max}$ be the smallest and the largest x-coordinate values among the points where the centered splitting hyperplane intersects with lines that pass through the horizontal edges of the bounding hyperrectangle U. It suffices to show that $x_{\max} \leq 1/4$, since then by symmetry we also have $x_{\min} \geq -1/4$, and thus both $V_{\text{left}}^{(1)} \leq 3/4$ and $V_{\text{right}}^{(1)} \leq 3/4$.

Denote by $u_1, \ldots, u_{2^{d-1}}$ the points $(1/4, \pm V^{(2)}/2, \ldots, \pm V^{(d)}/2)$, i.e., the points whose x-coordinate is $1/4$, and the last $d-1$ components match the coordinates of the corners of the hyperrectangle U. It is easy to see that $x_{\max} \geq 1/4$ if and only if all these points are on the same side of the centered splitting hyperplane. This holds if and only if

$$\operatorname{sign}\left(\langle u_1, Z\rangle\right) = \cdots = \operatorname{sign}\left(\langle u_{2^{d-1}}, Z\rangle\right),$$

or equivalently, by expanding the dot product, if and only if

$$\frac{Z_1}{4} \pm \frac{Z_2 V^{(2)}}{2} \pm \cdots \pm \frac{Z_d V^{(d)}}{2} \geq 0$$

or

$$\frac{Z_1}{4} \pm \frac{Z_2 V^{(2)}}{2} \pm \cdots \pm \frac{Z_d V^{(d)}}{2} < 0$$

for all permutations of the signs. This is equivalent to the condition

$$|Z_1| > 2 \sum_{j=2}^{d} V^{(j)} |Z_j|.$$

Since $V^{(j)} \leq V^{(1)} = 1$ for all $j = 2, \ldots, d$, we have

$$P\left\{ |Z_1| > 2 \sum_{j=2}^{d} V^{(j)} |Z_j| \right\} \geq P\left\{ |Z_1| > 2 \sum_{j=2}^{d} |Z_j| \right\}. \tag{19}$$

When $d = 2$, it follows from the rotational symmetry of the standard normal distribution and an elementary trigonometric argument that

$$P\{|Z_1| > 2|Z_2|\} = \frac{2 \arctan \frac{1}{2}}{\pi}.$$

When $d > 2$, the probability on the right hand side of (19) cannot be written in a closed form but we will provide a lower bound for it. Denote by $Z_{>1} := (Z_2, \ldots, Z_d)$ the vector of the last $d - 1$ components of the random vector Z. Since

$$2 \sum_{i=2}^{d} |Z_i| = 2 \, ||Z_{>1}||_1 \leq 2\sqrt{d-1} \, ||Z_{>1}||_2,$$

it holds that

$$\begin{aligned} P\left\{ |Z_1| > 2 \sum_{i=2}^{d} |Z_i| \right\} &\geq P\left\{ \frac{|Z_1|}{||Z_{>1}||_2} > 2\sqrt{d-1} \right\} \\ &= P\left\{ \frac{||Z_{>1}||_2}{|Z_1|} < \frac{1}{2\sqrt{d-1}} \right\} \\ &= P\left\{ \alpha < \arctan \frac{1}{2\sqrt{d-1}} \right\} \\ &= P\left\{ \sin \alpha < \frac{1}{\sqrt{4d-3}} \right\}, \end{aligned}$$

where $\alpha = \arctan \frac{||Z_{>1}||_2}{|Z_1|} = \arcsin \frac{||Z_{>1}||_2}{||Z||_2}$ is the smaller angle between the random vector Z and the first coordinate axis. By the rotational symmetry of the standard normal distribution, this probability is given by the area of the hyperspherical cap of a d-dimensional unit

sphere defined by the angle α (denoted by $A^{(d)}_{\alpha^*}$, where $\alpha^* := \arcsin \frac{1}{\sqrt{4d-3}}$), divided by the area of the hemisphere of the d-dimensional unit sphere (denoted by $A^{(d)}/2$):

$$\begin{aligned} P\left\{\sin\alpha < \frac{1}{\sqrt{4d-3}}\right\} &= \frac{A^{(d)}_{\alpha^*}}{A^{(d)}/2} \\ &= \frac{(A^{(d)}/2) I_{\sin^2\alpha^*}\left(\frac{d-1}{2}, \frac{1}{2}\right)}{A^{(d)}/2} \\ &= I_{\frac{1}{4d-3}}\left(\frac{d-1}{2}, \frac{1}{2}\right) \\ &= F\left(\frac{1}{4d-3}\right), \end{aligned}$$

where the area of $A^{(d)}_{\alpha^*}$ is given by Li (2011), and $I_x(a,b)$ denotes the regularized incomplete beta function that is also the value of the c.d.f. of the distribution $\mathrm{Beta}(a,b)$ evaluated at the point x. ■

We are now in a position to show that the leaf diameter of an RP tree converges to zero in probability. The following theorem verifies the conditions of Theorem 6 and thus proves consistency of RP trees for ANN search.

Theorem 13 *Assume that for the height of a random projection tree holds that $\ell \to \infty$ when $n \to \infty$. Then, for any $M > 0$, the leaf diameter constrained to within $[-M, M]^d$ converges to zero in probability, i.e.,*

$$P\{\mathrm{diam}(R_{q(X)} \cap [-M, M]^d > \delta\} \to 0$$

for any $\delta > 0$ as $n \to \infty$.

Proof Since nodes cannot grow in size when they are split, it suffices to show the claim for the partition induced by any intermediate level $\tilde{\ell} \leq \ell$. To this end, define $\ell' := \min(\ell, \lfloor \log_2 \log_2 n \rfloor)$. Denote the nodes of the ℓ'th level of a tree by $R_1, \ldots, R_{2^{\ell'}}$, and the maximum coordinate-wise diameters of their intersections with the hypercube $[-M, M]^d$ by $W_1, \ldots, W_{2^{\ell'}}$, where $W_l := V^{(j_{\max})}(R_l \cap [-M, M]^d)$ for each $l = 1, \ldots, 2^{\ell'}$. Since $\mathrm{diam}(A) \leq \sqrt{d}\, V^{(j_{\max})}(A)$ for any set A, to prove that $\mathrm{diam}(R_{q(X)})$ converges to zero in probability, it suffices to show that for any $\delta > 0$, $P\{W_{q(X)} > \delta\} \to 0$ when $n \to \infty$.

Choose $\epsilon > 0$ and denote the event that no node (again, at the ℓ'th level of a tree) has a probability mass larger than $(1+\epsilon)/2^{\ell'}$ by

$$G := \bigcap_{l=1}^{2^{\ell'}} \left\{\mu(R_l) \leq \frac{1+\epsilon}{2^{\ell'}}\right\},$$

where $\mu(A) := P\{X \in A\}$ is the probability distribution of X for any measurable set A. As in the proof of Theorem 11, our strategy is to first handle this case where all the nodes contain an approximately equal probability mass, and then bound the probability of the

event G^C by applying the Vapnik-Chervonenkis inequality to show the uniform convergence of the empirical distribution of X to its true distribution in the class of nodes of a random projection tree of height ℓ', i.e., in the class of convex polytopes in $\mathbb{R}^d$ that have ℓ' faces. To this end, we partition the probability of the event of interest as

$$P\{W_{q(X)} > \delta\} \leq \underbrace{P\{W_{q(X)} > \delta, G\}}_{I} + \underbrace{P(G^C)}_{II},$$

and bound both of these terms separately.

Term I: It follows from Lemma 10 that

$$P\{W_{q(X)} > \delta, G\} \leq \frac{E\left[\mathbb{1}_G W_{q(X)}\right]}{\delta}.$$

Therefore, it suffices to bound $E\left[\mathbb{1}_G W_{q(X)}\right]$, for which we have

$$\begin{aligned} E\left[\mathbb{1}_G W_{q(X)}\right] &= E\left[\mathbb{1}_G E\left[W_{q(X)} \,|\, Z, D_n\right]\right] \\ &= E\left[\mathbb{1}_G \sum_{l=1}^{2^{\ell'}} \mu(R_l) W_l\right] \\ &\leq \frac{1+\epsilon}{2^{\ell'}} E\left[\sum_{l=1}^{2^{\ell'}} W_l\right], \end{aligned}$$

where the first equality holds because $\mathbb{1}_G$ is independent of X, and the last inequality follows from the definition of G.

Consider any node R and its child nodes by R_{left} and R_{right}; denote the intersections of these node with the hypercube $[-M, M]^d$ by R', R'_{left}, and R'_{right}, respectively. Lemma 12 provides a lower bound for the probability of the event that the length of either R'_{left} or R'_{right} is less than or equal to the length of R' in the coordinate direction where the original set R' is longest. More precisely, the probability of this event is greater than or equal to $F(\frac{1}{4d-3})$, where F is c.d.f. of the random variable $B \sim \text{Beta}(\frac{d-2}{2}, \frac{1}{2})$. Observe that $C_d := F(\frac{1}{4d-3}) \in (0, 1)$ is a constant that depends only on the data dimension.

It follows from the rotational invariance of the standard normal distribution that the random vectors Z and $-Z$ have an equal density. Hence, we can assume that the indices of the left and the right child node are assigned uniformly at random. Therefore, at any node R_l on the path from the root, the probability that the diameter of $R'_l := R_l \cap [-M, M]^d$ in the coordinate direction where it is longest decreases at least by a factor of $3/4$ is greater than or equal to $C_d/2$. Clearly, if this event occurred d times on any path from the root to node R_l, then the maximum diameter of the node in any coordinate direction would shrink at least by a factor of $3/4$, i.e., $W_l \leq \frac{3}{4} \cdot 2M$.

For any node R_l at ℓ'th level of a tree, denote by K_n the number of times the coordinate-wise diameter of R'_l in the coordinate direction where it is longest has shrunk at least by a factor of $3/4$ on the path from the root to R_l. Now, we have $EW_l \leq 2M \cdot E\left[(3/4)^{\lfloor \frac{K_n}{d} \rfloor}\right]$, where K_n is a random variable that follows the binomial distribution $\text{Bin}\,(\ell', p)$ for whose

parameter p holds that $p \geq C_d/2$. Therefore,

$$\frac{1+\epsilon}{2^{\ell'}} E\left[\sum_{l=1}^{2^{\ell'}} W_l\right] = \frac{1+\epsilon}{2^{\ell'}} \sum_{l=1}^{2^{\ell'}} EW_l \leq 2M(1+\epsilon) E\left[(3/4)^{\lfloor \frac{K_n}{d} \rfloor}\right] \to 0$$

when $n \to \infty$ (since by the assumption then also $\ell' \to \infty$).

Term II: Let $\mathcal{A}$ be the class of all half-spaces in $\mathbb{R}^d$ and let $\mathcal{A}_{\ell'}$ be the class of all convex polytopes in $\mathbb{R}^d$ that have ℓ' faces. It holds for the shatter coefficient of the class $\mathcal{A}$ that $s(\mathcal{A}, n) \leq 2(n-1)^d + 2$ (see, e.g., Corollary 13.1 by Devroye et al. (1996, p. 223)). Since each $A \in \mathcal{A}_{\ell'}$ can be written as $A = \bigcap_{l=1}^{\ell'} A_l$ where $A_l \in \mathcal{A}$ for each $l = 1, \ldots, \ell'$, it follows from Theorem 13.5 (iii) by Devroye et al. (1996, p. 219) that

$$s(\mathcal{A}_{\ell'}, n) \leq s(\mathcal{A}, n)^d \leq n^{d\ell'}$$

for a large enough n. If $n \geq 2 \cdot \frac{\log_2 n}{\epsilon} \geq 2 \cdot \frac{2^{\ell'}}{\epsilon}$, then $\frac{1}{n} \leq \frac{1}{2} \cdot \frac{\epsilon}{2^{\ell'}}$. This means that for large enough n, we have

$$\begin{aligned}
P(G^C) &= P\left\{\exists l \text{ s.t. } \mu(R_l) > \frac{1+\epsilon}{2^{\ell'}}\right\} \\
&= P\left\{\exists l \text{ s.t. } \mu(R_l) - \left(\frac{1}{2^{\ell'}} + \frac{1}{n}\right) > \frac{\epsilon}{2^{\ell'}} - \frac{1}{n}\right\} \\
&\leq P\left\{\exists l \text{ s.t. } \mu(R_l) - \mu_n(R_l) > \frac{\epsilon}{2^{\ell'}} - \frac{1}{n}\right\} \\
&\leq P\left\{\exists l \text{ s.t. } \mu(R_l) - \mu_n(R_l) > \frac{\epsilon}{2^{\ell'+1}}\right\} \\
&\leq P\left\{\sup_{R \in \mathcal{A}_{\ell'}} |\mu(R) - \mu_n(R)| > \frac{\epsilon}{2^{\ell'+1}}\right\} \\
&\leq 8 s(\mathcal{A}_{\ell'}, n) \exp\left\{-\frac{n\epsilon^2}{128 \cdot 2^{2\ell'}}\right\} \\
&\leq 8 n^{d\ell'} \exp\left\{-\frac{n\epsilon^2}{128 \cdot 2^{2\ell'}}\right\}
\end{aligned} \tag{20}$$

when $n \to \infty$ since $\ell' \leq \log_2 \log_2 n$ by definition. The first inequality on the right hand side of (17) follows because for the empirical measure—denoted by $\mu_n(A) := \frac{1}{n}\sum_{i=1}^n \mathbb{1}_A(X_i)$ for any measurable set A—of any leaf R_l it holds that $\mu_n(R_l) \leq \frac{1}{2^{\ell'}} + \frac{1}{n}$. The fourth inequality follows from the Vapnik-Chervonenkis inequality (Vapnik and Chervonenkis, 1971); we use the version presented in Theorem 12.5. by Devroye et al. (1996, p. 197-198). ■

7.3.2 The sparse case

In a *sparse RP tree*, proposed by Hyvönen et al. (2016), the non-zero coordinate directions of a random vector $Z = (Z_1, \ldots, Z_d)$ are first selected by sampling the binary random

variables $B_1, \ldots, B_d$ from a Bernoulli(a)-distribution with a sparsity parameter $0 < a \leq 1$. The components of the random vector are then generated as

$$Z_j \,|\, B_j \sim \begin{cases} N(0,1), & \text{if } B_j = 1 \\ 0, & \text{otherwise} \end{cases} \tag{21}$$

for each $j = 1, \ldots, d$. Like a dense RP tree, a sparse RP tree is grown as in Algorithm 3.

Another version of a sparse RP tree can be obtained by sampling $\lceil ad \rceil$ coordinate directions uniformly at random from the set $\{1, \ldots, d\}$ without replacement, and letting $B_j = 1$ for these directions while letting $B_j = 0$ for the remaining $d - \lceil ad \rceil$ directions; the random vector Z is then generated as in (21). There is little practical difference between these two randomization methods when the data dimension d is high—as it usually is in the applications of ANN search[12].

A result analogous to Lemma 12 also holds for sparse RP trees.

Lemma 14 *Let R be a node of the sparse RP tree where the indicators for non-zero coordinate directions are sampled as $B_1, \ldots, B_d \sim$ Bernoulli(a) for any $0 < a \leq 1$. Denote the child nodes of R by R_{left} and R_{right}, and the intersections of these nodes with the hypercube $[-M, M]^d$ by R', R'_{left}, and R'_{right}, respectively. Further, denote $V^{(j)} := V^{(j)}(R')$, $V^{(j)}_{\text{left}} := V^{(j)}(R'_{\text{left}})$, and $V^{(j)}_{\text{right}} := V^{(j)}(R'_{\text{right}})$. It holds for the coordinate direction $j := j_{\max}(R')$ that*

$$P\left\{ V^{(j)}_{\text{left}} \leq \frac{3}{4} V^{(j)} \text{ or } V^{(j)}_{\text{right}} \leq \frac{3}{4} V^{(j)} \right\} \geq \frac{1}{2} a F\left(\frac{1}{4\lceil ad \rceil + 1} \right),$$

where F is the c.d.f. of a random variable following the beta distribution Beta$\left(\frac{d-1}{2}, \frac{1}{2}\right)$.

Proof Denote the event of interest by

$$A := \left\{ V^{(j)}_{\text{left}} \leq \frac{3}{4} V^{(j)} \text{ or } V^{(j)}_{\text{right}} \leq \frac{3}{4} V^{(j)} \right\}.$$

The jth coordinate direction must be among the non-zero coordinate directions of the random vector for the length of the node in that direction to be affected. The probability of this event is $P\{B_j = 1\} = a$. Thus,

$$P(A) = P\{A, B_j = 1\} = a P\{A \,|\, B_j = 1\}.$$

We provide a lower bound for the conditional probability $P\{A \,|\, B_j = 1\}$ by considering the case in which the total number of non-zero coordinate directions, denoted by $B := \sum_{i=1}^{d} B_i$, is no more than $\lceil ad \rceil + 1$:

$$\begin{aligned} P\{A \,|\, B_j = 1\} &\geq P\{A, B \leq \lceil ad \rceil + 1 \,|\, B_j = 1\} \\ &= P\{A \,|\, B \leq \lceil ad \rceil + 1, B_j = 1\} P\{B \leq \lceil ad \rceil + 1 \,|\, B_j = 1\}. \end{aligned} \tag{22}$$

12. In the experimental section of this article, we use the first version.

First, we provide a lower bound for the second term on the right hand side of (22). Denote by $B_{-j} = \sum_{i \neq j} B_i$ the sum of all but the jth indicator for the non-zero coordinate directions. Since $B_1, \ldots, B_d$ are independent,

$$\begin{aligned} P\{B \leq \lceil ad \rceil + 1 \,|\, B_j = 1\} &= P\{B_{-j} \leq \lceil ad \rceil\} \\ &\geq P\{B_{-j} \leq \lceil a(d-1) \rceil\} \geq \frac{1}{2}. \end{aligned}$$

The last inequality follows because $B_{-j} \sim \mathrm{Bin}(d-1, a)$, and for the median $m(X)$ of the binomially distributed random variable X holds that $m(X) \leq \lceil EX \rceil$ (see Kaas and Buhrman, 1980).

Finally, the proof is completed by applying the argument of Lemma 12 to the $(\lceil ad \rceil + 1)$-dimensional subspace containing the non-zero coordinate directions to obtain the lower bound

$$P\{A \,|\, B \leq \lceil ad \rceil + 1, B_j = 1\} \geq F\left(\frac{1}{4\lceil ad \rceil + 1}\right)$$

for the first term on the right hand side of (22). ■

We can also prove the consistency of sparse RP trees as in Theorem 13 by verifying the condition Theorem 6, i.e., verifying that the leaf diameter converges to zero as the sample sizes grows to infinity.

Theorem 15 *Assume that for the height of a sparse RP tree where the non-zero coordinate directions are sampled from* Bernoulli(a) *for any* $0 < a \leq 1$ *holds that* $\ell \to \infty$ *when* $n \to \infty$. *Then, for any* $M > 0$, *the leaf diameter constrained to within* $[-M, M]^d$ *converges to zero in probability, i.e.,*

$$P\{\mathrm{diam}(R_{q(X)} \cap [-M, M]^d > \delta\} \to 0$$

for any $\delta > 0$ *as* $n \to \infty$.

Proof As the proof of Theorem 13 but apply Lemma 14 instead of Lemma 12. ■

Remark 16 *The above results (Lemma 14 and Theorem 15) apply to the variant of sparse RP trees where the non-zero coordinate directions are sampled from* Bernoulli(a) *for any* $0 < a \leq 1$. *In the variant of an sparse RP tree where the* $\lceil ad \rceil$ *non-zero coordinate directions are sampled uniformly at random without replacement, a lower bound corresponding to Lemma 14 and Lemma 12 can be obtained as* $aF\left(\frac{1}{4\lceil ad \rceil - 3}\right)$. *The consistency of this variant can then be established as in Theorem 13 and Theorem 15 by applying this lower bound.*

8. Experiments

In this section, we present empirical results validating the utility of the proposed framework. In particular, we compare the natural classifier to the earlier candidate set selection methods (lookup search and voting search) when combined with different types of unsupervised trees that have been widely used for ANN search. Specifically, we use ensembles of randomized

k-d trees (Friedman et al., 1976; Silpa-Anan and Hartley, 2008), random projection (RP) trees (Dasgupta and Freund, 2008; Dasgupta and Sinha, 2015; Hyvönen et al., 2016), and principal component (PCA) trees (Sproull, 1991; Jääsaari et al., 2019) (see Sec. 7.3 and Appendix B for detailed descriptions of these data structures).

Another consequence of the multilabel formulation of Sec. 3.1 is that it enables using any established multilabel classifier for ANN search. To demonstrate this concretely, we train a random forest consisting of standard multilabel classification trees (trained under the PAL reduction (Reddi et al., 2019) by using multinomial log-likelihood as a split criterion) and use it as an index structure for ANN search; it turns out that the fully supervised classification trees have an improved performance compared to the earlier unsupervised trees on some—but, curiously, not on all—data sets.

We follow a standard ANN search performance evaluation setting (Aumüller et al., 2019a; Li et al., 2019) by using the corpus as the training set, searching for $k = 10$ nearest neighbors in Euclidean distance, and measuring performance by evaluating average recall and query time over the test set of 1000 points. We use four benchmark data sets: Fashion ($m = 60000$, $d = 784$), GIST ($m = 1000000$, $d = 960$), Trevi ($m = 101120$, $d = 4096$), and STL-10 ($m = 100000$, $d = 9216$). All the algorithms are implemented in C++ and run using a single thread. We tune the hyperparameters by grid search and plot the Pareto frontiers of the optimal hyperparameters. Further details of the experimental setup are found in Appendix A. The code used to produce the experimental results is attached as supplementary material and can also be found at `https://github.com/vioshyvo/a-multilabel-classification-framework`.

8.1 Comparison of candidate set selection methods

The candidate set selection method proposed in this article is the natural classifier (Algorithm 2). The earlier methods are lookup search and voting search (Hyvönen et al., 2016; Jääsaari et al., 2019). We first compare the averages of precision

$$\text{Prec}(S(x)) := \frac{|NN_k(x) \cap S(x)|}{|S(x)|}$$

and recall (2) of the candidate sets selected by the different methods. The results are shown in Fig. 4. We plot the Pareto frontiers with the optimal hyperparameters. Lookup search requires the largest candidate set to reach a given recall level in every case, and the natural classifier requires the smallest candidate set in every case, except for the RP trees on Fashion data set (and for the RP trees and PCA trees on the highest recall levels on GIST data set) where voting search reaches higher precision. Since the exact k-nn search time in the candidate set is approximately linear w.r.t. the candidate set size, a smaller candidate set at a given recall level leads to faster ANN queries at that recall level.

However, the query times depend also on the time it takes to select the candidate set. Thus, we follow the standard method (see, e.g., Aumüller et al., 2019a) of comparing the actual query times of different candidate set selection methods at different recall levels. The results are shown in Fig. 5 as the Pareto frontiers with the optimal hyperparameters (recall on the x-axis and query time on the y-axis). The results indicate that, as the discussion of Sec. 5 suggests, the natural classifier performs better than the earlier lookup-based methods for all types of trees; this finding holds consistently over all four data sets.

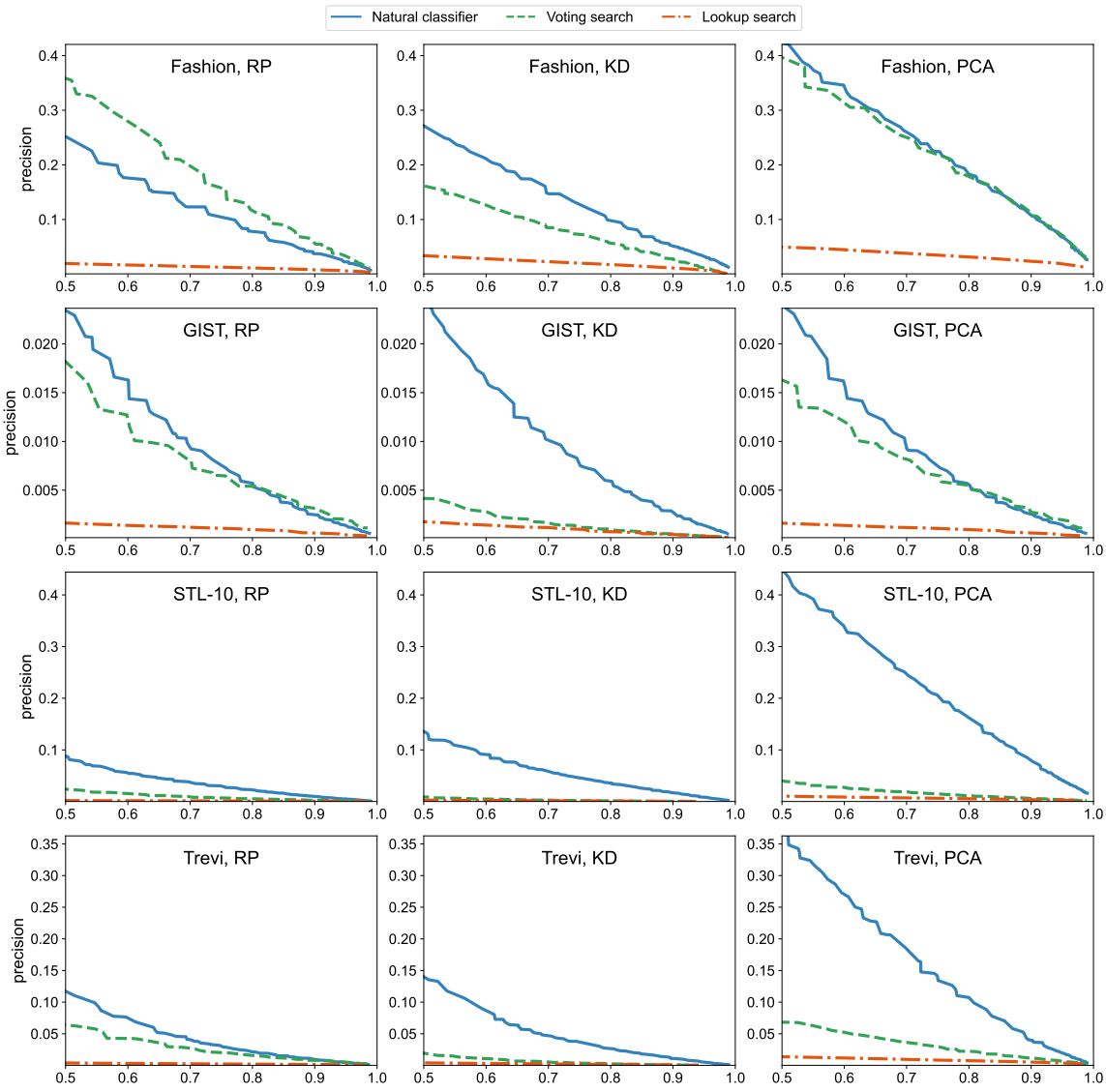

Figure 4: Recall (x-axis) vs. precision (y-axis) of ensembles of, RP, k-d, and PCA trees. The Pareto frontiers with the optimal hyperparameter combinations are shown. The solid blue line is the natural classifier proposed in this paper; the dashed green line is voting; and the dash-dotted red line is lookup search. The natural classifier has the highest precision at a given recall level in every case, expect with RP trees on the Fashion data set and on high recall levels on the GIST data set where voting search has the highest precision at a given recall level. Lookup search has the lowest precision at a given recall level in every case.

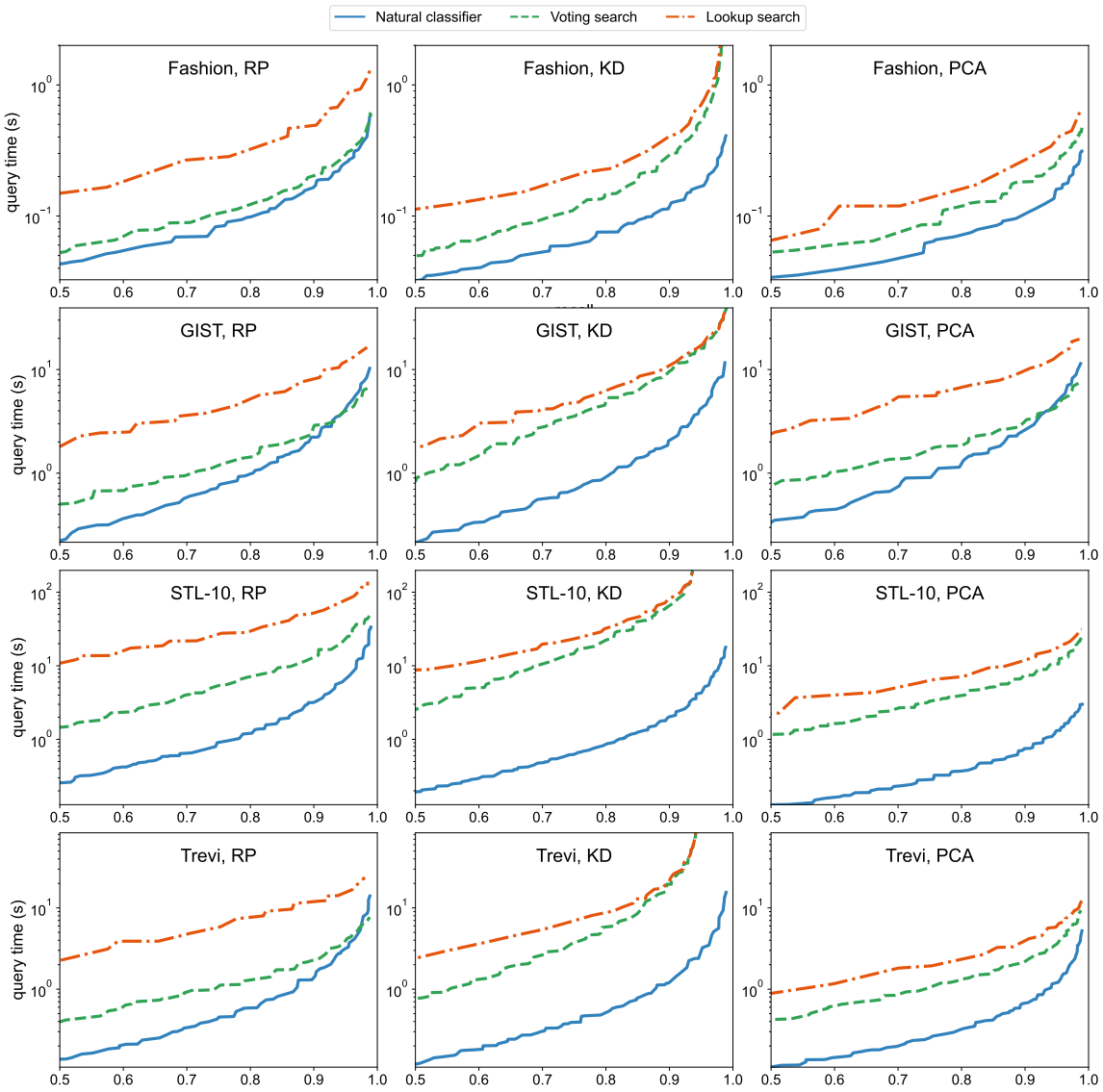

Figure 5: Recall (x-axis) vs. query time (y-axis; on the log scale) of ensembles of, RP, k-d, and PCA trees. The solid blue line is the natural classifier proposed in this paper; the dashed green line is voting; and the dash-dotted red line is lookup search. The natural classifier is the fastest method and the lookup search is the slowest method for each tree type.

8.2 Comparison of tree types

We compare the aforementioned ensembles of unsupervised (RP, KD, and PCA) trees and the random forest consisting of supervised classification trees (RF); for all four tree types the candidate set is selected by the natural classifier (Algorithm 2). The results are shown

in Table 2. Since the random forest (RF) leverages supervised information to learn the trees, we would expect that it is the fastest tree-based method. Indeed, this is the case on Fashion and GIST. However, on STL-10 and Trevi, the unsupervised PCA tree is the fastest method. We hypothesize that this is because of the high dimensionality of STL-10 and Trevi: standard supervised classification trees employed by the random forest are restricted to axis-aligned splits, whereas PCA trees—although they use an unsupervised split criterion—can find more informative oblique split directions. An interesting topic for future work would be to apply supervised classification trees that can utilize oblique split directions.

Table 2: Query times (seconds / 1000 queries) at different recall levels for the different tree types. The fastest method in each case is typeset in boldface.

data set	R (%)	PCA	KD	RP	RF
Fashion	80	0.075	0.076	0.099	**0.063**
	90	0.111	0.126	0.172	**0.095**
	95	0.163	0.171	0.261	**0.146**
GIST	80	1.330	0.958	1.009	**0.705**
	90	2.942	2.286	2.226	**1.530**
	95	5.641	4.451	4.598	**3.253**
STL-10	80	**0.382**	0.872	1.211	0.756
	90	**0.756**	2.126	3.248	1.774
	95	**1.315**	4.376	7.330	3.654
Trevi	80	**0.330**	0.543	0.591	0.582
	90	**0.684**	1.464	1.468	1.234
	95	**1.212**	3.244	3.289	2.350

9. Conclusion

We establish a general theoretical framework for ANN search by formulating the candidate set selection as a multilabel learning task. The empirical results validate the utility of the proposed framework: the natural classifier derived directly from the multilabel formulation is a strict improvement over the earlier candidate set selection methods. In addition, we provide a sufficient condition that guarantees the consistency of a partitioning classifier for ANN search. We verify this condition for chronological k-d trees and (both dense and sparse) random projection trees: given enough training data, these trees retrieve the candidate set that contains all the k nearest neighbors of the query point and no other corpus points.

9.1 Limitations

Supervised ANN search methods typically have longer pre-processing times compared to the unsupervised methods. This is, first, because they require the computation of the true

nearest neighbors $\{y_i\}_{i=1}^n$ of the training set points $\{x_i\}_{i=1}^n$; and, second, because supervised index structures are often slower to build compared to their unsupervised counterparts (see Appendix C.1). If fast index construction is required, the second issue can be mitigated by learning the trees in an unsupervised fashion, and then using them as partitioning classifiers as described in Sec. 4. The experiments of Sec. 8 suggest that the candidate set selection method has a more pronounced effect on the performance than the tree type.

9.2 Future research directions

An important consequence of the proposed framework is that it enables the use of any type of classifier as an index structure for ANN search. In particular, the gradient boosted trees model (Friedman, 2001) appears a promising candidate since it is often more accurate than the random forest that we applied in this article. *Extreme classification* models, including tree-based models (Agrawal et al., 2013; Prabhu and Varma, 2014; Jain et al., 2016), sparse linear models (Babbar and Schölkopf, 2017, 2019; Yen et al., 2017), and embedding-based neural networks (Guo et al., 2019), are also promising model candidates for ANN search since they are specifically tailored to multilabel classification problems with extremely large label spaces.

The proposed multilabel formulation enables analyzing ANN search in the statistical learning framework, thus opening up multiple theoretical research questions: (1) Can we establish a sufficient condition for *strong* consistency? (2) Can we prove consistency of more adaptive partitioning classifiers, such as PCA trees or classification trees? (3) Can we establish faster than logarithmic convergence rates? The last question is especially interesting, since prediction times of trees are logarithmic: a positive answer would theoretically justify decreasing query times by increasing the training set size.

Acknowledgments

This work has been supported by the Research Council of Finland (grants #345635, #311277, #313857, and the Finnish Center for AI (FCAI)) as well as the Helsinki Institute for Information Technology (HIIT). The authors wish to thank the Finnish Computing Competence Infrastructure (FCCI) for supporting this project with computational and data storage resources.

Appendix A. Experimental setup

We have included the code used to produce the experimental results in the companion repository of this article[13]. In this section, we describe the computing environment, data sets, and hyperparameter tuning process used in the experiments.

A.1 Computing environment

The experiments were ran on a machine with two Xeon E5-2680 v4 2.4GHz processors, 256GB RAM and CentOS 7 as the operating system. All queries were ran using only a single thread. The algorithms and test code were written in C++14 and compiled using GCC 5.4.0 with the optimization flags `-Ofast` and `-march=native`.

A.2 Data sets

Table 3 contains the specifications of the data sets. We used four publicly available and commonly used benchmark data sets (Fashion[14], GIST[15], STL-10[16], and Trevi[17]) consisting of raw or preprocessed images. We randomly divided the original data sets into the corpus, the validation set ($n_{\text{validation}} = 1000$), and the test set ($n_{\text{test}} = 1000$). The corpus $\{c_i\}_{i=1}^{m}$ was used as a training set. Since in the previous benchmarks for ANN search (Aumüller et al., 2019a; Li et al., 2019) the problem is not considered in the machine learning setting, they do not use a distinct test set, but present the optimal results on the validation set. To follow this standard practice, we also present the results on the validation set, but note that the results were stable between the validation and test sets; there was some random variability, but we observed no signs of overfitting to the validation set.

Table 3: Data sets used in the experiments

Data set	corpus size m	dimension d
Fashion	58000	784
GIST	100000	960
STL-10	98000	9216
Trevi	99120	4096

A.3 Hyperparameter settings

The exact hyperparameter grids used in the experiment can be found in the companion repository[18]. Here we describe the hyperparameter tuning process.

According to our initial experiments, the performance of each type of a tree was robust w.r.t. its sparsity/randomization parameter (the number of the uniformly at random chosen

13. https://github.com/vioshyvo/a-multilabel-classification-framework
14. https://github.com/zalandoresearch/fashion-mnist
15. http://corpus-texmex.irisa.fr
16. https://cs.stanford.edu/~acoates/stl10
17. http://phototour.cs.washington.edu/patches
18. https://github.com/vioshyvo/a-multilabel-classification-framework

coordinate directions $a \in \{1, \ldots, d\}$ from which the optimal split direction was chosen for multiclass classification trees; the dimensionality a of the random subspace in which first principal component was approximated for PCA trees; the expected number a of the non-zero components in the random vectors onto which the node points are projected in RP trees; and the number o of the highest variance directions of the node points from which the split direction was chosen uniformly at random in k-d trees). Therefore, we kept these parameters fixed in the final experiments: for multiclass classification trees and the PCA trees we used the value $a = \lceil \sqrt{d} \rceil$; for the RP trees the value $a = 1/\sqrt{d}$; and for the k-d trees the value $o = 5$. Further, we set the learning rate of the iterative PCA algorithm in PCA trees to $\gamma = 0.01$ and the maximum number of iterations to $t = 20$.

For the recall levels on the range [0.5, 0.99] considered in the article, the optimal numbers of trees T were generally on the range [5, 200], the optimal depths of the trees were on the range [10, 15], and the optimal values of the threshold parameter τ were on the range [1, 20] for PCA, RP, and k-d trees that use the raw counts as score function values. For multiclass classification trees that use the probability estimates (7) to select the candidate set, the optimal values of the threshold parameter τ were on the range [0.00001, 0.005]. We observed that using a value $k' > k$ to learn the trees sometimes improved performance. We tested values $k' \in \{10, 50, 100\}$ for learning the trees, with $k' = 10$ or $k' = 50$ usually being the optimal parameter value when $k = 10$.

For the other algorithms, we used the same hyperparameters as in ANN-benchmarks[19] as a starting point, and in many cases used even larger grids to ensure that the optimal hyperparameter settings were found.

Appendix B. Data structures

In this section we review the types of trees considered in this article (see B.1–B.4 below). Random projection, PCA, k-d trees and chronological k-d trees have been widely used for ANN search (see, e.g., (Silpa-Anan and Hartley, 2008; Muja and Lowe, 2014; Dasgupta and Sinha, 2015; Jääsaari et al., 2019)), whereas the multiclass classification tree is a standard data structure for classification. For completeness, we include the full descriptions of the algorithms[20] here.

We begin by motivating the natural classifier (9) from the point of view of the multilabel problem reductions (see Reddi et al. (2019); Menon et al. (2019)). First note that since the label set $L(X)$ is a deterministic function of the query point X—which means that the conditional label probabilities $\eta_1(x), \ldots, \eta_m(x)$ are all equal to either 0 or 1—the Bayes classifier for 0-1 loss is obtained by thresholding these label probabilities by any $\tau \in [0, 1)$. As a corollary, the same holds also for other less strict loss functions, such as precision, recall, and Hamming loss. This justifies following the common practice of estimating the conditional label probabilities by reducing the original multilabel classification problem into a series of binary or multiclass classification problems (Menon et al., 2019).

In the *pick-all-labels* (PAL) (Reddi et al., 2019) reduction, a separate multiclass observation is created from each positive label, whereas in in the *one-versus-all* (OVA) reduction, each of the m labels is modeled as an independent binary classification problem. Assume

19. `https://github.com/erikbern/ann-benchmarks/blob/master/algos.yaml`
20. The random projection tree is covered in the main article (see Sec. 7.3) and thus it is not discussed here.

that the query distribution is continuous. Then, almost surely $|L(x)| = k$, and thus the maximum likelihood estimates for the label probabilities under the PAL and OVA reductions—i.e., under the multinomial and binomial models, respectively—are $\hat{\theta}_{\text{PAL}} = \frac{1}{nk}\sum_{i=1}^{n} y_{ij}$ and $\hat{\theta}_{\text{OVA}} = \frac{1}{n}\sum_{i=1}^{n} y_{ij}$. The two estimates are proportional to each other, $\hat{\theta}_{\text{OVA}} = k\hat{\theta}_{\text{PAL}}$, and hence, given the partition $\mathcal{P}$, the parameter estimates of the natural classifier (9)—i.e., observed label proportions among the training set points in a given partition element—minimise the log-likelihood under both reductions.

Motivated by the above, we use the natural classifier (9) for prediction in combination with all tree types. When an ensemble of trees is used as a classifier, we compute the conditional label probability estimates of the ensemble as in (8) by averaging the contributions of the individual trees.

B.1 Chronological k-d tree

The chronological k-d tree (Bentley, 1975) was the first data structure proposed for speeding up nearest neighbor search. It rotates the split directions and uses the same split direction for all the nodes at one level of a tree. At the first level the training data is split at the median of the first coordinates of the data points. At the second level both nodes are split at the median of the second coordinates of the node points. At the $(d+1)$th level, the nodes are split again at the median of the first coordinates, and so on (see Algorithm 4). More adaptive version of the k-d tree that splits at the coordinate direction in which the node points have the highest variance was proposed by Friedman et al. (1976); we use a randomized version of this adaptive k-d tree (see Sec. B.3 and Algorithm 6) in the experiments of this article.

Algorithm 4 Grow a chronological k-d tree (Bentley, 1975)

Input: a set of node points X, current level ℓ', maximum height ℓ
Output: a node of a tree
procedure GROW-KD(X, ℓ', ℓ)
    if $\ell' = \ell$ **then**
        return X node as a leaf node
    $\hat{r} \leftarrow (\ell' \text{ modulo } d) + 1$
    $\hat{s} \leftarrow$ median of the $\hat{r}$th coordinates of the node points
    left $\leftarrow$ GROW-KD($\{x_i \in X : x_{i\hat{r}} \leq \hat{s}\}$, $\ell' + 1$, ℓ)
    right $\leftarrow$ GROW-KD($\{x_i \in X : x_{i\hat{r}} > \hat{s}\}$, $\ell' + 1$, ℓ)
    return (left, right, $\hat{r}$, $\hat{s}$) as an inner node

B.2 Multiclass classification tree

As discussed at the beginning of the section, the maximum likelihood estimates of the piecewise constant model under both the PAL and OVA reductions coincide in the special case of ANN search. Thus, in principle it would make no difference which one of these reductions we employed to learn the classification trees. However, in practice computation of the binomial likelihood requires keeping track of the contributions of the negative labels which is inconvenient when the label space is large. Thus, we employ the PAL reduction where

Algorithm 5 Grow a randomized multiclass classification tree

Input: a set of node points X, current depth, maximum depth ℓ, sparsity parameter a
Output: a node of a tree
procedure GROW(X, depth, $\ell, a = \lceil\sqrt{d}\rceil$)
    if depth $= \ell$ **then**
        return X as a leaf node
    $D \leftarrow a$ random unique dimensions from $\{1, \ldots, d\}$
    $(\text{maxgain}, \hat{r}, \hat{s}) \leftarrow (0, 0, 0)$
    $N \leftarrow |X|$
    for $r \in D$ **do**
        let $s_1 \leq s_2 \leq \cdots \leq s_N$ be the rth coordinate of points $x \in X$ sorted in ascending order
        for $s \in \{s_1, \ldots, s_N\}$ **do**
            $X_{\text{left}} \leftarrow \{x_i \in X : x_{ir} \leq s\}$; $N_{\text{left}} = |X_{\text{left}}|$
            $X_{\text{right}} \leftarrow \{x_i \in X : x_{ir} > s\}$; $N_{\text{right}} = |X_{\text{right}}|$
            for $j \in \{1, \ldots, m\}$ **do**
                $v_j^{(\text{left})} \leftarrow \sum_{x_i \in X_{\text{left}}} y_{ij}$
                $v_j^{(\text{right})} \leftarrow \sum_{x_i \in X_{\text{right}}} y_{ij}$
                $v_j \leftarrow \sum_{x_i \in X} y_{ij}$
                $\hat{\alpha}_j^{(\text{left})} := v_j^{(\text{left})} / (kN_{\text{left}})$
                $\hat{\alpha}_j^{(\text{right})} := v_j^{(\text{right})} / (kN_{\text{right}})$
                $\hat{\alpha}_j := v_j / (kN)$
            $\text{gain} \leftarrow \sum_{j=1}^m v_j^{(\text{left})} \log \hat{\alpha}_j^{(\text{left})} + \sum_{j=1}^m v_j^{(\text{right})} \log \hat{\alpha}_j^{(\text{right})} - \sum_{j=1}^m v_j \log \hat{\alpha}_j$
            if gain $>$ maxgain **then**
                $(\text{maxgain}, \hat{r}, \hat{s}) \leftarrow (\text{gain}, r, s)$
    if maxgain ≤ 0 **then**
        return X as a leaf node
    left $\leftarrow$ GROW($\{x_i \in X : x_{i\hat{r}} \leq \hat{s}\}$, depth + 1, ℓ, a)
    right $\leftarrow$ GROW($\{x_i \in X : x_{i\hat{r}} > \hat{s}\}$, depth + 1 ℓ, a)
    return $(\text{left}, \text{right}, \hat{r}, \hat{s})$ as an inner node

each positive label is modeled by a separate multiclass observation, and learn the standard multiclass classification trees by greedily maximising the multinomial log-likelihood (i.e., use the multiclass cross-entropy as a split criterion); see Algorithm 5 for details.

To grow a random forest, we randomize the multiclass classification trees by optimising the split point only in randomly chosen $a = \lceil\sqrt{d}\rceil$ coordinate directions at each node of a tree. We do not use bootstrap samples, but fit each tree to the original training data. To decrease learning time, we subsample 100 training points at each node (if node size > 100), and use only this subset to optimize the splits; we did not observe any negative impact on prediction performance.

Algorithm 6 Grow a randomized k-d tree (Silpa-Anan and Hartley, 2008)

Input: a set of node points X, current level, maximum height ℓ, the number of highest variances directions from which the split direction is sampled o
Output: a node of a tree
procedure GROW-KD(X, level, $\ell, o = 5$)
    if level $= \ell$ **then**
        return X as a leaf node
    $D \leftarrow$ set of o coordinate directions in which the node points have the highest variance
    $\hat{r} \leftarrow$ uniformly at random sampled dimension from the set D
    $\hat{s} \leftarrow$ median of $\hat{r}$th coordinates of the node points
    left $\leftarrow$ GROW-KD($\{x_i \in X : x_{i\hat{r}} \leq \hat{s}\}$, level + 1, ℓ, o)
    right $\leftarrow$ GROW-KD($\{x_i \in X : x_{i\hat{r}} > \hat{s}\}$, level + 1 ℓ, o)
    return (left, right, $\hat{r}, \hat{s}$) as an inner node

B.3 k-d tree

Algorithm 6 details the recursive algorithm for growing a randomized k-d tree. As in multiclass classification trees, the splits are restricted to the directions of the coordinate axes. In k-d trees (Friedman et al., 1976) the normal of the splitting hyperplane is chosen as the coordinate direction $r \in \{1, \ldots, d\}$ along which the node points have the highest variance. The split point $\hat{s}$ is chosen as the median of the $\hat{r}$th coordinate of the node points. To grow an ensemble of randomized trees, we use the randomization scheme proposed by Silpa-Anan and Hartley (2008): instead of splitting at the direction of the highest variance, we draw uniformly at random one of the o highest variance directions and use it as a split direction $\hat{r}$. We use the default value $o = 5$ recommended by Muja and Lowe (2014) for this hyperparameter.

B.4 PCA tree

In a PCA tree (Sproull, 1991), the projection direction at each node is the first principal component, i.e. the direction the node points have the greatest variance when projected onto. PCA trees are on the one hand slow to compute, as computing exact PCA is expensive, and on the other hand they are deterministic and thus multiple trees cannot be grown to boost accuracy.

To solve the first problem, McCartin-Lim et al. (2012) proposed an *approximate PCA tree*, which uses gradient descent updates to approximate the first principal component of the data at each node of the tree. To address the second problem, Jääsaari et al. (2019) proposed a *sparse approximate PCA tree*, which draws at each node of the tree uniformly at random only $a = \sqrt{d}$ dimensions, and computes the approximate first principal component in the subspace defined by these dimensions.

The gradient descent update for approximate PCA is

$$r_t := r_{t-1} + \gamma \operatorname{Cov}(Z) r_{t-1}, \qquad r_t := r_t / \left\| r_t \right\|_2,$$

where r_t is the projection vector at time t, Z is a matrix containing the data, and γ is the learning rate which we fix as 0.01. We did not observe further tuning of this hyperparameter to be necessary.

Algorithm 7 details a recursive algorithm for growing a sparse approximate PCA tree. Algorithm 8 details the actual approximate PCA algorithm used to find the split direction. On line 7, the matrix Z is formed by taking the vectors of the current points X as columns of a matrix and then slicing only the rows (dimensions) that were randomly selected into the set D on line 6. The sample covariance matrix C is formed from Z on lines 8-9. Lines 10-11 initialize the projection vector from the unit sphere, while lines 12-17 implement the gradient descent update described above. By default, we do $t = 20$ iterations of gradient descent, unless the ℓ_1 norm of the projection vector changes by less than $\epsilon = 0.01$ in a single iteration.

Algorithm 7 Grow a randomized PCA tree (Jääsaari et al., 2019)

1: **Input:** a set of node points X, current depth, maximum depth ℓ, sparsity parameter a, maximum number of iterations t, learning rate γ, threshold parameter ϵ
2: **Output:** a node of a tree
3: **procedure** GROW(X, depth, $\ell, a = \lceil\sqrt{d}\rceil, t = 20, \gamma = 0.01, \epsilon = 0.01$)
4: **if** depth $= \ell$ **then**
5: **return** X as a leaf node
6: direction $\leftarrow$ PCA-GENERATE-SPLIT-DIRECTION($X, a, t, \gamma, \epsilon$)
7: proj $\leftarrow$ PROJECT(X, direction)
8: $\hat{s} \leftarrow$ MEDIAN(proj)
9: $X_{\text{left}} \leftarrow$ points in X for which proj $\leq \hat{s}$
10: $X_{\text{right}} \leftarrow$ points in X for which proj $> \hat{s}$
11: left $\leftarrow$ GROW(X_{left}, depth + 1, $\ell, a, t, \gamma, \epsilon$)
12: right $\leftarrow$ GROW(X_{right}, depth + 1, $\ell, a, t, \gamma, \epsilon$)
13: **return** (left, right, direction, $\hat{s}$) as an inner node

Appendix C. Additional experimental results

C.1 Index construction times

The index construction times of the algorithms at the optimal parameters for the recall levels $R = 0.8, 0.9, 0.95$ are shown in Table 4. The computation time for finding the nearest neighbors (i.e., the labels) of the corpus points is not included in the index construction times, since they are computed only once for each data set; they are found in Table 6 (in the column "exact").

The ensembles of unsupervised trees are relatively fast to build, especially on the high-dimensional data sets: on STL-10, PCA trees have the fastest query times, and are an order of magnitude faster to train compared to the graph and quantization methods.

The multiclass classification trees (RF) are slower to train compared to the unsupervised (PCA, KD, and RP) trees. We expect that their training times could be decreased by standard techniques, such as using weighted quantile sketches (Greenwald and Khanna, 2001; Zhang and Wang, 2007; Chen and Guestrin, 2016) when optimizing the split points.

Algorithm 8 Generate projection vector for a randomized PCA tree (Jääsaari et al., 2019)

Input: a set of node points X, sparsity parameter a, maximum number of iterations t, learning rate γ, threshold parameter ϵ
Output: an approximate first principal component p in a-dimensional subspace; observe that p has only a nonzero components.
procedure PCA-GENERATE-SPLIT-DIRECTION($X, a, t, \gamma, \epsilon$)
    $N \leftarrow |X|$
    initialize d-dimensional vector p with zeros
    $D \leftarrow a$ random unique dimensions from $1, \ldots, d$
    $Z \leftarrow$ points in X with all components but those in D removed
    $M \leftarrow Z(I_N - \frac{1}{N}\mathbf{1}\mathbf{1}^{\mathrm{T}})Z^T$
    $C \leftarrow \frac{1}{N-1}MM^T$
    $r \leftarrow \mathcal{X}_a(\mathbf{0}, \mathbf{I})$
    $r \leftarrow r/\|r\|_2$
    for $i \in \{1, \ldots, t\}$ **do**
        $r' \leftarrow r$
        $r \leftarrow r + \gamma Cr$
        $r \leftarrow r/\|r\|_2$
        if $\|r - r'\|_1 < \epsilon$ **then**
            break
    $j \leftarrow 1$
    for $i \in D$ **do**
        $p[i] \leftarrow r[j]$
        $j \leftarrow j + 1$
    return p

Table 4: Index building time (seconds) at optimal parameters. The fastest time for each recall level is typeset in bold.

data set	R (%)	PCA	KD	RP	RF	ANNOY	HNSW	IVF-PQ
Fashion	80	2.014	**0.929**	1.298	14.422	1.244	1.518	5.867
	90	1.621	1.500	**1.284**	25.591	11.564	1.690	5.867
	95	1.847	2.208	1.925	45.683	11.564	**1.518**	5.867
GIST	80	27.732	27.162	30.520	131.430	16.349	19.114	**13.031**
	90	30.313	36.664	56.624	300.340	62.931	21.560	**13.031**
	95	30.313	49.707	48.056	300.340	**8.319**	26.456	13.031
STL-10	80	**4.497**	25.426	12.204	316.790	32.036	93.393	92.577
	90	**8.891**	30.814	12.145	647.320	489.482	132.500	92.577
	95	**7.918**	28.286	12.145	466.520	489.480	201.070	92.577
Trevi	80	**4.900**	11.158	10.185	420.250	141.794	60.044	43.520
	90	18.937	13.886	**10.169**	420.250	141.790	60.044	43.520
	95	18.937	12.432	**11.261**	420.250	141.790	60.044	43.520

C.2 Comparison to graph and quantization methods

To empirically justify studying partition-based ANN algorithms, we also include in the comparison Hierarchical Navigable Small World (HNSW) (Malkov and Yashunin, 2018) graphs and Inverted File Product Quantization (IVF-PQ) Jegou et al. (2010), that were the fastest graph-based and the fastest quantization-based algorithm, respectively, according to ANN-benchmarks (Aumüller et al., 2019a) project at the time of its publication[21]. For completeness, we also include the commonly-used tree-based method ANNOY[22] in the comparison. See Table 2 for the results. We emphasize that this is not a benchmark article with the goal of proposing a single ANN algorithm and demonstrating its superiority over the competition—rather, we aim to establish a widely applicable theoretical framework for partition-based ANN search.

Table 5: Query times (seconds / 1000 queries) at different recall levels for the different tree types. The fastest method in each case is typeset in boldface.

data set	R (%)	PCA	KD	RP	RF	ANNOY	HNSW	IVF-PQ
Fashion	80	0.075	0.076	0.099	**0.063**	0.193	0.064	0.266
	90	0.111	0.126	0.172	0.095	0.296	**0.089**	0.291
	95	0.163	0.171	0.261	0.146	0.419	**0.097**	0.340
GIST	80	1.330	0.958	1.009	0.705	2.525	**0.524**	0.872
	90	2.942	2.286	2.226	1.530	5.973	**0.819**	2.037
	95	5.641	4.451	4.598	3.253	7.477	**1.212**	2.657
STL-10	80	**0.382**	0.872	1.211	0.756	21.110	1.473	6.140
	90	**0.756**	2.126	3.248	1.774	24.826	2.717	6.860
	95	**1.315**	4.376	7.330	3.654	35.459	3.963	6.860
Trevi	80	**0.330**	0.543	0.591	0.582	5.259	0.705	1.677
	90	**0.684**	1.464	1.468	1.234	9.921	1.202	1.892
	95	**1.212**	3.244	3.289	2.350	17.172	1.896	2.655

C.3 Training classifiers with noisy labels

The disadvantage of the supervised ANN search algorithms compared to the purely unsupervised algorithms is that they require computing the true nearest neighbors $\{y_i\}_{i=1}^n$ of the training set points $\{x_i\}_{i=1}^n$, which is an $\mathcal{O}(nmd)$ operation. This is not a problem for the benchmark data sets used in this article—for the largest data set (STL-10, $n = 98000$, $m = 98000$, $d = 9216$), computing the ground truth took 50 minutes on a single machine—but in the typical applications of ANN search the corpus size may be hundreds of millions or even billions.

21. As of May 2022, the fastest graph-based method is NGTQG (`https://github.com/yahoojapan/NGT/blob/master/bin/ngtqg/README.md`), the fastest quantization-based algorithm is SCANN (Guo et al., 2020) (see `http://ann-benchmarks.com/index.html` for updated results).
22. `https://github.com/spotify/annoy`

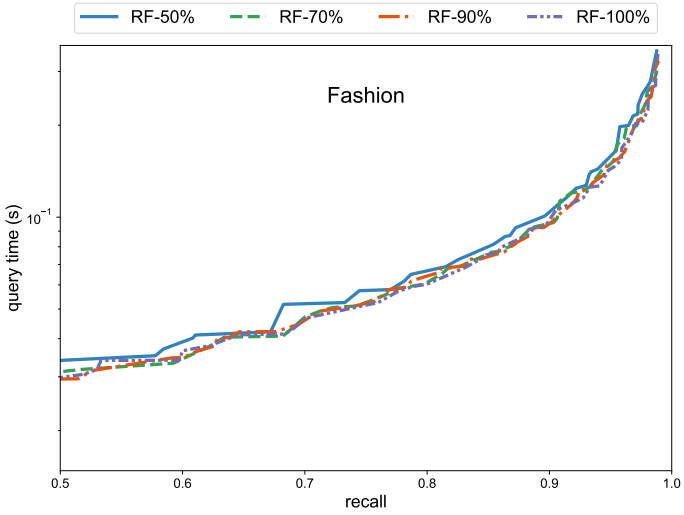

Figure 6: Recall vs. query time (log scale) of a random forest trained with different amounts of label noise on the Fashion data set. RF-100% is the random forest trained using the exact k-nn matrix, RF-90% is the random forest trained using approximate k-nn matrix with that contains on average 90% of the correct neighbors, etc. Allowing 10% noise in labels has no visible effect on performance, and even allowing 50% noise has very little effect.

The labels used to train the classifier do not have to be exact. We can also compute the *approximate* nearest neighbors of the training set $\{x_i\}_{i=1}^m$ and use them as labels $\{y_i\}_{i=1}^m$ to train the classifier. For instance, using approximate nearest neighbors computed at average recall level of 90% amounts to using noisy labels with 10% noise.

To test how the noisy labels affect the performance, we fit the random forest to training sets with 10%, 20%, 30%, 40%, and 50% label noise. The noisy labels are obtained by running the MRPT algorithm (Hyvönen et al., 2016) (i.e., an ensemble of RP trees where the candidate set is selected by voting) in combination with the automatic hyperparameter tuning algorithm (Jääsaari et al., 2019) to find the approximate nearest neighbors of the training set points with recall levels 90%, 80%, 70%, 60%, and 50%, respectively.

The results (c.f. Fig 6) indicate that tree-based classifiers are robust with respect to label noise: training the random forest with 10% label noise has no visible effect on the performance of the algorithm, and even training on labels with 50% noise has very little effect.

Computing times for exact and approximate nearest neighbors for the training set are found on Table 6 for all the four data sets. The results indicate that significant savings in preprocessing time can be obtained by using noisy labels: for instance, on STL-10 computing the exact computation took 50 minutes, whereas the approximate computation took only 1-10 minutes depending on the recall level.

Table 6: Computation times for exact (brute force) and noisy (MRPT algorithm) labels in seconds. The percentage is the average number of correct approximate nearest neighbors.

data set	exact	95%	90%	70%	50%
Fashion	105.8	5.6	3.1	1.8	1.0
GIST	371.7	92.7	61.9	20.2	10.0
Trevi	1450.2	118.6	104.4	31.4	24.1
STL-10	2992.7	596.3	409.0	97.2	66.5

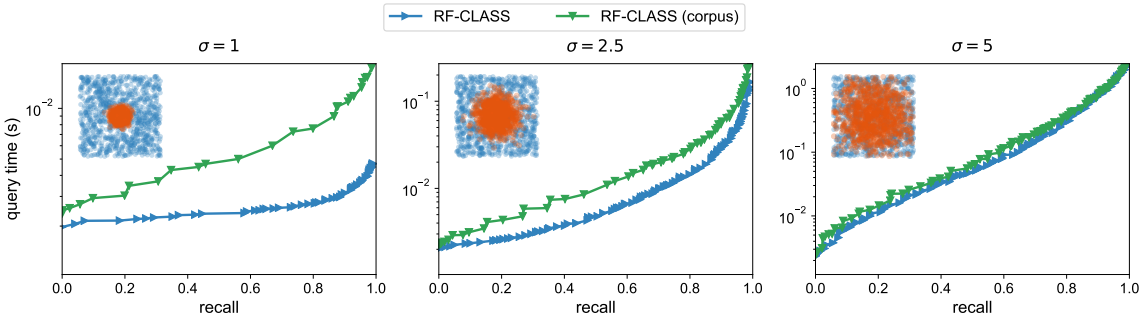

Figure 7: Recall vs. query time (log scale) and two-dimensional projections of the corpus and query points. The blue points are corpus points drawn from $U(-10, 10)$ and the orange points are query points that are drawn from $N(0, \sigma^2 I)$, where $\sigma \in \{1, 2.5, 5\}$. The performance difference between the random forest fit to the sample from the query distribution (RF-CLASS) compared to the random forest fit to the corpus (RF-CLASS corpus) is greater for the more concentrated query distributions.

C.4 Mismatch between the query distribution and the corpus distribution

An additional strength of the proposed framework is that it directly handles a situation where the query distribution differs from the corpus distribution. We explore the effect of the difference between the corpus distribution and the query distribution by generating three synthetic data sets with varying degrees of concentration of the query distribution. The corpus $\{c_j\}_{j=1}^m$ is always a set of 100 000 points is drawn from the 500-dimensional uniform distribution on the interval $(-10, 10)$. The training set of $\{x_i\}_{i=1}^n$ of 100 000 points and a test set of 100 query points are drawn from the 500-dimensional normal distribution $N(0, \sigma^2 I)$, with standard deviations $\sigma = 1, 2.5, 5$, respectively.

In all three cases, we fit a multilabel random forest by both using the corpus $\{c_j\}_{j=1}^m$ as a training set, and by using $\{x_i\}_{i=1}^n$—i.e., a sample from the actual query distribution—as a training set. We use the natural classifier (Algorithm 2) to select the candidate set in

both cases, and test the performance on the test set that is drawn from the actual query distribution.

The results and the two-dimensional projections of samples from the corpus distribution and the query distribution can be found in Figure 7. The more concentrated the query distribution, the greater the performance difference between the model trained using a training set from the query distribution (RF-CLASS), and the model trained using the corpus as a training set (RF-CLASS (corpus)). When the query distribution is close to the corpus distribution ($\sigma = 5$), their performance is almost equal. This demonstrates that the proposed framework enables adapting an index structure to the actual query distribution, assuming that a training set from the query distribution is available.

References

Rahul Agrawal, Archit Gupta, Yashoteja Prabhu, and Manik Varma. Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 13–24. ACM, 2013.

Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87, 2019a.

Martin Aumüller, Tobias Christiani, Rasmus Pagh, and Michael Vesterli. PUFFINN: parameterless and universally fast finding of nearest neighbors. *arXiv preprint arXiv:1906.12211*, 2019b.

Rohit Babbar and Bernhard Schölkopf. Dismec: Distributed sparse machines for extreme multi-label classification. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 721–729, 2017.

Rohit Babbar and Bernhard Schölkopf. Data scarcity, robustness and extreme multi-label classification. *Machine Learning*, 108(8-9):1329–1351, 2019.

Dmitry Baranchuk, Dmitry Persiyanov, Anton Sinitsin, and Artem Babenko. Learning to route in similarity graphs. *arXiv preprint arXiv:1905.10987*, 2019.

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

Lawrence Cayton and Sanjoy Dasgupta. A learning framework for nearest neighbor search. *Advances in Neural Information Processing Systems*, 20:233–240, 2007.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016.

Jingyi Cui, Hanyuan Hang, Yisen Wang, and Zhouchen Lin. GBHT: Gradient boosting histogram transform for density estimation. In *International Conference on Machine Learning*, pages 2233–2243. PMLR, 2021.

Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 537–546, 2008.

Sanjoy Dasgupta and Kaushik Sinha. Randomized partition trees for nearest neighbor search. *Algorithmica*, 72(1):237–263, 2015.

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th Annual Symposium on Computational Geometry*, pages 253–262, 2004.

Krzysztof Dembczynski, Weiwei Cheng, and Eyke Hüllermeier. Bayes optimal multilabel classification via probabilistic classifier chains. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 279–286, 2010.

Luc Devroye, László Györfi, and Gábor Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer Science & Business Media, 1996.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 28(2):337–407, 2000.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic time. *ACM Trans. Math. Software*, 3(SLAC-PUB-1549-REV. 2):209–226, 1976.

Long Gong, Huayi Wang, Mitsunori Ogihara, and Jun Xu. iDEC: indexable distance estimating codes for approximate nearest neighbor search. *Proceedings of the VLDB Endowment*, 13(9), 2020.

Yunchao Gong, Sanjiv Kumar, Vishal Verma, and Svetlana Lazebnik. Angular quantization-based binary codes for fast similarity search. *Advances in Neural Information Processing Systems*, 25:1196–1204, 2012.

Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. *ACM SIGMOD Record*, 30(2):58–66, 2001.

Chuan Guo, Ali Mousavi, Xiang Wu, Daniel N Holtmann-Rice, Satyen Kale, Sashank Reddi, and Sanjiv Kumar. Breaking the glass ceiling for embedding-based classifiers for large output spaces. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4944–4954, 2019.

Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, pages 3887–3896. PMLR, 2020.

John A Hartigan. *Clustering algorithms*. John Wiley & Sons, Inc., 1975.

Ville Hyvönen, Teemu Pitkänen, Sotiris Tasoulis, Elias Jääsaari, Risto Tuomainen, Liang Wang, Jukka Corander, and Teemu Roos. Fast nearest neighbor search through sparse random projections and voting. In *Proceedings of the 4th IEEE International Conference on Big Data*, pages 881–888. IEEE, 2016.

Ville Hyvönen, Elias Jääsaari, and Teemu Roos. A multilabel classification framework for approximate nearest neighbor search. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 604–613, 1998.

Masajiro Iwasaki and Daisuke Miyazaki. Optimization of indexing based on k-nearest neighbor graph for proximity search in high-dimensional data. *arXiv preprint arXiv:1810.07355*, 2018.

Elias Jääsaari, Ville Hyvönen, and Teemu Roos. Efficient autotuning of hyperparameters in approximate nearest neighbor search. In *Proceedings of the 23rd Pacific-Asia Conference on Knowledge Discovery and Data Mining*, volume 2, pages 590–602. Springer, 2019.

Himanshu Jain, Yashoteja Prabhu, and Manik Varma. Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 935–944, 2016.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33 (1):117–128, 2010.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

R. Kaas and J.M. Buhrman. Mean, median and mode in binomial distributions. *Statistica Neerlandica*, 34(1):13–18, 1980.

Weihao Kong and Wu-Jun Li. Isotropic hashing. *Advances in Neural Information Processing Systems*, 25:1646–1654, 2012.

Petri Kontkanen and Petri Myllymäki. MDL histogram density estimation. In *Artificial Intelligence and Statistics*, pages 219–226. PMLR, 2007.

Oluwasanmi O Koyejo, Nagarajan Natarajan, Pradeep K Ravikumar, and Inderjit S Dhillon. Consistent multilabel classification. *Advances in Neural Information Processing Systems (NeurIPS)*, 28:3321–3329, 2015.

Der-Tsai Lee and Chak-Kuen Wong. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*, 9(1):23–29, 1977.

Shengqiao Li. Concise formulas for the area and volume of a hyperspherical cap. *Asian Journal of Mathematics and Statistics*, 4(1):66–70, 2011.

Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. Approximate nearest neighbor search on high dimensional data-experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, pages 1475–1488, 2019.

Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *Proceedings of the 28th International Conference on Machine Learning*, pages 1–8, 2011.

Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2074–2081. IEEE, 2012.

Ezequiel López-Rubio. A histogram transform for probability density function estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(4):644–656, 2013.

Gábor Lugosi and Andrew Nobel. Consistency of data-driven histogram methods for density estimation and classification. *The Annals of Statistics*, 24(2):687–706, 1996.

Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 2014.

Songrit Maneewongvatana and David M Mount. The analysis of a probabilistic approach to nearest neighbor searching. In *Workshop on Algorithms and Data Structures*, pages 276–286. Springer, 2001.

David McAllester and Luis Ortiz. Concentration inequalities for the missing mass and for histogram rule error. *Journal of Machine Learning Research*, 4(Oct):895–911, 2003.

Mark McCartin-Lim, Andrew McGregor, and Rui Wang. Approximate principal direction trees. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pages 1611–1618, 2012.

Aditya K Menon, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. Multilabel reductions: what is my loss optimising? *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

Marius Muja and David G Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

Mohammad Norouzi and David J Fleet. Minimal loss hashing for compact binary codes. In *28th International Conference on on Machine Learning*, pages 353–360, 2011.

Yashoteja Prabhu and Manik Varma. FastXML: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 263–272, 2014.

Sashank J Reddi, Satyen Kale, Felix Yu, Daniel Holtmann-Rice, Jiecao Chen, and Sanjiv Kumar. Stochastic negative mining for learning with large output spaces. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1940–1949. PMLR, 2019.

Chanop Silpa-Anan and Richard Hartley. Optimised kd-trees for fast image descriptor matching. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.

Robert F Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6(1-6):579–589, 1991.

Christoph Strecha, Alex Bronstein, Michael Bronstein, and Pascal Fua. LDAHash: Improved matching with smaller descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1):66–78, 2011.

Vladimir Vapnik and Alexey Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2): 264–280, 1971.

Jingdong Wang, Ting Zhang, Nicu Sebe, and Heng Tao Shen. A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):769–790, 2017.

Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data – a survey. *Proceedings of the IEEE*, 104(1):34–57, 2015.

Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1753–1760, 2009.

Ian EH Yen, Xiangru Huang, Wei Dai, Pradeep Ravikumar, Inderjit Dhillon, and Eric Xing. PPDSparse: A parallel primal-dual sparse method for extreme classification. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 545–553, 2017.

Li Yuan, Tao Wang, Xiaopeng Zhang, Francis EH Tay, Zequn Jie, Wei Liu, and Jiashi Feng. Central similarity quantization for efficient image and video retrieval. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3083–3092, 2020.

Qi Zhang and Wei Wang. A fast algorithm for approximate quantiles in high speed data streams. In *19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)*, pages 29–29. IEEE, 2007.