Improved Search of Relevant Points for Nearest-Neighbor Classification

Alejandro Flores-Velazco
Department of Computer Science, University of Maryland, College Park, MD, USA

Abstract

Given a training set $P \subset \mathbb{R}^d$, the *nearest-neighbor classifier* assigns any query point $q \in \mathbb{R}^d$ to the class of its closest point in P. To answer these classification queries, some training points are more relevant than others. We say a training point is *relevant* if its omission from the training set could induce the misclassification of some query point in $\mathbb{R}^d$. These relevant points are commonly known as *border points*, as they define the boundaries of the Voronoi diagram of P that separate points of different classes. Being able to compute this set of points efficiently is crucial to reduce the size of the training set without affecting the accuracy of the nearest-neighbor classifier.

Improving over a decades-long result by Clarkson (FOCS'94), Eppstein (SOSA'22) recently proposed an *output-sensitive* algorithm to find the set of border points of P in $\mathcal{O}(n^2 + nk^2)$ time, where k is the size of such set. In this paper, we improve this algorithm to have time complexity equal to $\mathcal{O}(nk^2)$ by proving that the first phase of their algorithm, which requires $\mathcal{O}(n^2)$ time, are unnecessary.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases nearest-neighbor classification, nearest-neighbor rule, decision boundaries, border points, relevant points

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.54

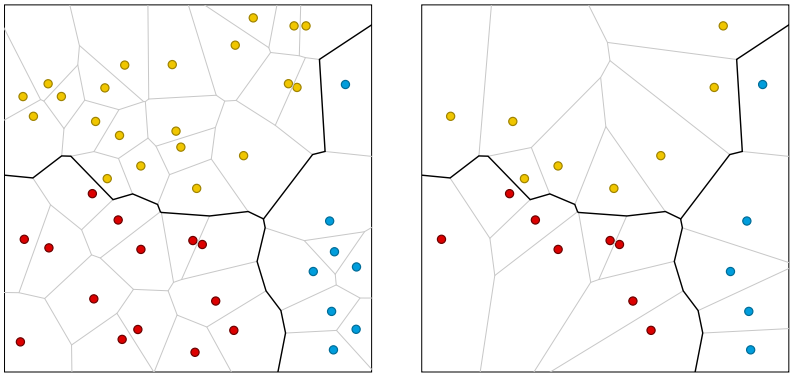
Acknowledgements Thanks to Prof. David Mount for pointing out Eppstein's paper [12] and for the valuable discussions on the results presented in this paper.

1 Introduction

In the context of non-parametric classification, we are given a training set $P \subset \mathbb{R}^d$ consisting of n *labeled* points in d-dimensional Euclidean space, where the label of every point in P indicates the *class* (or *color*) that the point belongs to. The goal of a classifier is to use the training set P to *predict* the class for any *unlabeled* query point $q \in \mathbb{R}^d$, that is, to *classify* q.

The *nearest-neighbor classifier* (also known as *nearest-neighbor rule*) [13] stands out as a simple yet powerful method, that works by assigning any query point q to the class of its closest point in P. Despite its simplicity, the nearest-neighbor classifier is well-known to exhibit good classification accuracy both experimentally and theoretically [10, 11, 30]. In fact, it is still frequently used in many applications [5, 18, 22, 25–27, 29] over more recent and sophisticated techniques like support-vector machines [9] and deep neural networks [28].

One of the principal disadvantages of this technique is its high dependency on the size and dimensionality of the data, especially in light of *big data* applications. With training sets with billions of points becoming increasingly common, reducing the nearest-neighbor classifier's dependency on n and d is one approach to enhance its efficiency. There has been significant progress towards this goal, mainly focusing on two directions. The first involves the design of efficient data structures to answer approximate nearest-neighbor queries [2–4, 17, 19, 20, 23]. The second direction focuses on reducing the size of the training set used by the nearest-neighbor classifier, thus effectively reducing n. However, most practical techniques for training set reduction provide limited guarantees on the effect of this reduction to the accuracy of the nearest-neighbor classifier [1, 14–16].



(a) Original training set P. **(b)** Border/Relevant points of P.

Figure 1 On the left, a training set P with points of three classes: *red*, *blue* and *yellow*. There the black lines highlight the *boundaries* of P between points of different classes. On the right, a subset of these points corresponding to the set of border points of P. Note that by definition, the boundaries between points of different classes remain the same for P and for its set of border points.

Only a handful of works [6, 8, 12] have proposed training set reduction algorithms that guarantee the same classification of every query point, before and after the reduction took place. These are called *boundary preserving* algorithms, and it is the focus of this paper.

The set of *border points* (or *relevant points*[1]) of the training set P are those that define the boundaries between points of different classes, and whose omission from the training set would imply the misclassification of some query points in $\mathbb{R}^d$. Formally, two points $p, \hat{p} \in P$ are border points of P if they belong to different classes, and there exist some point $q \in \mathbb{R}^d$ such that q is equidistant to both p and $\hat{p}$, and no other point of P is closer to q than these two points (*i.e.*, the empty ball property of Voronoi Diagrams). See Figure 1 for an example of a training set P in $\mathbb{R}^2$ and its set of border points. Throughout, we let k denote the total number of border points in the training set. By definition, if instead of building the nearest-neighbor classifier with the entire training set P we use the set of border points of P, its dependency is reduced from n to k, while still obtaining the same classification for any query point in $\mathbb{R}^d$. This becomes particularly relevant for applications where $k \ll n$.

In this paper, we improve a recently proposed algorithm by [12] that computes the set of border points of any training set $P \subset \mathbb{R}^d$, where dimension d is assumed to be constant. While the original algorithm computes such set in $\mathcal{O}(n^2 + nk^2)$ time, where k is the number of border points of P, our new algorithm computes the same set in $\mathcal{O}(nk^2)$ time.

1.1 Previous Work

Other related problems in the realm of training set reduction are NP-hard [24, 31, 33] to solve exactly (*e.g.*, those of finding minimum cardinality *consistent* subsets and *selective* subsets). However, the problem of preserving the class boundaries of the nearest-neighbor classifier, or simply, finding the set of border points of P, is tractable.

For training sets $P \subset \mathbb{R}^2$ in 2-dimensional Euclidean space, Bremner *et al.* [6] proposed an output-sensitive algorithm for finding the set of border points of P in $\mathcal{O}(n \log k)$ worst-case time. However, how to generalize this algorithm for higher dimensions remained unclear.

[1] While [12] uses the term *relevant points*, the term *border points* has been the standard in the literature of this and other related problems [14, 15, 21, 32]. For this reason, we stick to the term *border points*.

Until very recently, the best result for the higher dimensional case was that of Clarkson [8]. He proposed an algorithm to find the set of border points of $P \subset \mathbb{R}^d$, with bounded d, that runs in $\mathcal{O}(\min(n^3, kn^2 \log n))$ worst-case time. For almost three decades, this remained the best result for training sets in $\mathbb{R}^d$. Recently, Eppstein [12] proposed a significantly faster algorithm for the d-dimensional Euclidean case, which runs in $\mathcal{O}(n^2 + nk^2)$ worst-case time.

Eppstein's algorithm is strikingly simple, yet full of interesting ideas (see Algorithm 1). The algorithm works as follows: it begins by selecting an initial set of border points of P, one point from every class region. From here, the algorithm uses a series of subroutines which we will group together and denote as the "*inversion method*", to find the remaining border points of P. Thus, the algorithm can be naturally split into two phases: the initialization of R with some border points, and the search process for the remaining border points of P.

Algorithm 1 Recent Eppstein's algorithm [12] to find the set of border points of P.

Input: Initial training set P
Output: The set of border points of P
1 Let M be the MST of P
2 Initialize R with the end points of every bichromatic edge of M
3 **foreach** $p \in R$ **do**
4 Let c be p's class and P_c be the points of P that belong to class c
5 Let S_p be the inverted points of $P \setminus P_c$ around p
6 Find all extreme points of S_p and their corresponding original points E_p
7 $R \leftarrow R \cup E_p$
8 **return** R

The initialization phase (lines 1–2 of Algorithm 1) involves finding a subset of border points such that at least one point for every class region is selected. Eppstein observes that this can be achieved by computing the Minimum Spanning Tree (MST) of P, identifying the edges of the MST that connect points of different classes (denoted as *bichromatic* edges), and selecting the endpoints of all such edges. This phase takes $\mathcal{O}(n^2)$ time, but we will prove that it is not necessary.

The search phase (lines 3–6 of Algorithm 1) is in charge of finding every remaining border point of P. This phase iterates over all selected points, and for each such point p, it performs what we call the inversion method. This method identifies a subset of border points of P, which are added to R. Once the algorithm has done the inversion method on every point of R, it terminates with the guarantee of having selected every border point of P.

Given any point $p \in P$, the inversion method on p is described in lines 4–6 of Algorithm 1. Let c be p's class, and P_c be the points of P that belong to class c, the inversion method on p consists of: (i) inverting all points of $P \setminus P_c$ around a ball centered at p (call the set of these inverted points as S_p and include p itself in the set), (ii) computing the set of extreme points of S_p, and finally (iii) returning the set E_p of those points of P that correspond to the extreme points of S_p before inversion. For a detailed description and proof of correctness of this method, we refer the reader to Eppstein's paper [12]. However, for the purposes of this paper we only need a property presented in Lemma 3 of [12]: the points in E_p reported by the inversion method are the Delaunay neighbors of p with respect to the set $(P \setminus P_c) \cup \{p\}$.

Every call of the inversion method takes $\mathcal{O}(nk)$ time by leveraging well-known output-sensitive algorithms for computing extreme points. Given that this method is called exclusively on every border point of the training set, this yields a total of $\mathcal{O}(nk^2)$ time to complete the search phase of the algorithm. Overall, this implies that Eppstein's algorithm computes the entire set of border points of P in $\mathcal{O}(n^2 + nk^2)$ worst-case time.

2 Our Approach

We propose a simple modification to Eppstein's algorithm, which avoids the step of computing the MST of the training set P, along with the subsequent selection of bichromatic edges to produce the initial subset of border points.

Instead, we simply start the search process with any arbitrary point of P. The rest of the algorithm remains virtually unchanged (see Algorithm 2 for a formal description). We show that this new approach is not only correct, meaning that it only finds border points of P, but also complete, as all border points of P are eventually found by our algorithm. Additionally, by avoiding the main bottleneck of the original algorithm, our new algorithm computes the same result in $\mathcal{O}(nk^2)$ time, eliminating the $\mathcal{O}(n^2)$ term.

Algorithm 2 New algorithm to find the set of border points of P.

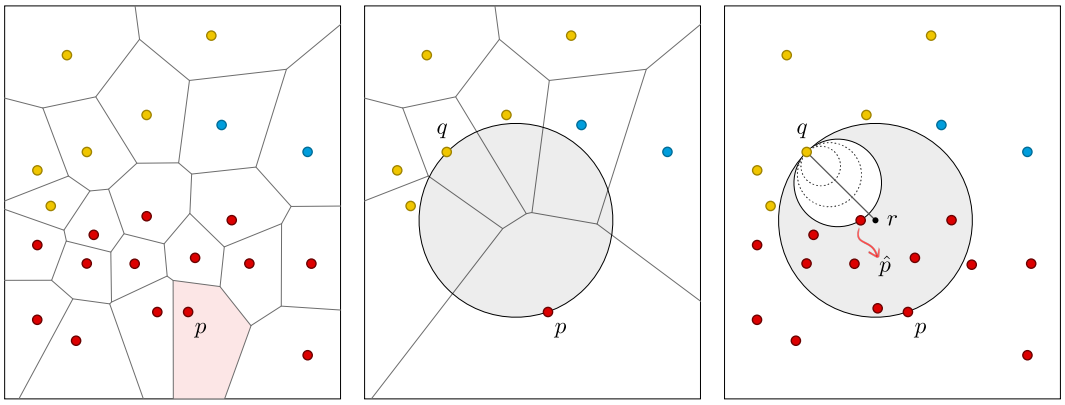
Input: Initial training set P
Output: The set of border points of P
Let s be any "*seed*" point from P
$R \leftarrow \phi$
foreach $p \in R \cup \{s\}$ **do**
    Let c be p's class and P_c be the points of P that belong to class c
    Let S_p be the inverted points of $P \setminus P_c$ around p
    Find all extreme points of S_p and their corresponding original points E_p
    $R \leftarrow R \cup E_p$
return R

Before proceeding, it is useful to explore why Eppstein's algorithm computes the MST of the training set P. First, note that the original algorithm only applies the inversion method on border points of P. In fact, Eppstein's correctness proof relies on it: Lemma 6 in [12] proves that all points in E_p are border points by assuming that point p is also a border point. From the description of our algorithm, note that we initially apply the inversion method on a "seed" point s, which might not be a border point. Therefore, we need to generalize Lemma 6 in [12] for the case where p is not a border point of P. Additionally, using the points from all bichromatic pairs of the MST of P guarantees that Eppstein's algorithm starts the search phase with at least one point from every boundary of P. Eppstein's completeness proof shows that this search can then "move along" any given boundary and eventually select all its defining points. We show that the search process is far more powerful, and can even "jump" between nearby boundaries, thus rendering the MST computation unnecessary.

The following description outlines the necessary steps to prove both the correctness and completeness of our new algorithm, which are unfolded in the rest of this section.

- By applying the inversion method to any point of P, not necessarily a border point, all reported points are border points of P. This is established in Lemma 1, generalizing the statement of Lemma 6 of [12] for non-border points.
- For any class boundary of P, once the algorithm selects a point from this boundary, it will eventually select every other point defining the same boundary. This is originally proved in Lemma 10 [12], however, we provide simpler proofs in Lemmas 2 and 3.
- Given two disconnected boundaries separated by a class region, we prove that if our algorithm selects a defining point from one of the boundaries, it will eventually select all defining points from both boundaries. This is proved in Lemma 4.

All together, these lemmas are used to prove the main result: the correctness, completeness, and worst-case time complexity of Algorithm 2, as stated in Theorems 5 and 6.



(a) Training set P. **(b)** Points from $(P \setminus P_c) \cap \{p\}$. **(c)** q and $\hat{p}$ are border points.

Figure 2 Example showing the inversion method from any point $p \in P$. On the left, training set P. The middle figure shows every non red point of P, except for p itself, along with a point q selected from the inversion method on p. On the right, we see evidence that q is a border point of P.

▶ **Lemma 1.** *Let $p \in P$ be any point of the training set. Then every point selected using the inversion method on p must be a border point of P.*

Proof. Let E_p be the points of P corresponding (before inversion) to the extreme points of S_p. According to Lemma 3 [12], every point in E_p is a neighbor of point p with respect to the Voronoi Diagram of set $(P \setminus P_c) \cup \{p\}$. This implies that for every point $q \in E_p$ other than p, there exists a ball such that both p and q are on its surface and no points of $P \setminus P_c$ lie inside (see Figures 2a and 2b). We can now leverage similar techniques to the ones described in [6], to find a "witness" point to the hypothesis that q must be a border point of P.

Recall that the empty ball we just described, as illustrated in Figure 2b, is empty from points of $P \setminus P_c$. However, there might be points of P_c inside. And moreover, we know that at least one point of P_c, point p, lies on its surface. Now, let r be the center of this ball, we grow an empty ball, this time with respect to the entire training set P, such that its center lies on the line $\overline{qr}$ and point q is on its surface (see Figure 2c). This ball will grow until it hits another point $\hat{p}$ of P, which we are guaranteed it will be of the same class as point p, and thus, of different class as point q. Finally, we have just found an empty ball with respect to P, which has points q and $\hat{p}$ on its surface, and were the class of both points differ. Therefore, this implies that q is a border point of P. ◀

Before continuing, we need to formally define a few concepts. First, we define a *wall* of P as any $(d-1)$-dimensional face of the Voronoi Diagram of P. By known properties of these structures, every wall w is *defined* by two distinct points $p, q \in P$ such that any point on w has p and q as its two equidistant nearest-neighbors in the training set. We say two walls are *adjacent* if their intersection is not empty. That is, if there exists a point in $\mathbb{R}^d$ with all the defining points of these two walls as its equidistant nearest-neighbors in P.

Additionally, we define a *class boundary* (or just *boundary*) of P as the union of adjacent walls, where each of these walls is defined by two points of different classes. Similarly, we define a *class region* of P as the union of adjacent Voronoi cells whose defining points belong to the same class. Based on these definitions, note that class boundaries are the ones that separate different class regions of P. Figure 4 illustrates a training set in $\mathbb{R}^2$ with points of three classes, whose Voronoi Diagram describes five class regions and two class boundaries.

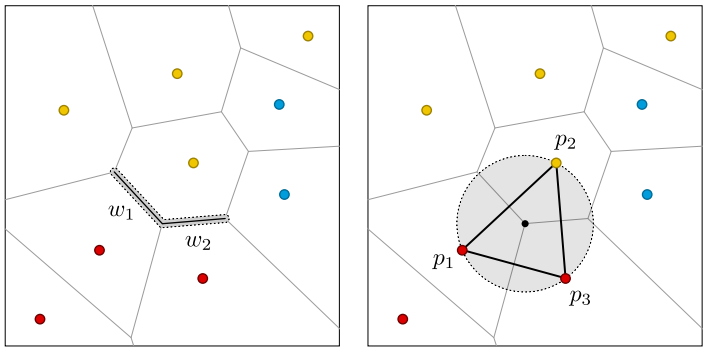


Figure 3 By definition, any two adjacent walls w_1 and w_2 of the Voronoi Diagram of P hold the empty ball property with the points that define them. When these walls are part of the class boundaries of P, the points that define them belong to at least two classes.

▶ **Lemma 2.** *Let w_1 and w_2 be two adjacent walls in a class boundary of P. If the algorithm selects one of the points defining one of these walls, it eventually selects the remaining points defining both walls.*

Proof. Let $\mathcal{W}$ be the set of points defining both walls w_1 and w_2 (see Figure 3). By definition, these two walls of the Voronoi Diagram of P are adjacent if there exists an empty ball with all the points of $\mathcal{W}$ on its surface. Knowing these two walls are part of the class boundaries of P, the set $\mathcal{W}$ must contain at least three points, and at least two classes.

Let p_1 be the first point of $\mathcal{W}$ to be selected by the algorithm. When doing the inversion method on point p_1, the algorithm will select all points of $\mathcal{W}$ of different class than p_1, of which we know there is at least one. Let p_2 be one such point. Finally, when doing the inversion method on point p_2, the algorithm will select the remaining points of $\mathcal{W}$ of the same class as p_1. Therefore, all points of $\mathcal{W}$ will eventually be selected by the algorithm. ◀

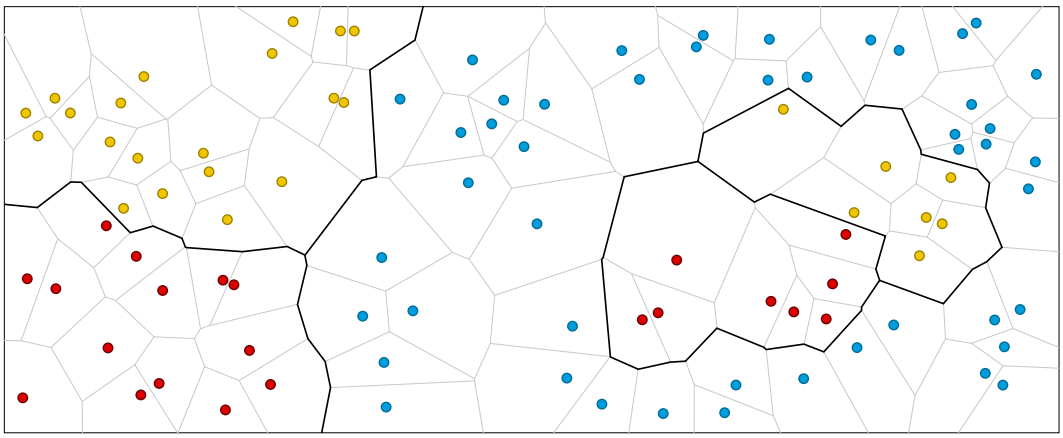


Figure 4 A training set with five class regions (one *blue*, two *red*, and two *yellow* regions), along with two disconnected class boundaries that separate all these regions. On the left, a boundary separating the blue region and the leftmost yellow and red regions. On the right, a boundary that separates the same blue region and the remaining red and yellow regions.

▶ **Lemma 3.** *Let $\mathcal{A}$ be a class boundary of P, and assume that the algorithm selects one of the defining points of $\mathcal{A}$. Then, the algorithm will eventually select all defining points of $\mathcal{A}$.*

This comes as a direct consequence of Lemma 2 and the definition of a class boundary of the training set P. It remains to show what happens with boundaries that are disconnected.

▶ **Lemma 4.** *Let $\mathcal{A}$ and $\mathcal{B}$ be two disconnected boundaries of P, such that there exists a path in space from $\mathcal{A}$ to $\mathcal{B}$ that is completely contained within one color region. Without loss of generality, say that every point that defines $\mathcal{A}$ has been selected by the algorithm. Then, every point that defines $\mathcal{B}$ must also be selected by the algorithm.*

Proof. Given these two disconnected boundaries $\mathcal{A}$ and $\mathcal{B}$, we assume there exists some path $\mathcal{P}$ in $\mathbb{R}^d$ going from a wall of $\mathcal{A}$ to a wall of $\mathcal{B}$, such that this path passes exclusively through a single class region (see Figure 5a). Without loss of generality, say this is a *red* class region. Formally, for every point r along $\mathcal{P}$ we know r's nearest-neighbor in P is red. Additionally, we assume that every border point defining $\mathcal{A}$ is selected by the algorithm. Hence, the proof consists of showing that there exists a sequence of border points $\langle p_1, \hat{p}_1, p_2, \hat{p}_2, \ldots, p_m, \hat{p}_m \rangle$ such that (i) p_1 and $\hat{p}_m$ are defining points of $\mathcal{A}$ and $\mathcal{B}$, respectively, (ii) $\hat{p}_i$ is retrieved by the inversion method on p_i, for every $i \in [1, m]$, and finally (iii) points p_i and $\hat{p}_{i-1}$ are both defining the same boundary, for every $i \in [2, m]$. See Figure 5 for a visual description.

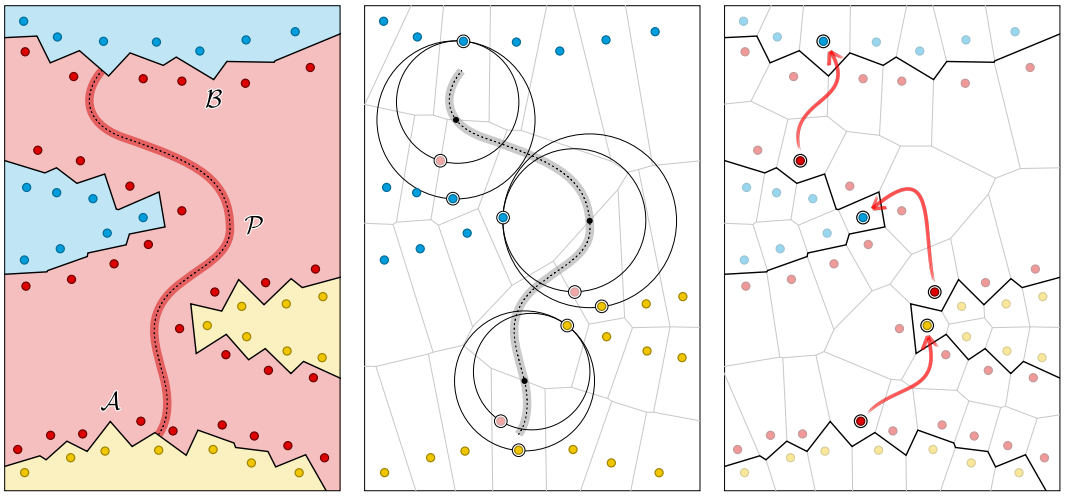


Figure 5 On the right, two disconnected boundaries $\mathcal{A}$ and $\mathcal{B}$ enclosing a red class region. Thus, there is a path $\mathcal{P}$ completely contained inside such region and connecting both boundaries. Other boundaries can also be enclosing the same region and be near path $\mathcal{P}$. On the left, we proof that there exists a sequence of points that can be retrieved by calls to the inversion method, such that if points of $\mathcal{A}$ are selected by the algorithm, eventually points of $\mathcal{B}$ will also be selected.

By definition, for every point r along path $\mathcal{P}$ we know r's nearest-neighbor is a red point. Now, let's delete every red point from consideration, including the ones defining boundaries $\mathcal{A}$ and $\mathcal{B}$ (see Figure 5b). This immediately implies that r's nearest-neighbor just became a non-red border point of P. The fact that r's new nearest-neighbor is a border point is easy to proof, using similar arguments as the ones laid down in Lemma 1. Additionally, these border points could be defining other boundaries apart from $\mathcal{A}$ and $\mathcal{B}$, as seen in Figure 5b.

Let's start moving along the path $\mathcal{P}$, starting from the end-point of the path that lies on a wall of boundary $\mathcal{A}$. Then, find all r_i points along the path, where each r_i has two equidistant nearest-neighbors among the remaining non-red points, and both points define

two distinct boundaries of P. We say there are m of these points along the path, and denote r_i's two equidistant nearest-neighbors as $q_{i,1}$ and $q_{i,2}$ for $i \in [1, m]$. Clearly, $q_{i,1}$ and $q_{i-1,2}$ are border points defining the same boundary, for all $i \in [2, m]$. See Figure 5b, where the three black points along the path are the r_i points, and the *yellow* and *blue* points on the surface of the balls centered at each r_i are the corresponding $q_{i,1}$ and $q_{i,2}$ points.

For now, let's fix the analysis on one such r_i point, and consider the ball centered at r_i with both $q_{i,1}$ and $q_{i,2}$ on its surface. There must exist some other point $q_{i,3}$ lying inside of r_i's ball, such that $q_{i,3}$ is one of the deleted red points defining the same boundary as $q_{i,1}$. It is now easy to see that there exist an empty ball, with respect to the set $P \setminus P_{red} \cup \{q_{i,3}\}$, with both $q_{i,3}$ and $q_{i,2}$ on its boundary. This implies that $q_{i,2}$ is retrieved by the inversion method on $q_{i,3}$. Therefore, let's add $p_i \leftarrow q_{i,3}$ and $\hat{p}_i \leftarrow q_{i,2}$ to the sequence of points that we are looking for. Repeat this for every r_i with $i \in [1, m]$ to identify all points in the sequence.

Finally, we have the sequence of border points $\langle p_1, \hat{p}_1, p_2, \hat{p}_2, \ldots, p_m, \hat{p}_m \rangle$ such that for any $i \in [1, m]$ assuming that the algorithm selects the points defining the same boundary as p_i, it will also select $\hat{p}_i$, and leveraging Lemma 3 it will eventually select all other points defining the same boundary as $\hat{p}_i$. Given that p_1 and $\hat{p}_m$ are defining border points of boundaries $\mathcal{A}$ and $\mathcal{B}$, respectively, and by the assumption that all points defining $\mathcal{A}$ are selected by the algorithm, we know that eventually, all points defining $\mathcal{B}$ will be selected too. ◀

▶ **Theorem 5.** *The algorithm selects every border point of P in $\mathcal{O}(nk^2)$ time.*

Proof. Proving the worst-case time complexity of our algorithm follows directly from the time complexity of the search phase of Eppstein's algorithm [12]. However, the correctness and completeness of our algorithm follows from Lemmas 1 to 4.

First, we know by Lemmas 1-3 that Algorithm 2 will select the defining border points of at least one class boundary of P. Denote this boundary as $\mathcal{A}$ and consider any other boundary $\mathcal{B}$ of P. Evidently, we can draw a path $\mathcal{P}$ from $\mathcal{A}$ to $\mathcal{B}$, which would generally pass through several class regions. Then, let's split $\mathcal{P}$ into several subpaths $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m$ such that each subpath is completely contained within a single class region. From this, we can directly apply Lemma 4 on each of the intermediate boundaries that "cut" $\mathcal{P}$ into these subpaths. Finally, this implies that our algorithm will eventually select every defining point of boundary $\mathcal{B}$, and similarly, it will do the same with all other boundaries of P. ◀

▶ **Theorem 6.** *Leveraging Chan's algorithm [7] for finding extreme points, the algorithm selects every border point of P in randomized expected time $\mathcal{O}(nk \log k)$ for $d = 3$, and in*

$$\mathcal{O}\left(nk(\log k)^{\mathcal{O}(1)} + k(nk)^{1-\frac{1}{\lfloor d/2 \rfloor + 1}}(\log n)^{\mathcal{O}(1)}\right)$$

time for all constant dimensions $d > 3$.

Just as with Eppstein's original algorithm, we can use Chan's randomized algorithm [7] for finding extreme points of point sets in $\mathbb{R}^d$, in order to reduce the expected time complexity of our improved algorithm. The remaining of the proof is the same as for Theorem 5.

References

1 Fabrizio Angiulli. Fast nearest neighbor condensation for large data sets classification. *IEEE Transactions on Knowledge and Data Engineering*, 19(11):1450–1464, 2007.

2 Sunil Arya, Guilherme D. da Fonseca, and David M. Mount. Optimal approximate polytope membership. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 270–288. SIAM, 2017.

3 Sunil Arya, Guilherme D. Da Fonseca, and David M. Mount. Approximate polytope membership queries. *SIAM Journal on Computing*, 47(1):1–51, 2018.

4 Sunil Arya, Theocharis Malamatos, and David M. Mount. Space-time tradeoffs for approximate nearest neighbor searching. *Journal of the ACM (JACM)*, 57(1):1, 2009.

5 Oren Boiman, Eli Shechtman, and Michal Irani. In defense of nearest-neighbor based image classification. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.

6 David Bremner, Erik Demaine, Jeff Erickson, John Iacono, Stefan Langerman, Pat Morin, and Godfried Toussaint. Output-sensitive algorithms for computing nearest-neighbour decision boundaries. In Frank Dehne, Jörg-Rüdiger Sack, and Michiel Smid, editors, *Algorithms and Data Structures: 8th International Workshop, WADS 2003, Ottawa, Ontario, Canada, July 30 - August 1, 2003. Proceedings*, pages 451–461, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. doi:10.1007/978-3-540-45078-8_39.

7 Timothy M Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete & Computational Geometry*, 16(4):369–387, 1996.

8 Kenneth L Clarkson. More output-sensitive geometric algorithms. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 695–702. IEEE, 1994.

9 Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine Learning*, pages 273–297, 1995.

10 T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, January 1967. doi:10.1109/TIT.1967.1053964.

11 Luc Devroye. On the inequality of cover and hart in nearest neighbor discrimination. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, pages 75–78, 1981.

12 David Eppstein. Finding relevant points for nearest-neighbor classification. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 68–78. SIAM, 2022.

13 E. Fix and J. L. Hodges. Discriminatory analysis, nonparametric discrimination: Consistency properties. *US Air Force School of Aviation Medicine*, Technical Report 4(3):477+, January 1951.

14 Alejandro Flores-Velazco. Social distancing is good for points too! In *Proceedings of the 32st Canadian Conference on Computational Geometry, CCCG 2020, August 5-7, 2020, University of Saskatchewan, Saskatoon, Saskatchewan, Canada*, 2020.

15 Alejandro Flores-Velazco and David M. Mount. Guarantees on nearest-neighbor condensation heuristics. In *Proceedings of the 31st Canadian Conference on Computational Geometry, CCCG 2019, August 8-10, 2019, University of Alberta, Edmonton, Alberta, Canada*, 2019.

16 Alejandro Flores-Velazco and David M. Mount. Coresets for the Nearest-Neighbor Rule. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 47:1–47:19, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2020.47.

17 Alejandro Flores-Velazco and David M. Mount. Boundary-sensitive approach for approximate nearest-neighbor classification. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 44:1–44:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.44.

18 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*, 2020. arXiv:1908.10396.

19 Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization, 2020. arXiv:1908.10396.

20 Sariel Har-Peled. A replacement for Voronoi diagrams of near linear size. In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 94–103, 2001.

21 Norbert Jankowski and Marek Grochowski. Comparison of instances selection algorithms I. Algorithms survey. In *Artificial Intelligence and Soft Computing-ICAISC 2004*, pages 598–603. Springer, 2004.

22 Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *arXiv preprint*, 2017. arXiv:1702.08734.

23 Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus, 2017. arXiv:1702.08734.

24 Kamyar Khodamoradi, Ramesh Krishnamurti, and Bodhayan Roy. Consistent subset problem with two labels. In *Conference on Algorithms and Discrete Applied Mathematics*, pages 131–142. Springer, 2018.

25 Marc Khoury and Dylan Hadfield-Menell. Adversarial training with Voronoi constraints. *CoRR*, abs/1905.01019, 2019. arXiv:1905.01019.

26 Nicolas Papernot and Patrick McDaniel. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv preprint*, 2018. arXiv:1803.04765.

27 Neehar Peri, Neal Gupta, W. Ronny Huang, Liam Fowl, Chen Zhu, Soheil Feizi, Tom Goldstein, and John P. Dickerson. Deep k-nn defense against clean-label data poisoning attacks. In *European Conference on Computer Vision*, pages 55–70. Springer, 2020.

28 Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014. arXiv:1404.7828.

29 Chawin Sitawarin and David Wagner. On the robustness of deep k-nearest neighbors. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 1–7. IEEE, 2019.

30 Charles J. Stone. Consistent nonparametric regression. *The annals of statistics*, pages 595–620, 1977.

31 Gordon Wilfong. Nearest neighbor problems. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, SCG '91, pages 224–233, New York, NY, USA, 1991. ACM. doi:10.1145/109648.109673.

32 D. Randall Wilson and Tony R. Martinez. Instance pruning techniques. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 403–411, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. URL: http://dl.acm.org/citation.cfm?id=645526.657143.

33 A. V. Zukhba. NP-completeness of the problem of prototype selection in the nearest neighbor method. *Pattern Recog. Image Anal.*, 20(4):484–494, December 2010. doi:10.1134/S1054661810040097.