# ABSTRACT

Title of dissertation: ALGORITHMS AND DATA STRUCTURES FOR FASTER NEAREST-NEIGHBOR CLASSIFICATION

Alejandro Flores-Velazco,
Doctor of Philosophy, 2022

Dissertation directed by: Professor David M. Mount
Computer Science

Given a set $P$ of $n$ labeled points in a metric space $(\mathcal{X}, \mathsf{d})$, the *nearest-neighbor rule* classifies an unlabeled query point $q \in \mathcal{X}$ with the class of $q$'s closest point in $P$. Despite the advent of more sophisticated techniques, nearest-neighbor classification is still fundamental for many machine-learning applications. Over the years, this has motivated numerous research aiming to reduce its high dependency on the size and dimensionality of the data. This dissertation presents various approaches to reduce the dependency of the nearest-neighbor rule from $n$ to some smaller parameter $k$, that describes the intrinsic complexity of the class boundaries of $P$. This is of particular significance as it is usually assumed that $k \ll n$ on real-world training sets.

One natural way to achieve this dependency reduction is to reduce the training set itself, selecting a subset $R \subseteq P$ to be used by the nearest-neighbor rule to answer incoming queries, instead of using $P$. Essentially, reducing its dependency from $n$, the size of $P$, to the size of $R$. We propose different techniques to select $R$, all of which select subsets whose sizes are proportional to $k$, and have varying degrees of correct classification guarantees.

Another alternative involves building data structures designed to answer these classification queries, bypassing the preprocessing step of reducing $P$. We propose the Chromatic AVD to answer $\varepsilon$-approximate nearest-neighbor classification queries, and whose query times and storage requirements dependent on $k_\varepsilon$, which describes the intrinsic complexity of the $\varepsilon$-approximate class boundaries of $P$.

# ALGORITHMS AND DATA STRUCTURES FOR FASTER NEAREST-NEIGHBOR CLASSIFICATION

by

Alejandro Flores-Velazco

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2022

Advisory Committee:
Professor David M. Mount: *Advisor, Chair*
Professor Yancy Diaz-Mercado: *Dean's Representative*
Professor MohammadTaghi HajiAghayi
Professor John P. Dickerson
Professor Hanan Samet
Professor Samir Khuller: *External Member*

*To Nelly.*

# Table of Contents

# Chapter 1: Introduction

In the context of non-parametric classification in *machine learning*, we are given a training set $P$ which consists of $n$ points in a metric space $(\mathcal{X}, \mathsf{d})$, where $P \subseteq \mathcal{X}$ and metric $\mathsf{d} : \mathcal{X}^2 \to \mathbb{R}^+$ defines the distance between any two points in $\mathcal{X}$. Additionally, this training set $P$ is partitioned into a finite set of *classes*, meaning that each point $p \in P$ is assigned a *label* $l(p)$ that indicates the class to which $p$ belongs to. Finally, given an *unlabeled* query point $q \in \mathcal{X}$, the goal of a *classifier* is to predict $q$'s label using the training set $P$.

The *nearest-neighbor rule* (or *nearest-neighbor classifier*) is among the best-known classification techniques [1]. It classifies any query point $q \in \mathcal{X}$ with the label of its closest point in $P$ according to the distance function $\mathsf{d}$. This idea can be generalized to predict $q$'s class using its first $\mathsf{k}$ nearest-neighbors in $P$, instead of only one. Such generalized approach is known to as the $\mathsf{k}$ nearest-neighbor classifier, or simply $\mathsf{k}$-NN. Throughout this book, we focus exclusively on the 1-NN classifier.

Despite being conceptually simple, numerous results show that the nearest-neighbor rule exhibits good classification accuracy both experimentally and theoretically [2–4]. In fact, its probability of error is bounded by twice the Bayes probability of error, which is the best achievable error by any theoretical classifier. However, the nearest-neighbor rule is often criticized for its high space and time complexities, as a straightforward implementation of this approach implies storing all the points of $P$ to answer such queries. This would therefore make the nearest-neighbor rule be highly dependent on the size $n$ of the training set $P$.

This drawback raises an important question of whether it is possible to reduce the dependency of the nearest-neighbor classifier, from $n$ to some smaller parameter. In this thesis, we proof this is indeed possible, proposing new approaches for nearest-neighbor classification that are dependent on some parameter $k$, where commonly $k \ll n$ on real training sets. This parameter is defined as the number of *border points* of $P$, and depicts the inherent complexity of the boundaries between points of different classes in the training set.

Some of the approaches presented in this thesis deal with computing a subset of $P$ whose size is dependent on $k$, which can then be used by the nearest-neighbor rule to answer classification queries, instead of using the entire training set. These are known as training set reduction techniques, and are explored in Chapters 3 to 5. Additionally, Chapter 6 explores a completely different approach, proposing a tailor-made data structure that can directly answer nearest-neighbor classification queries, thus bypassing the preprocessing step of having to reduce the training set. In terms of the underlying metric space, Chapters 3, 4 and 6 assume $P$ lies in $\mathbb{R}^d$ for constant dimension $d$, and using $\ell_2$ as the distance metric, while Chapter 5 deals with training sets in general metric spaces. Finally, in terms of the model of computation, Chapters 3 and 4 assume that nearest-neighbor queries are computed exactly, while Chapters 5 and 6 allow multiplicative approximation errors when answering such classification queries.

## Preliminaries

The set of *border points* 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 point in $\mathcal{X}$. Formally, two points $p, \hat{p} \in P$ are border points of $P$ if they belong to different classes, and there exist some point $q \in \mathcal{X}$ 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.

For any given point $p \in P$, define an *enemy* of $p$ to be any point of $P$ in a different class than $p$. The *nearest-enemy* of $p$, denoted $\mathrm{ne}(p)$, is the closest such point according to metric $\mathsf{d}$. Finally, define the *nearest-enemy distance* of $p$ to be $\mathsf{d}_{\mathrm{ne}}(p) = \mathsf{d}(p, \mathrm{ne}(p))$, and the *nearest-enemy ball* of $p$ to be the metric ball centered at $p$ with radius $\mathsf{d}_{\mathrm{ne}}(p)$. Let $\kappa$ be the number of distinct nearest-enemy points of $P$, that is, the cardinality of the set $\{\mathrm{ne}(p) \mid p \in P\}$. Similarly, denote the nearest-neighbor of $p$ as $\mathrm{nn}(p)$, and the nearest-neighbor distance as $\mathsf{d}_{\mathrm{nn}}(p) = \mathsf{d}(p, \mathrm{nn}(p))$.

In fact, in this thesis we prove that every nearest-enemy of $P$ is also a border point, implying that $\kappa \leq k$. Thus, we will use these two parameters to analyze our proposed approaches, showing their dependency on the complexity of the boundaries between points of different classes in $P$.

Through a suitable uniform scaling, we may assume that the *diameter* of $P$ (*i.e.*, the maximum distance between any two points in the training set) is 1. The *spread* of $P$, denoted as $\Delta$, is defined to be the ratio between the largest and smallest distances in $P$. Define the *margin* of $P$, denoted $\gamma$, to be the smallest nearest-enemy distance in $P$. Clearly, this implies that $1/\gamma \leq \Delta$.

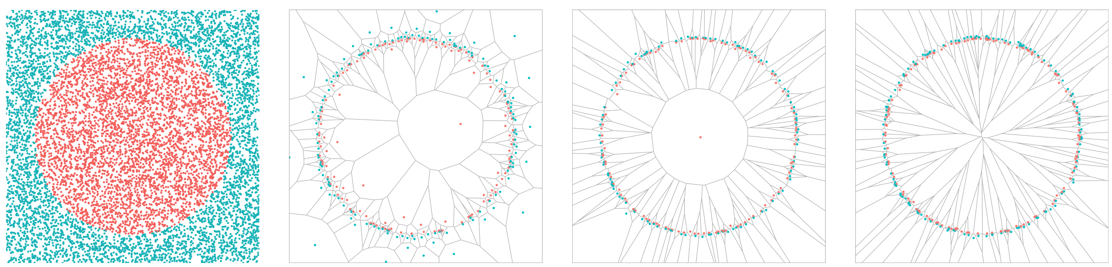
Additionally, a metric space $(\mathcal{X}, \mathsf{d})$ is said to be *doubling* [5] if there exist some bounded value $\lambda$ such that any metric ball of radius $r$ can be covered with at most $\lambda$ metric balls of radius $r/2$. Its *doubling dimension* is the base-2 logarithm of $\lambda$, denoted as $\mathrm{ddim}(\mathcal{X}) = \log \lambda$. Many natural metric spaces of interest are doubling, including $d$-dimensional Euclidean space whose doubling dimension is $\Theta(d)$.

## 1.1 Training Set Reduction

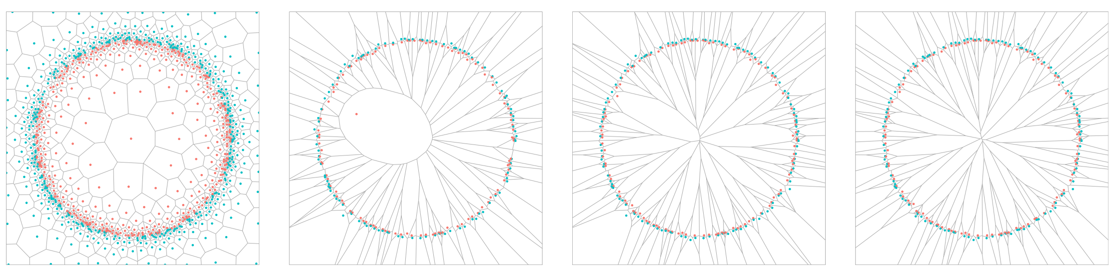
The idea behind training set reduction techniques is to select a subset of the original training set $P$, and then use this reduced set to answer any nearest-neighbor classification queries. Evidently, the goal is to select a subset as small as possible, subject to maintaining the classification accuracy of the nearest-neighbor classifier.

By definition, if instead of applying the nearest-neighbor rule with the entire training set $P$ we use the set of border points of $P$, the complexity of answering classification queries is now dependent on $k$ instead of $n$, while still obtaining the same classification for any query point in $\mathcal{X}$. In other words, this approach would maintain the same boundaries between points of different classes, before and after reducing the training set. For this reason, algorithms for finding the set of border points of $P$ are known as *boundary preservation* algorithms.

For training sets $P \subset \mathbb{R}^2$ in the Euclidean plane, 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. For almost three decades, the best result for training sets in $\mathbb{R}^d$, assuming constant $d$, was Clarkson's [7] algorithm that runs in $\mathcal{O}(\min(n^3, kn^2 \log n))$ worst-case time. Recently, Eppstein [8] proposed a significantly faster algorithm for the $d$-dimensional Euclidean case, which runs in $\mathcal{O}(n^2 + nk^2)$ worst-case time. In Chapter 3, we propose an improvement over Eppstein's algorithm [8] to compute the set of border points of any training set $P \subset \mathbb{R}^d$, where dimension $d$ is assumed to be constant. Moreover, our algorithm reduces the complexity of computing the set of border points of $P$ to $\mathcal{O}(nk^2)$ worst-case time, where $k$ is the size of such subset.



(a) Set $P$ ($10^4$ pts) (b) CNN (281 pts) (c) FCNN (222 pts) (d) MSS (272 pts)



(e) NET (875 pts) (f) SFCNN (220 pts) (g) RSS (233 pts) (h) VSS (233 pts)

Figure 1.1: Illustrative example of the subsets selected by different condensation algorithms from an initial training set $P$ in $\mathbb{R}^2$ of $10^4$ points. It includes existing algorithms (b)-(e), along with new algorithms introduced by our work (f)-(h).

Another approach for training set reduction is called condensation. Formally, the problem of *nearest-neighbor condensation* consists of finding a consistent subset of $P$, where a subset $R \subseteq P$ is said to be *consistent* [9] if and only if for every point $p \in P$ its nearest-neighbor in $R$ is of the same class as $p$. Intuitively, $R$ is consistent if and only if all points of $P$ are correctly classified using the nearest-neighbor rule over $R$. Compared to the boundary preservation techniques, using consistent subsets only guarantees the correct classification of the points of $P$, while using the set of border points extends this guarantee for every point of $\mathcal{X}$.

While finding subsets of $P$ that are consistent can be done efficiently, computing minimum cardinality consistent subsets is much harder. In fact, it is known to be an NP-hard problem [10–12], even to approximate within practical factors. Therefore, most research has focused on proposing practical heuristics to find either consistent subsets (for comprehensive surveys, see *e.g.*, [13–15]). Some of the best know heuristic algorithms for this problem are known as CNN [9], FCNN [16], and MSS [17], the last two being considered state-of-the-art for computing consistent subsets in $\mathcal{O}(n^2)$ time. However, while these heuristics have been extensively studied experimentally [18], theoretical results are scarce, with no upper-bounds known for the size of the subsets selected by these algorithms. In Chapter 4, we present the first theoretical results on upper-bounding the subset sizes of condensation algorithms. In particular, we show that FCNN and MSS can not be bounded in terms of $k$, and propose new quadratic-time algorithms called SFCNN, RSS and VSS, that can be effectively upper-bounded in terms of $k$ (see Figure 1.1 for illustrative examples on their selected subsets).

So far, these approaches for training set reduction have made a key assumption: that after reduction, the nearest-neighbor rule will answer classification queries (*i.e.*, compute nearest-neighbors on the reduced training set) exactly. However, in practice, nearest-neighbors are frequently computed approximately rather than exactly. In this context, given an approximation parameter $\varepsilon \in [0, 1]$, a query point $q \in \mathcal{X}$ can be assigned the class of any point of $P$ whose distance to $q$ is at most $1+\varepsilon$ times the distance from $q$ to its true nearest-neighbor. Sadly, the classification guarantees given by both boundary preservation and condensation algorithms, rely on the assumption that nearest-neighbors are computed exactly, and break when approximate query answers are allowed. In Chapter 5, we propose a framework for training set reduction that is sensitive to these approximations, along with a characterization of $\varepsilon$-coresets for the problem of nearest-neighbor classification.

## 1.2 Nearest-Neighbor Search vs Classification

Due to its conceptual closeness, the problem of nearest-neighbor classification is extremely related to the problem of nearest-neighbor search. Evidently, one can reduce the problem of classifying a query point $q \in \mathcal{X}$ via the nearest-neighbor rule, to simply compute $q$'s closest point in the training set, and assign $q$ to the class of such point. Unsurprisingly, this approach is standard for nearest-neighbor classification, and under such problem reduction, there are only two alternatives to improve the complexity of answering these classification queries. Either by reducing the training set (as explained in the previous section), or by improving the techniques to answer nearest-neighbor queries, which is a known and extensive line of research [19–22].

However, there exist an alternative approach towards achieving more efficient nearest-neighbor classification. This would imply avoiding the reduction to the search problem, and instead having algorithms and data structures to directly compute the class of the nearest-neighbor of any given query point; *i.e.*, without computing the nearest-neighbor itself. In computational geometry, this is usually known as the *chromatic nearest-neighbor search* problem.

In Chapter 6, we propose a tailor-made data structure for approximate nearest-neighbor classification (or approximate chromatic nearest-neighbor search), which we call the *Chromatic* AVD. Given a training set $P$ in $d$-dimensional Euclidean space (assuming constant $d$ and the $\ell_2$ metric) and an approximation parameter $\varepsilon \in [0, \frac{1}{2}]$, we construct a quadtree-based partitioning of space to answer any classification query approximately. That is, for any query point $q \in \mathbb{R}^d$ this data structure returns the class of any of $q$'s valid $\varepsilon$-approximate nearest-neighbors in $P$. Moreover, the Chromatic AVD is designed as a simplification of state-of-the-art AVDs [20] for approximate nearest-neighbor search. Thus, reducing its dependency from $n$ to a parameter $k_\varepsilon$ that describes the complexity of the approximate class boundaries.

# Chapter 2: Literature Review

The problem of classification is of high relevance in the field of machine learning. It has motivated numerous approaches that are able to predict the class of a given query point, by constructing and using some model "trained" from a given training set. However, and despite the advent of more sophisticated techniques (*e.g.*, support-vector machines [23] and deep neural networks [24]), nearest-neighbor classification is still widely used in practice [25–27], proving its value in constructing resilient defense strategies against adversarial [28] and poisoning [29] attacks, as well as in achieving interpretable and reliable classification models [30, 31].

Its importance has motivated diverse research in many fields, from statistics to computational geometry. In this book, we approach the nearest-neighbor classification problem from the perspective of computational geometry, leveraging algorithms, data structures, and analysis techniques from this field. This chapter delves into the most important concepts related to this classification technique, as well as the previous work on ways to improve its efficiency.

## 2.1 Classification Boundaries

The concept of *boundaries* in classification can be vague, as it depends on the particularities of the classification model being studied. The terms *class* boundaries, *classification* boundaries, and *decision* boundaries often refer to the same concept. Broadly speaking, they refer to the separation between two regions in space, such that the model of interest classifies points lying on either side of the boundary (*i.e.*, on each region) with different classes.

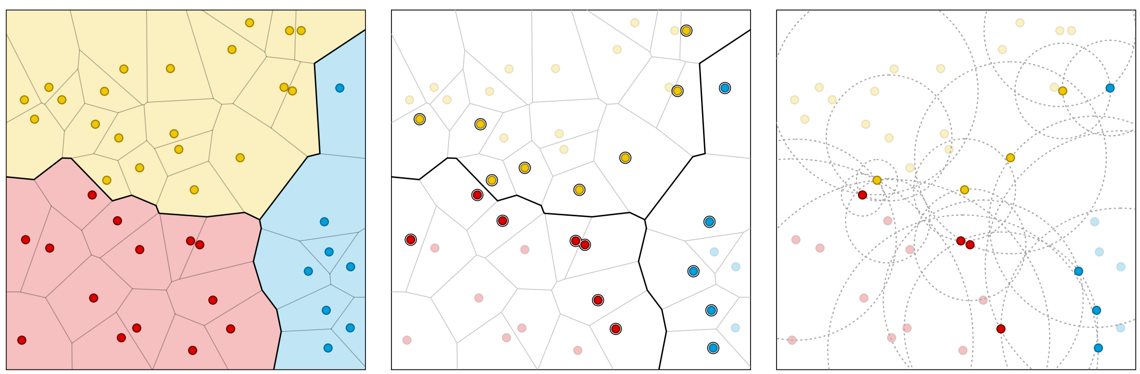
In order to understand the class boundaries of the nearest-neighbor rule, we first need to introduce a well-known concept in computational geometry called *Voronoi Diagrams*. The Voronoi diagram of a point set $P \subseteq \mathcal{X}$ is a partition of the underlying space $\mathcal{X}$ into different regions called *cells*. Each of these Voronoi cells has an associated point $p \in P$ (often call the *site* of the cell) such that for every point $q$ inside of the cell, $p$ is $q$'s closest point in $P$ according to the distance metric $\mathsf{d}$.

Throughout this book, all the figures depicting Voronoi diagrams will assume that the underlying metric space is Euclidean, and the distance function is the $\ell_2$ metric. Here, the Voronoi cell of each point $p \in P$ is a convex region of $\mathbb{R}^d$, and can be described as the intersection of $n-1$ closed halfspaces, each being the halfspace containing $p$ that is bounded by the bisector between $p$ and another point of $P$.

Two sites $p, \hat{p} \in P$ are said to be *Delaunay neighbors*, if and only if there exists a point $q$ in the underlying space that is part of the cells of both $p$ and $\hat{p}$. That is, if $q$

is equidistant to both $p$ and $\hat{p}$, and no other points of $P$ are closer to $q$. This defines the edges of the *Delaunay triangulation* of $P$, which is characterized as the dual graph of the Voronoi Diagram. Moreover, we can define the boundaries between the Voronoi cells of $P$ as the union of all such points $q$ in the space that are equidistant to at least two points/sites of $P$.

When considering the classes of these points, we can introduce a few useful concept. First, any edge of the Delaunay triangulation of $P$ that connects two points of different classes is called a *bichromatic edge*. Additionally, we can define the class boundaries of the nearest-neighbor rule similarly to the definition of the boundaries between the Voronoi cells of $P$. We say the *class boundaries* of $P$ are the union of points $q$ in the underlying space that are equidistant to at least two points of $P$ from different classes. Intuitively, these boundaries separate different *class regions*, where each such region is defined as the union of neighboring Voronoi cells corresponding to points of the same class. See Figure 2.1 for an illustrative example on these concepts.



(a) Class regions of $P$ (b) Border points of $P$ (c) Nearest-enemies of $P$

Figure 2.1: Example of a training set $P \in \mathbb{R}^2$ with points of three classes: *red*, *blue* and *yellow*. (a) shows the different class regions defined by the union of adjacent Voronoi cells of the same class, (b) highlights the border points of $P$, while (c) highlights the nearest-enemy points of $P$. Both (a) and (b) draw the class boundaries with solid black lines.

We say that the class boundaries of $P$ are defined by a subset of the training set called the set of *border points* of $P$. Basically, these border points are the endpoints of all the bichromatic edges of the Delaunay triangulation of $P$. 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 the underlying space 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. See Figure 2.1a and 2.1b for an example of a training set $P \subset \mathbb{R}^2$ and its set of border points. Throughout, we denote $k$ to be the total number of border points in the training set. Unsurprisingly, $k$ is key in understanding the complexity of the class boundaries of $P$, making it an ideal candidate to analyze the complexity of the classification problem.

Another concept that is useful to characterize the class boundaries of $P$ is that of nearest-enemy points. Just as defined in Chapter 1, we say the *nearest-enemy*[1] of a point $p \in P$ (denoted by $\text{ne}(p)$) is $p$'s closest point in $P$ of different class. Then, we let $\kappa$ be the number of distinct nearest-enemy points of $P$, that is, the cardinality of the set $\{\text{ne}(p) \mid p \in P\}$. See an example of this concept in Figure 2.1c. While not immediately obvious, we are able to (see Section 4.4) that every nearest-enemy point of $P$ is a also border point of $P$. Therefore, the set of nearest-enemy points is a subset of those points defining the class boundaries, and can then be used to characterize its complexity.

## 2.2 Boundary Preservation

In this section, we review known algorithms for finding the set of border points of the training set $P$, also referred to as *boundary preservation algorithms*. Note that for each of these algorithms, $P$ is assumed to lie in $d$-dimensional Euclidean space (*i.e.*, $P \subset \mathbb{R}^d$), and the distance function is assumed to be the $\ell_2$ metric.

### 2.2.1 Clarkson's algorithm

In 1994, Clarkson [7] proposed the first algorithm to compute the set of border points of a training set $P$ in $\mathbb{R}^d$. His algorithm runs in $\mathcal{O}(\min(n^3, kn^2 \log n))$ time, by leveraging linear programming formulations along with output-sensitive techniques to compute extreme points.

Starting with an empty selection, the algorithm incrementally adds newly found border points to its current selection. To achieve this, it performs a "non-borderness" test on every point $p$ of the training set. This test can only decide with certainty if $p$ is *not* a border point, but not otherwise. However, if the test's result is uncertain, Clarkson proposes a method to find some other point $\hat{p} \in P$ that is certainly a border point. Further details on this algorithm are left for the interested reader, and can be found in Section 5 of Clarkson's paper.

### 2.2.2 Eppstein's algorithm

In early 2022, Eppstein [8] proposed a new algorithm to find the set of border points of a given training set $P$ in $\mathbb{R}^d$ that runs in $\mathcal{O}(n^2 + nk^2)$ worst-case time. The algorithm's simplicity is striking, specially when considering the small progress achieved for this problem in the last three decades. Intuitively, the algorithm works somewhat as an implicit graph traversal, where the border points of $P$ are nodes in this graph, connected by certain implicit edges. Formally, the algorithm begins by selecting some initial set of border points of $P$, one point from every class region. From here, it uses a series of subroutines (which we group together and denote as the "*inversion method*") to find the remaining border points of $P$. To better understand this algorithm, we study its two phases: the initialization and search phases.

[1]This concept was first introduced by Dasarathy [32] in 1995 under the name of *nearest unlike neighbors* (or NUN), and was later renamed by Wilson & Martinez [33] in 1997 as nearest-enemies.

The initialization phase consists of finding a subset of points of $P$, such that all these must be border points, and there is at least one point for every class region of $P$. Eppstein's approach to find these points is to compute the Minimum Spanning Tree (MST) of $P$, identify the bichromatic edges of this MST, and then select the endpoints of all such edges. Evidently, this phase takes a total of $\mathcal{O}(n^2)$ time.

The search phase completes the algorithm, and consists of finding all the remaining border point of the training set. This phase iterates over all the currently selected points, and for each of these points $p$, it performs the inversion method on $p$, identifying a subset of border points that are "visible" by $p$. These are then selected by the algorithm, and the search continues. Basically, the inversion method unveils the edges of some implicit graph that are incident on $p$, allowing the algorithm to fully traverse this implicit graph. Once the inversion method has been applied on every selected point, the algorithm terminates with the guarantee of having selected all the border points of $P$. Each call of the inversion method takes $\mathcal{O}(nk)$ time, making the total runtime of this phase being $\mathcal{O}(nk^2)$.

It is important to note that Eppstein's algorithm works under the assumption that this implicit graph can have multiple connected components. Therefore, the initialization phase looks to select at least one point on every connected component, to guarantee that all the border points will eventually be discovered by the algorithm. In Chapter 3, we further improve Eppstein's algorithm by showing that in fact, this implicit graph has a single connected component. Hence, rendering the initialization phase unnecessary, and reducing the total runtime to $\mathcal{O}(nk^2)$.

### 2.2.3 Bremner *et al.*'s algorithm

This is a special case algorithm intended only for the planar case; *i.e.*, then the training set lies in $\mathbb{R}^2$. It was proposed by Bremner *et al.* [6] in 2003, as the result of a research workshop, and has a runtime of $\mathcal{O}(n \log k)$.

Their algorithm consists of two levels. First, the higher level algorithm, which consists of repeatedly guessing the value of $k$, and running a lower level algorithm under this assumption. This lower level algorithm works as follows: given some value $m$, if $m \geq k$ it finds the set of all border points of $P$, and otherwise it fails. Moreover, it does this within $\mathcal{O}((m^2 + n) \log m)$ time. Therefore, by repeatedly using values of $m = 2^{2^i}$, with $i = 0, 1, 2, \dots, \lceil \log \log k \rceil)$, and stopping when $m \geq k$ or $m \geq \sqrt{n}$, the total runtime of the higher level algorithm equals $\mathcal{O}(n \log k)$.

Interestingly, the lower level algorithm works in a similar way to Eppstein's algorithm. First, it finds an initial bichromatic edge of the Delaunay triangulation of $P$ (*i.e.*, identifying two border points), and then follows iteratively by applying a series of "*pivot operations*" on the border points found so far, to unveil new border points of $P$. If at some point, the algorithm finds more than $m$ border points, it fails as the assumption that $m \geq k$ is false. Otherwise, the algorithm terminates with the guarantee of having found all the border points of $P$.

The remaining details of this algorithm are left to the reader, and can be found in the original paper [6]. However, we describe the pivot operation in higher detail, as it would be leveraged throughout this book as a useful tool in some of our algorithms and analyses. Basically, any pivot operation receives three parameters: a point $p$, a vector $\vec{v}$, and a set of points $S \subset \mathbb{R}^d$. Intuitively, the pivot operation grows an empty ball with $p$ on its surface and its center on the direction of $\vec{v}$, until the ball hits a point of $\hat{p} \in S$. Then, the operation returns point $\hat{p}$. Despite its simplicity, this operation is a useful tool to find border points, as seen in the remaining of this book.

## 2.3 Nearest-Neighbor Condensation

Essentially, the problem of *nearest-neighbor condensation* consists of selecting some subset of the original training set $P$, subject to the classification accuracy of the nearest-neighbor rule not being "greatly reduced" when replacing the original training set by this subset of points. One way of achieving this would be to simply select the set of border points of $P$ via a boundary preserving algorithm. However, this approach is too strict, which has lead to the introduction of more relaxed criteria for the purpose of condensation.

In this section, we explore the main criteria used for condensation, hardness results, as well as the most relevant algorithms proposed for this problem.

### 2.3.1 Criteria

The central definition to understand the problem of condensation is that of consistent subsets, proposed by Hart [9] in 1968. We say a subset $R \subseteq P$ is *consistent* if and only if for every point $p \in P$ its nearest-neighbor in $R$ is of the same class as $p$. Intuitively, this means that any subset $R$ is consistent if and only if all points of $P$ are correctly classified using the nearest-neighbor rule "trained" on $R$. That is, with a nearest-neighbor rule that answers queries by searching among the points in $R$, instead of searching among the points of the original training set $P$.

Another criterion frequently used for condensation is known as *selectiveness*, and was proposed by Ritter *et al.* [34] in 1975. A subset $R \subseteq P$ is said to be *selective* if and only if for all points $p \in P$ its nearest-neighbor in $R$ is closer to $p$ than its nearest-enemy in $P$. Clearly selectiveness implies consistency, as the nearest-enemy distance in $R$ of any point of $P$ would at least be its nearest-enemy distance in $P$. In fact, selective subsets were introduced as a stricter version of consistency, hoping this would allow the existence of polynomial-time algorithms to find minimum cardinality selective subsets. As we will see in the following subsections, this would later be discovered to be impossible unless P=NP.

Recall that none of these two criteria, namely consistency and selectiveness, imply that every query point in $\mathcal{X}$ would be correctly classified after condensation. Instead, they simply guarantee the correct classification of the points in $P$. The only techniques that guarantee correct classification of every query point are boundary preservation algorithms, as explored in Section 2.2. It is easy to see that any subset that preserves the class boundaries of $P$ (*i.e.*, the set of border points of $P$) must be selective, and that any selective subset of $P$ must also be consistent. Therefore, any algorithm that finds subsets of $P$ holding any of these properties is considered to be a condensation algorithm.

### 2.3.2 Hardness Results

Clearly the stricter notion of condensation, involving the full preservation of the class boundaries of $P$, can be achieved in polynomial-time as described in Section 2.2. The relaxation to the consistent and selective criteria imply that subsets holding these properties can be computed even faster. These subsets always exist, as $P$ itself is both consistent and selective. Evidently, the more interesting research question deals with finding ideally small subsets of $P$ under these two condensation criteria.

Therefore, understanding the complexity of finding such subsets while minimizing their sizes becomes of major relevance. It took close to 30 years since consistency was originally defined in 1968 for the first hardness results to appear, and it took another 30 years to close some of the remaining gaps into understanding the hardness of approximation for these problems.

Denote MIN-CS and MIN-SS to be the problems of finding minimum cardinality consistent and selective subsets of $P$, respectively, we know that these two problems are NP-hard to solve, and their decision versions are NP-complete [10–12]. Originally, Wilfong [10] proved this complexity for when $P$ lies in Euclidean space, and restricting the result for MIN-CS on training sets with at least 3 classes. These results were later generalized by Zukhba [11] and Khodamoradi *et al.* [12] for the cases when $P$ has at least two classes, and also when it lies in some general metric space $(\mathcal{X}, \mathsf{d})$.

Hardness of approximation results have also been proposed for this problem. The first result dates back to 2000, when Nock & Sebban [35] found that unless NP $\subseteq$ DTIME($n^{\log\log n}$), the MIN-SS problem is NP-hard to approximate in polynomial-time within a factor of $(1 - o(1)) \ln n$. Later in 2014, Gottlieb *et al.* [36] found that the MIN-CS problem is NP-hard to approximate in polynomial-time within a factor of $2^{(\text{ddim}(\mathcal{X}) \log 1/\gamma)^{1-o(1)}}$, where $\text{ddim}(\mathcal{X})$ is defined as the doubling dimension of the space $\mathcal{X}$, and $\gamma$ is the margin or minimum nearest-enemy distance of $P$.

More recently, Chitnis [37] presented the first results on the parameterized complexity of the decision version of MIN-CS. This version of the problem decides whether there exists a consistent subset of $P$ of size $\leq m$, assuming $P \subset \mathbb{Z}^d$ and $\ell_p$ being the distance metric. Then, Chitnis proves two main results. First, that this problem is W[1]-hard parameterized by $m + d$, meaning that unless FPT=W[1], there is no $f(m, d) \cdot n^{\mathcal{O}(1)}$ time algorithm for any computable function $f$. Additionally, that under the Exponential Time Hypothesis (ETH) there is no $d \geq 2$ and computable function $f$ such that this problem can be solved in $f(m, d) \cdot n^{o(m^{1-1/d})}$ time.

### 2.3.3 Algorithms

Due to the importance of this problem, numerous algorithmic approaches have been proposed to compute such subsets. These include some optimal algorithms, as well as some case-specific, and approximation algorithms. However, due to the complexity of computing even close to optimal solutions for the condensation problem, most research has focused on proposing practical heuristics to find either consistent or selective subsets of $P$. For comprehensive surveys on these heuristics, see [13–15].

Here we review some of the most relevant algorithms proposed for this problem, in chronological order. In 1968, along with the definition of consistency, Hart [9] proposed the CNN (*Condensed Nearest-Neighbor*) algorithm to compute consistent subsets of $P$. Even though it has been widely used in the literature, CNN suffers from several drawbacks: its running time is cubic in the worst-case, and the resulting subset is order-dependent, meaning that the points selected are determined by the order in which they are considered by the algorithm. Additionally, CNN tends to select points far from the class boundaries, which is usually undesirable as these points are unlikely to contribute to such boundaries. To combat this, Gates [38] proposed the RNN (*Reduced Nearest-Neighbor*) algorithm in 1972, which consists of first running CNN and then proceed with a postprocessing technique to further reduce the subset. While this resolves the issue of selecting points far from the class boundaries, this approach still runs in worst-case cubic time and is order-dependent.

Also in 1975, Ritter *et al.* [34] proposed the SNN (*Selective Nearest-Neighbor*) algorithm, along with the definition of selective subsets. The authors hoped it would be possible to compute minimum cardinality selective subsets in polynomial time, and propose this algorithm to find them. Unsurprisingly, it was later proved that its runtime is worst-case exponential [10], even though Wilson & Martinez [33] claimed that the algorithm's average runtime is quadratic.

The new century saw a big leap in heuristic approaches for condensation. In particular, two algorithms stand out: MSS (*Modified Selective Subset*) proposed by Barandela *et al.* [17] in 2005, and FCNN (*Fast* CNN) proposed by Angiulli [16] in 2007. Both algorithms compute consistent and selective subsets, respectively. But more importantly, both algorithms run in quadratic worst-case time, and are order-independent. These characteristics, together with their ease of implementation, and specially, their exceptional performance in practice, have established these algorithms as state-of-the-art for condensation.

In 2014, Gottlieb *et al.* [36] proposed an approximation algorithm called NET, along with the almost matching hardness lower-bounds described in Section 2.3.2. The algorithm is fairly simple, consisting on just computing a $\gamma$-net of $P$ where $\gamma$ is $P$'s margin. This clearly results in a consistent subset, which can be proven to be a tight approximation of the MIN-CS problem. However, in practice, $\gamma$ tends to be small in real training sets, thus making the subsets selected by the NET algorithm of much higher cardinality than those selected by state-of-the-art heuristics.

Recently, in 2019 Biniaz *et al.* [39] proposed a subexponential-time algorithm for finding minimum cardinality consistent subsets of point sets $P \subset \mathbb{R}^2$ in the Euclidean plane, along with other case-specific algorithms for special instances of the problem in $\mathbb{R}^2$. Their main algorithm runs in $n^{\mathcal{O}(\sqrt{m})}$ where $m$ is the size of minimum cardinality consistent of $P$, which after the hardness results presented by Chitnis [37] in 2022, it has been proven to be asymptotically tight for the planar case.

It is important to note that, while the heuristics described in this section (*i.e.*, CNN, RNN, FCNN, and MSS) have been extensively studied experimentally [18], theoretical results are scarce. Before our work described in Chapter 4, little was known about any theoretical guarantees on the size of their selected subsets. There was a clear gap between practical heuristics without any theoretical guarantee, and approximation algorithms with poor performance in practice.

## 2.4 Nearest-Neighbor Search

Given a set $P \subseteq \mathcal{X}$ of $n$ points in a metric space $(\mathcal{X}, \mathsf{d})$ and a query point $q \in \mathcal{X}$, the goal of the nearest-neighbor search problem is to compute $q$'s closest point in $P$ according to the distance function $\mathsf{d}$. The most common assumption for this problem is that $P$ is given initially to be preprocessed into a data structure, while query points are later received as a stream to be answered using the prepocessed data structure.

In this section, we discuss different techniques to compute nearest-neighbors, either exactly or approximately. Additionally, we consider the related problem of chromatic nearest-neighbor search, which assuming that $P$ is a training set (*i.e.*, that each point in $P$ is labeled), its goal is to compute the class of each query point's nearest-neighbor, and not necessarily its nearest-neighbor itself.

### 2.4.1 Exact Search

There are two main approaches to compute nearest-neighbors exactly. The first, via exhaustive search over the entire point set $P$, which evidently implies linear query times and storage. Otherwise, the approaches usually apply point location algorithms over space partitioning data structures. For the low dimensional case of $d \leq 2$, it is possible to obtain $\mathcal{O}(\log n)$ query times and linear storage. Sadly, when the dimension is $d > 2$, the "*curse of dimensionality*" starts to kick in, and the overhead between query times and storage requirements grows increasingly fast. On either extreme, this means that we have solutions with logarithmic query time but roughly $\mathcal{O}(n^{d/2})$ storage [40], or solutions with linear storage but barely sublinear query times [41]. Therefore, the applicability of exact nearest-neighbor search becomes severely limited for high-dimensional data.

### 2.4.2 Approximate Search

In practice, nearest-neighbors are often computed approximately rather than exactly, as this relaxation allows for further improvements on query time and storage requirements. Formally, given an approximation parameter $\varepsilon \in [0, 1]$, the problem of *$\varepsilon$-approximate nearest-neighbor searching* (or $\varepsilon$-ANN) deals with computing a point $p \in P$ whose distance from the query point $q$ is within a factor of $1 + \varepsilon$ of $q$'s distance to its nearest-neighbor in $P$. We say that any such point $p$ is a valid $\varepsilon$-approximate nearest-neighbor of $q$, and can be retrived as a valid answer to $q$'s query.

This problem has been studied extensively, both theoretically [19–22, 42, 43] and from a practical standpoint [44, 45]. When the number of dimensions is low, most techniques involve some sort of space partitioning; *e.g.*, $k$d-Trees [46], Quadtrees [47–49], and Approximate Voronoi Diagrams [19, 20, 43]. Otherwise, alternatives for high-dimensional data include hashing techniques like Locality-Sensitive Hashing [21, 50], and proximity graph techniques like Hierarchical Navigable Small Worlds graphs [22]. However, throughout this dissertation we focus on low-dimensional data approaches, as many of our results have exponential dependency on the number of dimensions.

### 2.4.3 Chromatic Search

Evidently, given some query point $q$, to classify $q$ with the nearest-neighbor rule involves retrieving the class of $q$'s nearest-neighbor in the training set. Note that this is slightly different from retrieving the nearest-neighbor itself. Potentially, this difference opens the possibility of improvements over the standard search problem. In the context of computational geometry, this is sometimes referred to as the problem of *chromatic nearest-neighbor search*, where classes are intuitively described as colors.

Evidently, to answer *exact* chromatic queries there is one clear approach: to compute the set of border points of the training set, and use any of the techniques described in Section 2.4.1 to answer exact search queries over this subset. Despite having increased preprocessing times, this approach would lead to query times and storage requirements dependent on $k$ instead of $n$. However, it is unclear if it is possible to achieve similar results without having to recur to boundary preservation algorithms and the reduction to the standard search problem.

However, this is indeed possible in the approximate setting. Some work has been done along these lines by Mount *et al.* [51], showing that when query points are far from the class boundaries and surrounded by points of the same class, query times can be significantly reduced. However, these query times are still dependent on $n$. Moreover, the data structure itself dates back to 2000, and is based on standard search techniques of the time. Evidently, many improvements have been achieved in the last two decades on approximate nearest-neighbor searching, making this data structure clearly outdated.

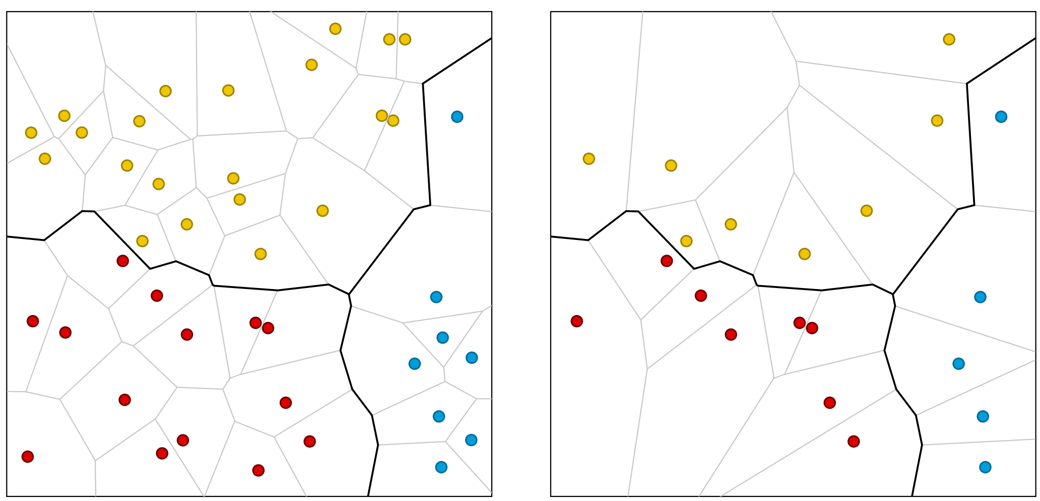
In Chapter 6, we propose new data structure for this problem called *Chromatic* AVD, thought as a simplification of state-of-the-art AVDs [20] from 2017. This new approach has query times and storage dependencies on a parameter $k_\varepsilon$ that describes the complexity of the approximate class boundaries (similarly to how $k$ describes the complexity of the exact class boundaries).

# Chapter 3: Boundary Preserving Algorithms

## 3.1 Introduction

As previously discussed, a common approach to reduce the dependency on $n$ of the nearest-neighbor rule is to reduce the training set itself. However, most training set reduction techniques offer limited guarantees on the effect that this reduction makes on the accuracy of the classifier. Only a handful of works [6–8] have proposed algorithms that guarantee the same classification of every query point, before and after the reduction took place. These are called *boundary preserving* algorithms, and are the focus of this chapter.

The results presented on this chapter can also be found here [52].



(a) Original training set $P$ (b) Border points of $P$

Figure 3.1: On the left, a training set $P \in \mathbb{R}^2$ with points of three classes: *red*, *blue* and *yellow*. On the right, a subset of these points corresponding to the set of border points of $P$. The solid black lines highlight the *boundaries* of $P$ between points of different classes. By definition, the class boundaries remain the same in both cases.

While other problems in the realm of training set reduction are NP-hard [10–12] to solve exactly (*e.g.*, those of finding minimum cardinality *consistent* subsets and *selective* subsets), the problem of preserving the class boundaries of the nearest-neighbor rule is tractable. As discussed in Sections 2.1 and 2.2, this problem is equivalent to that of finding the set of border points of $P$.

The set of *border points* 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, as seen in Section 2.1, 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 3.1 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 applying the nearest-neighbor rule 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 $k \ll n$.

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. Until very recently, the best result for the higher dimensional case was that of Clarkson [7]. 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 [8] proposed a significantly faster algorithm for the $d$-dimensional Euclidean case, which runs in $\mathcal{O}(n^2 + nk^2)$ worst-case time.

In this chapter, we propose an improvement over Eppstein's algorithm [8] to compute 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.

## 3.2 Eppstein's algorithm

This algorithm is strikingly simple, yet full of interesting ideas (see a formal description in Algorithm 1), and it 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 $P$.

### 3.2.1 The Initialization Phase

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.

**Algorithm 1:** Eppstein's algorithm [8] 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** $R$

### 3.2.2 The Search Phase

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 $P$.

### 3.2.3 The Inversion Method

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 [8]. However, for the purposes of this chapter we only need a property presented in Lemma 3 of [8]: 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.

## 3.3 A Simpler Initialization

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 $P$.

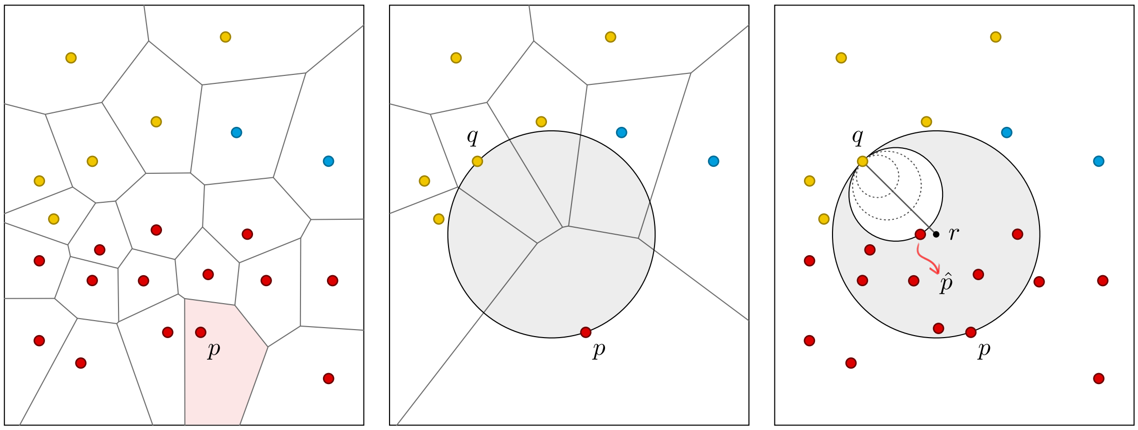
**Input:** Initial training set $P$
**Output:** The set of border points of $P$

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 [8] 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 [8] 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 the search phase 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. (i) By applying the inversion method to any point of $P$, not necessarily a border point, we must prove that all reported points are border points of $P$. This is established in Lemma 3.1, generalizing the statement of Lemma 6 of [8] for non-border points. (ii) For any class boundary of $P$, once the algorithm selects a point from this boundary, we must prove that it will eventually select every other point defining the same boundary. This is originally proved in Lemma 10 [8], however, we provide simpler proofs in Lemmas 3.2 and 3.3. (iii) Given two disconnected boundaries separated by a class region, we must 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 established in Lemma 3.4.

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 3.5 and 3.6.




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

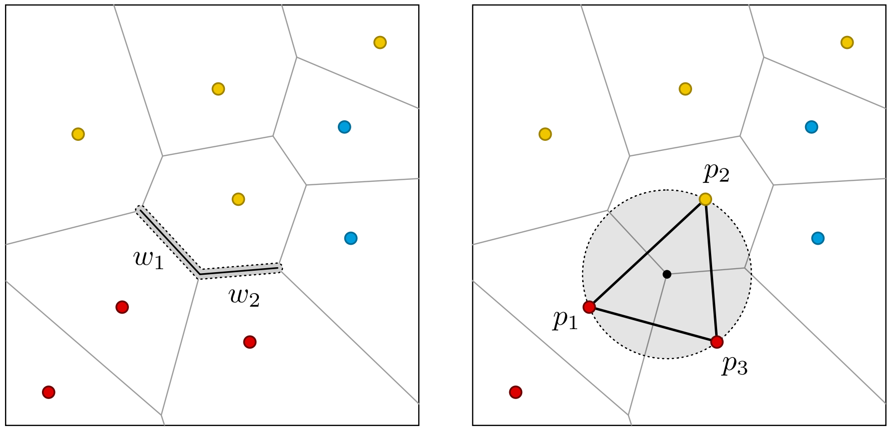
Figure 3.2: Example showing the inversion method from any point $p \in P$. On the left, training set $P$. The middle figure shows every non red point of $P$, except for $p$ itself, along with a point $q$ selected from the inversion method on $p$. On the right, we see evidence that $q$ is a border point of $P$.

### 3.3.1 Correctness Proof

**Lemma 3.1.** *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 [8], 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 3.2a and 3.2b). 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 3.2b, 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 3.2c). 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 $P$. □




(a) (b)

Figure 3.3: 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.

### 3.3.2 Completeness Proof

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 3.4 illustrates a training set in $\mathbb{R}^2$ with points of three classes, whose Voronoi Diagram describes five class regions and two class boundaries.

**Lemma 3.2.** *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 3.3). 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. □

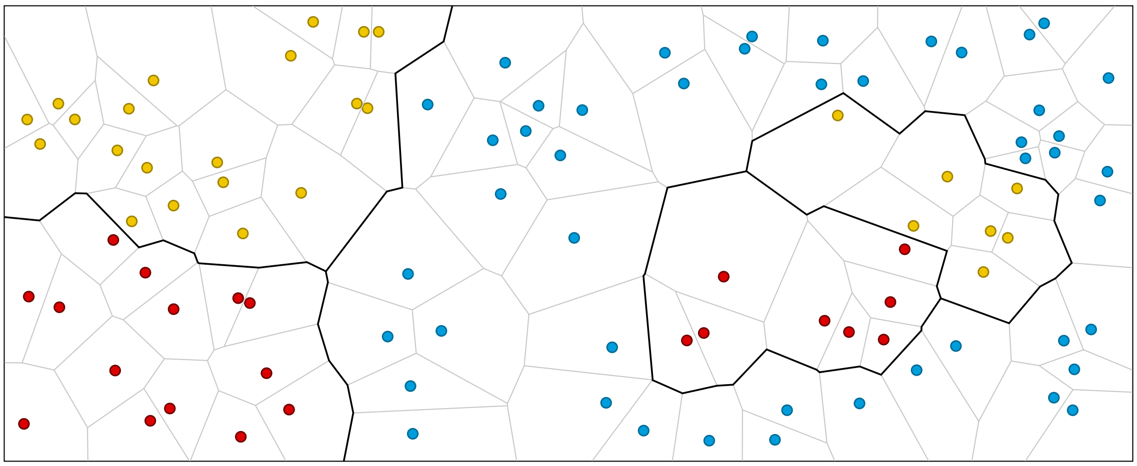


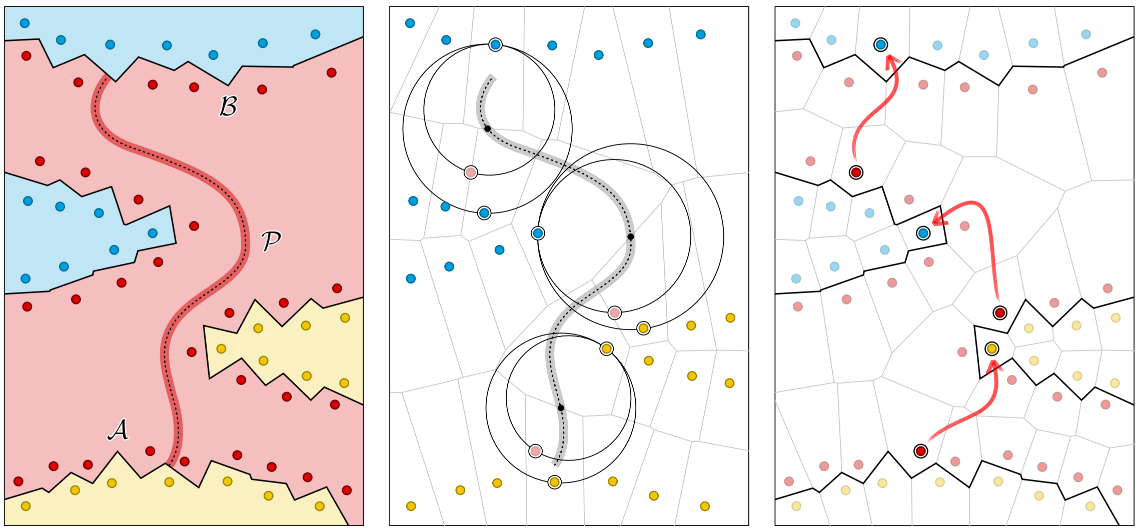
Figure 3.4: A training set with five class regions (one *blue*, two *red*, and two *yellow* regions), along with two disconnected class boundaries that separate all these regions. On the left, a boundary separating the blue region and the leftmost yellow and red regions. On the right, a boundary that separates the same blue region and the remaining red and yellow regions.

**Lemma 3.3.** *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 3.2 and the definition of a class boundary of the training set $P$. It remains to show what happens with boundaries that are disconnected.

**Lemma 3.4.** *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 3.5a). 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 3.5 for a visual description.




(a) (b) (c)

Figure 3.5: 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 3.5b). 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 3.1. Additionally, these border points could be defining other boundaries apart from $\mathcal{A}$ and $\mathcal{B}$, as seen in Figure 3.5b.

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 3.5b, 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 3.3 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 3.5.** *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 [8]. However, the correctness and completeness of our algorithm follows from Lemmas 3.1 to 3.4.

First, we know by Lemmas 3.1-3.3 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 3.4 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 3.6.** *Leveraging Chan's algorithm [53] 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*

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

*time for all constant dimensions $d > 3$.*

Just as with Eppstein's original algorithm, we can use Chan's randomized algorithm [53] 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 3.5.

# Chapter 4: Condensation Algorithms

## 4.1 Introduction

As previously discussed, preserving the class boundaries of a given training set $P$ is a desirable objective towards the goal of more efficient nearest-neighbor classification. However, the high complexity of computing the entire set of border points of $P$ has motivated the study of alternative approaches for training set reduction, which include the problems of computing either *consistent* or *selective* subsets of $P$. Thus, even though computing minimum cardinality subsets that are either consistent or selective is known to be NP-hard [10–12], computing non-minimal subsets holding these properties can be done efficiently. Algorithms that compute such subsets are frequently referred to as nearest-neighbor condensation heuristics, as they "heuristically" select points that are close to the class boundaries induced by $P$. Therefore, such selected subsets induce new class boundaries that resemble the original boundaries, albeit not exactly the same.

However, one significant shortcoming in research on practical nearest-neighbor condensation algorithms is the lack of theoretical results on the sizes of their selected subsets, where typically, their performance has been established experimentally. This chapter presents the first theoretical guarantees on the performance of both new and existing condensation algorithms. These results have been published in [54–56].

The bounds presented in this chapter are established with respect to the size of two well-known and structured subsets of points: $(i)$ the set of all nearest-enemy points of $P$ of size $\kappa$, and $(ii)$ the set of border points of $P$ of size $k$. Additionally, some bounds depend on the minimum nearest-enemy distance of $P$ denoted as $\gamma$.

| Algorithm | Subset size |
|---|---|
| CNN | $\mathcal{O}(\kappa \log 1/\gamma)$ |
| FCNN | Unbounded *w.r.t.* $k$ and $\kappa$ |
| SFCNN • | $\mathcal{O}(\kappa \log 1/\gamma)$ |
| NET | $\mathcal{O}(\kappa \log 1/\gamma)$ |
| MSS | Unbounded *w.r.t.* $k$ and $\kappa$ |
| RSS • | $\mathcal{O}(\kappa)$ |
| VSS • | $\leq k$ |

Table 4.1: Contributions on the worst-case analysis on the sizes of subsets selected by different condensation algorithms. Those marked with • are new algorithms.

## 4.2 Lower Bounds

If we hope to leverage both $k$ and $\kappa$ to analyze the worst-case performance of different nearest-neighbor condensation heuristics, we first need to establish realistic expectations on what upper-bounds can be achieved. Therefore, before analyzing each of these algorithms individually, we introduce two lower-bounds on the size of consistent subsets, one in terms of $\kappa$ and the other in terms of $k$.

First, lets recall the definition of *consistency*. A subset $R \subseteq P$ is said to be *consistent* [9] if and only if for every $p \in P$ its nearest-neighbor in $R$ is of the same class as $p$. Intuitively, $R$ is consistent if and only if all points of $P$ are correctly classified with the nearest-neighbor rule *w.r.t.* $R$.

**Theorem 4.1.** *There exists a training set $P \subset \mathbb{R}^d$ with $\kappa$ number of nearest-enemy points, for which any consistent subset has $\Omega(\kappa c^{d-1})$ points, for some constant $c$.*

*Proof.* Lets construct a training set $P$ in $d$-dimensional Euclidean space, which contains points of two classes: *red* and *blue*. Consider the following arrangement of points: create a red point $p$, and take *every* point at distance 1 from $p$ as a blue point. Simply, point $p$ plus the points on the surface of the unit ball centered at $p$.

Take any consistent subset of $P$ and consider some point $\hat{p}$ in this subset, along with the bisector between $p$ and $\hat{p}$. The intersection between this bisector and the unit ball centered at $p$ describes a cap of the ball of height $1/2$. Any point located inside this cap is closer to $\hat{p}$ than $p$, and hence, correctly classified. Clearly, by definition of consistency, all points in the ball must be covered by at least one cap. By a simple packing argument, we know such a covering needs $\Omega(c^{d-1})$ points, for some constant $c$. The training set constructed so far has only two nearest-enemy points; *i.e.*, the red point $p$ and the one blue point closest to $p$ (assuming general position). Then, we can repeat this arrangement $\kappa/2$ times using sufficiently separated center points. This gives us a new training set $P$ with $\kappa$ nearest-enemy points in total, for which any consistent subset has size $\Omega(\kappa c^{d-1})$. □

**Theorem 4.2.** *There exists a training set $P \subset \mathbb{R}^d$ with $k$ number of border points, for which any consistent subset has at least $k$ points.*

Basically, Theorem 4.1 shows that the best upper-bound that can proved on the size of the subsets selected by condensation algorithm, in terms of $\kappa$, is $\mathcal{O}(kc^{d-1})$. Similarly, Theorem 4.2 proves that in terms of $k$, the best upper-bound is $k$ itself. Clearly, these results provide a steady floor on which to compare the results presented in the coming sections of this chapter.

## 4.3 Consistent Subsets

In this section, we explore several algorithms that compute consistent subsets of $P$, and present a formal worst-case analysis on the size of the subsets that these algorithms select. Namely, we study the CNN, FCNN, and SFCNN algorithms.

### 4.3.1 CNN

The CNN algorithm (or *Condensed Nearest-Neighbor*) was the first algorithm to be proposed for computing consistent subsets [9]. In fact, it was introduced right along the definition of consistency in Hart's seminal paper. For more than 30 years, and until the introduction of the FCNN algorithm, CNN was considered to be the state-of-the-art among condensation heuristics.

**Algorithm 3:** CNN

**Input:** Initial training set $P$
**Output:** Consistent subset $R \subseteq P$

$R \leftarrow \phi$
**while** true **do**
  $R' \leftarrow R$
  **foreach** $p_i \in P$, *where* $i = 1 \dots n$ **do**
    Let $\text{nn}(p_i, R)$ be the nearest-neighbor of $p_i$ *w.r.t.* $R$
    **if** $l(p_i) \neq l(\text{nn}(p_i, R))$ **then**
      $R \leftarrow R \cup \{p_i\}$
  Exit the while loop if $R = R'$
**return** $R$

This algorithm is fairly simple (see Algorithm 3 for a formal description). Beginning with an empty $R$ set, it repeatedly scans all the points of $P$. For each point $p \in P$, the algorithm checks if its nearest-neighbor within the current $R$ set is of the same class as $p$. That is, the algorithm checks if $p$ would be correctly classified using $R$. If not, $p$ is added to $R$. Otherwise, the algorithm continues checking the next point of $P$ being scanned. Note that $P$ can be scanned multiple times, as adding a point to $R$ can induce the misclassification of a previously checked point.

Evidently, this is a cubic-time algorithm. More specifically, a straightforward implementation of CNN yields a $\mathcal{O}(n^2 m)$ worst-case time complexity, where $m$ denotes the size of the selected subset. Another characteristic about this algorithm is that it is *order-dependent*, meaning that the resulting subset depends on the order in which the points of $P$ were scanned by the algorithm. Unsurprisingly, this is an undesirable property for practitioners.

**Theorem 4.3.** *The* CNN *algorithm selects* $\mathcal{O}\left(\kappa \log \frac{1}{\gamma}\right)$ *points.*

*Proof.* This follows by a charging argument on every nearest-enemy point in $P$. Therefore, consider one such point $p \in \{\text{ne}(r) \mid r \in P\}$ and some value $\sigma \in [\gamma, 1]$, and define $R_{p,\sigma}$ to be the subset of points selected by CNN whose nearest-enemy is $p$, and whose distance to $p$ is between $\sigma$ and $2\sigma$. That is, $R_{p,\sigma} = \{r \in R \mid \text{ne}(r) = p \wedge \mathsf{d}(r, p) \in [\sigma, 2\sigma)\}$. All these subsets define a partitioning of $R$ when considering all nearest-enemy points of $P$, and values of $\sigma = \gamma\, 2^i$ for $i = \{0, 1, 2, \dots, \lceil \log \frac{1}{\gamma} \rceil\}$.

Now, lets consider any two points $a, b \in R_{p,\sigma}$ in these subsets. We want to prove that $\mathsf{d}(a, b) \geq \sigma$. Evidently, if these two points belong to different classes, then by definition we have that $\mathsf{d}(a, b) \geq \sigma$. Thus, lets consider the case when both points belong to the same class. Lets assume *w.l.o.g.* that point $a$ was added to $R$ before point $b$. Note that when the algorithm selected point $b$, the following must be true $\mathsf{d}(a, b) \geq \mathsf{d}_{\text{ne}}(b, R)$, as otherwise point $b$ would have been correctly classified using $R$. Moreover, by our partitioning of $R$, we also know that $\mathsf{d}_{\text{ne}}(b, R) \geq \mathsf{d}_{\text{ne}}(b) \geq \sigma$.

Thus, we have proved that $\mathsf{d}(a, b) \geq \sigma$. By a simple packing argument based on $d$-dimensional Euclidean balls, we have that $|R'_{p,\sigma}| \leq 5^d$. Altogether, by counting over all the $R_{p,\sigma}$ sets for every nearest-enemy in the training set and values of $\sigma$, the size of the subset $R$ selected by CNN is upper-bounded by $|R| \leq \kappa \lceil \log 1/\gamma \rceil 5^{d+1}$. Assuming $d$ to be constant, this completes the proof. □

Moreover, this analysis is tight. A simple example showcasing this behavior can be described as follows: consider a training set $P \subset \mathbb{R}$ in 1-dimensional Euclidean space, and let $\gamma \ll 1$ be a small value. Place a single *red* point in the origin, and make all the points in the segment $[\gamma, 1]$ *blue* points. Now consider running CNN on this training set, scanning the points starting with the single red point and following with the blue points in decreasing order on their coordinate (*i.e.*, in decreasing order on their distance to the origin, or also, their nearest-enemy distance). It is easy to see that this would make CNN select $\mathcal{O}(\log 1/\gamma)$ points. This arrangement can be repeated to increase the value of $\kappa$, and can also be discretized to make $n$ finite and independent from $k$, $\kappa$, and $\gamma$.

The result of Theorem 4.3 confirms the empirical behavior observed for CNN, where we see many selected points located far from the boundaries of $P$ (see Figure 1.1b). Evidently, the many drawbacks of the CNN algorithm (*e.g.*, its empirical behavior, order-dependence, and cubic worst-case time complexity) has motivated numerous research on improved approaches towards computing consistent subsets.

### 4.3.2 FCNN

The FCNN algorithm (or *Fast* CNN) became the state-of-the-art algorithm for computing consistent subsets [16] improving over many of CNN's drawbacks. When introduced, it represented a big leap forward on practical methods for nearest-neighbor condensation, being among the first worst-case quadratic time algorithms for this problem, with an implementation that runs in $\mathcal{O}(nm)$ time, where $m$ is the size of its selected subset. Additionally, its selected subset is order-independent, which is a highly desirable property among practitioners. More importantly, in practice, this algorithm significantly outperforms other condensation techniques.

This is an iterative algorithm that incrementally builds a consistent subset $R$ of $P$ (see Algorithm 4). First, it begins by selecting the set of centroids of each class, and then continues with an iterative process until the subset becomes consistent. During each iteration, the algorithm identifies all points of $P$ that are misclassified with the current subset $R$ (*i.e.*, whose nearest-neighbor is of different class), and adds some of these points to the subset. In particular, for every point $p$ already in the subset, FCNN selects one *representative* among all the points not yet selected, whose nearest-neighbor is $p$, and that belong to a different class than $p$. That is, the

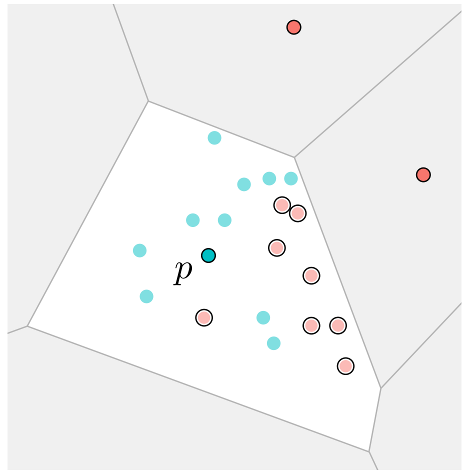



Figure 4.1: Example of the *voren* function for any point $p \in R$. Light-colored points belong to $P \setminus R$, and points in $\text{voren}(p, R, P)$ are outlined (defined as the enemies of $p$ inside its voronoi cell *w.r.t.* $R$).

representative is selected from the set $\text{voren}(p, R, P)$ which is defined as:

$$\text{voren}(p, R, P) = \{q \in P \mid \text{nn}(q, R) = p \wedge l(q) \neq l(p)\}$$

See Figure 4.1 for an illustrative example. Usually, the representative chosen is the one closest to $p$, although different approaches can be used. Finally, during each iteration, the representative of each point in $R$ is added to the subset (all in a batch operation). This is repeated until no misclassified points are left; *i.e.*, until no point of $R$ has a representative to choose.

**Algorithm 4:** FCNN

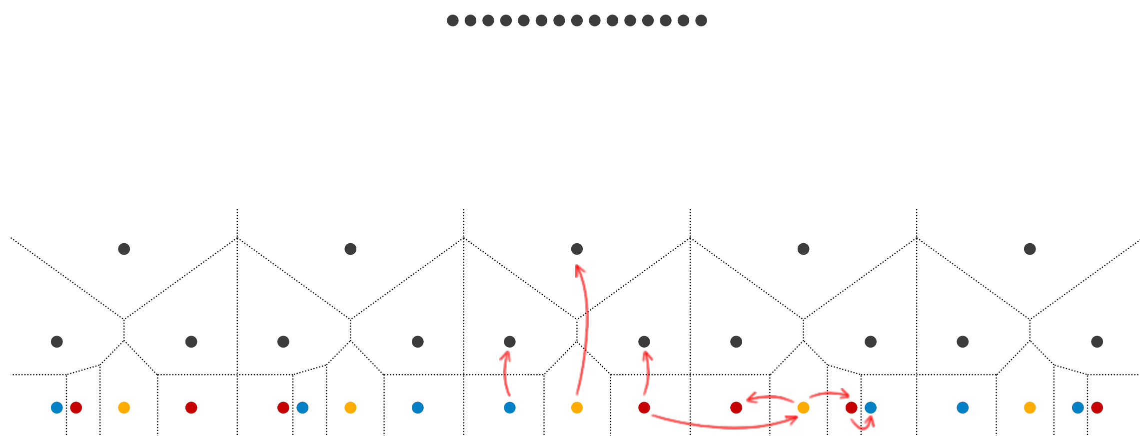
**Input:** Initial training set $P$
**Output:** Consistent subset $R \subseteq P$

$R \leftarrow \phi$
$S \leftarrow \text{centroids}(P)$
**while** $S \neq \phi$ **do**
  $R \leftarrow R \cup S$
  $S \leftarrow \phi$
  **foreach** $p \in R$ **do**
    $S \leftarrow S \cup \{\text{rep}(p, \text{voren}(p, R, P))\}$
**return** $R$

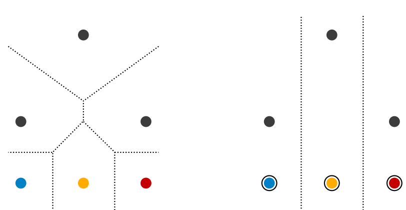
Unfortunately, to the best of our knowledge, no upper-bounds for the selection of FCNN where known before our work. The remaining of this section presents the first results along this line, showing that the selection size of this algorithm cannot be upper-bounded in terms of either $\kappa$ or $k$, as stated in Theorems 4.4 and 4.5 respectively.

**Theorem 4.4.** *For any $0 < \xi < 1$, there exists a training set $P \subset \mathbb{R}^d$ in Euclidean space, with constant number of classes, for which* FCNN *selects $\Omega(k + 1/\xi)$ points.*

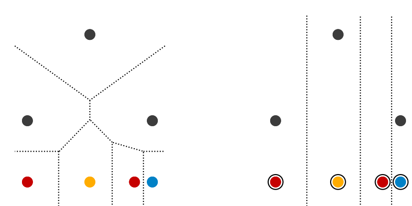
*Proof.* Consider the arrangement in Figure 4.2b (left), consisting of points of four classes. The centroids of the blue, yellow, and red classes are the only points labeled as such. By placing a sufficient number of black points far at the top of this arrangement, we avoid their centroid to be any of the three black points in the arrangement. Beginning with the centroids, the first iteration of FCNN would have added the points outlined in Figure 4.2b (right). Now each of these points have one black point inside their Voronoi cells, and therefore, these black points will be the representatives added in the second iteration. This small example, with $k = 5$, shows how to force FCNN to add all the border points plus two non-border points. Out of these two non-border black points, one is the centroid added in the initial step. The remaining non-border black point, however, was added by the algorithm during the iterative process. This scheme can be extended to larger values of $k$, without increasing the number of classes.



(a) Entire arrangement of points.



(b) Middle arrangement.



(c) Side arrangement.

Figure 4.2: Example of a training set $P \subset \mathbb{R}^2$ for which FCNN selects more than $k$ points.

The previous is the first building block of the entire training set, shown in Figure 4.2a. To this "middle" arrangement, we append "side" arrangements of points, as the one illustrated in Figure 4.2c, which will have similar behavior to the middle arrangement. This particular side arrangement will be appended to the right of the middle one, such that the distance between the red points is greater than the distance from the yellow to the red point. Every time we append a new side arrangement, its blue and red labels are swapped. The arrangements appended to the left side are simply a horizontal flip of the right arrangement. Now, the behavior of FCNN in such a setup is illustrated with the arrows in Figure 4.2a. The extreme point of the previous arrangement adds the yellow point at the center of the current arrangement, which then adds the red point next to the blue point, as is closer than the other red point. Next, this red point adds the blue point, and the yellow point adds the remaining red point. Finally, the Voronoi cells of these points will look as shown in Figure 4.2c (right), and in the next iteration, the tree black points will be added.

After adding side arrangements as needed (same number of the left and right), it is easy to show that the centroids are still the tree points in the middle arrangement and the black point at the top (by adding a sufficient number of black points in the top cluster). This implies that FCNN will be forced to select more than $k$ points. $\square$

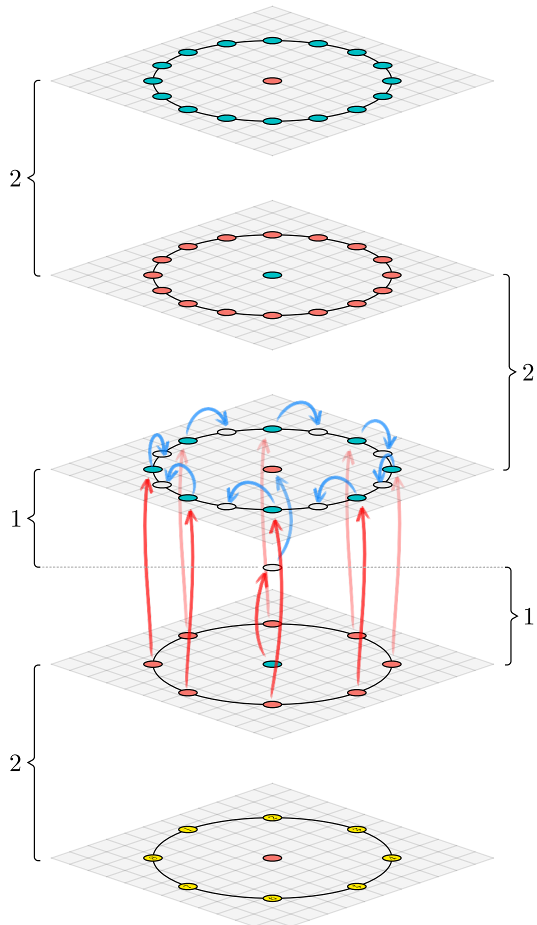
**Theorem 4.5.** *For any $0 < \xi < 1$, there exists a training set $P \subset \mathbb{R}^d$ in Euclidean space, with constant number of classes, for which* FCNN *selects $\Omega(\kappa/\xi)$ points.*

*Proof.* Without loss of generality, let $\xi = 1/2^t$ for some value $t > 3$, we construct a training set $P \subset \mathbb{R}^3$ with constant number of classes, and number of nearest-enemy points $\kappa$ equal to $\mathcal{O}(1/\xi)$, for which FCNN is forced to select $\mathcal{O}(1/\xi^2)$ points. The key downside of the algorithm occurs when points are added to the subset in the same iteration. In general, during any given iteration, the representatives of two neighboring points in FCNN can be arbitrarily close to each other. This flaw can be exploited to force the algorithm to add $\mathcal{O}(1/\xi)$ such points.

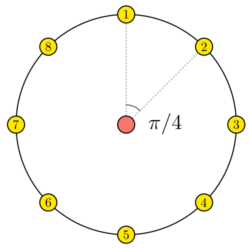
Intuitively, our constructed training set $P$ consists of several layers of points arranged parallel to the $xy$-plane, and stacked on top of each other around the $z$-axis (see Figure 4.3). Each layer is a disk-like arrangement, formed by a center point and points at distance 1 from this center. Thus, define the backbone points of $P$ as the center points $c_i = 2i\vec{v}_z$ for $i \geq 0$. We now describe the different arrangements of points as follows (see Figure 4.3):

$$\begin{aligned}
\mathcal{B} &= c_0 \cup \{y_j = c_0 + \vec{v}_x R_z(j\pi/4) \mid j \in [0, \ldots, 8)\} \\
\mathcal{M}_i &= \{c_{2i}, c_{2i+1}, m_i = (c_{2i} + c_{2i+1})/2\} \\
&\quad \cup \left\{r_{ij} = c_{2i} + \vec{v}_x R_z(j\pi/2^{1+i}) \mid j \in [0, \ldots, 2^{2+i})\right\} \\
&\quad \cup \left\{b_{ij} = c_{2i+1} + \vec{v}_x R_z(j\pi/2^{1+i}) \mid j \in [0, \ldots, 2^{2+i})\right\} \\
&\quad \cup \left\{w_{ij} = c_{2i+1} + \vec{v}_x R_z((2j+1)\pi/2^{2+i} - \xi^2) \mid j \in [0, \ldots, 2^{2+i})\right\} \\
\mathcal{R}_i &= \{c_{2i}, c_{2i+1}\} \\
&\quad \cup \{r_{ij} = c_{2i} + \vec{v}_x R_z(j2\pi/\xi) \mid j \in [0, \ldots, \xi)\} \\
&\quad \cup \{b_{ij} = c_{2i+1} + \vec{v}_x R_z(j2\pi/\xi) \mid j \in [0, \ldots, \xi)\}
\end{aligned}$$

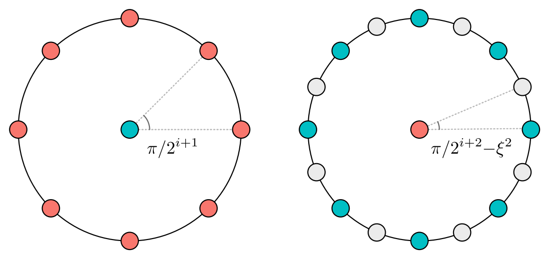
These points belong to one of 11 classes named $\{1, \ldots, 8, \text{red}, \text{blue}, \text{white}\}$. Then define the labeling function $l$ as follows: $l(c_i)$ is red when $i$ is even and blue when $i$ is odd, $l(m_i)$ is white, $l(y_j)$ is the $j$-th class, $l(r_{ij})$ is red, $l(b_{ij})$ is blue, and $l(w_{ij})$ is white.



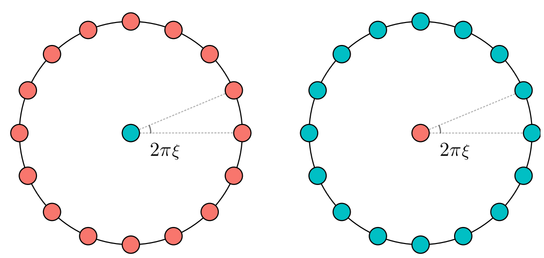

(a) Entire arrangement of points, by stacking the different arrangements along the $z$-axis. The arrows illustrate the selection process by FCNN on a multiplicative arrangement $\mathcal{M}_i$.




(b) Base arrangement $\mathcal{B}$. Each point in the circumference belongs to a unique class (here colored in yellow and numbered 1 to 8 for clarity).




(c) A multiplier arrangement $\mathcal{M}_i$. This forces FCNN to duplicate the number of representatives around the circumference selected on an iteration.




(d) A repetitive arrangement $\mathcal{R}_i$. This maintains the number of representatives selected by FCNN on each iteration of the algorithm.

Figure 4.3: Example of a training set $P \subset \mathbb{R}^3$ for which FCNN selects $\Omega(\kappa/\xi)$ points.

Base arrangement $\mathcal{B}$: Consists of one single layer of points, with one red center point $c_0$ and 8 points $y_j$ in the circumference of the unit disk (parallel to the $xy$-plane), each labeled with a unique class $j$ (see Figure 4.3b). The goal of this arrangement is that each of these points is the centroid of its corresponding class. The centroids of the blue and white classes can be fixed to be far enough, so we won't consider them for now. Hence, the first iteration of FCNN will add all the points of $\mathcal{B}$. In the next iteration, each of these points will select a representative in the arrangement above. Clearly, the size of $\mathcal{B}$ is 9, and it contributes with 8 nearest-enemy points in total.

Multiplier arrangement $\mathcal{M}_i$: Our final goal is to have $\mathcal{O}(1/\xi)$ arbitrarily close points selecting representatives on a single iteration; currently, we only have 9 (the base arrangement). While this could be simply achieved with $\mathcal{O}(1/\xi)$ points in $\mathcal{B}$ each with a unique class, we want to use a constant number of classes. Instead, we use each multiplier arrangement to double the number of representatives selected. $\mathcal{M}_i$ consists of (1) a layer with a blue center $c_{2i}$ and $2^{2+i}$ red points $r_{ij}$ around the unit disk's circumference, (2) a layer with a red center $c_{2i+1}$ and $2^{3+i}$ blue $b_{ij}$ and white $w_{ij}$ points around the unit disk's circumference, and (3) a middle white center point $m_i$ between the red and blue center points (see Figure 4.3c). Suppose at iteration $3i-1$ all the points $r_{ij}$ and $c_{2i}$ of the first layer are added as representatives of the previous arrangement, which is given for $\mathcal{M}_1$ from the selection of $\mathcal{B}$. Then, during iteration $3i$ each $r_{ij}$ adds the point $b_{ij}$ right above, while $c_{2i}$ adds point $m_i$ (see the red arrows in Figure 4.3a). Finally, during iteration $3i+1$, $m_i$ adds $c_{2i+1}$, and each $b_{ij}$ adds point $w_{ij}$ as its the closest point inside the voronoi cell of $b_{ij}$ (see the blue arrows in Figure 4.3a). Now, with all the points of this layer added, each continues to select points in the following arrangement (either $\mathcal{M}_{i+1}$ or $\mathcal{R}_{i+1}$). The size of each $\mathcal{M}_i$ is $3(1+2^{2+i}) = \mathcal{O}(2^{3+i})$, and contributes with $3+2(2^{2+i}) = \mathcal{O}(2^{3+i})$ to the total number of nearest-enemy points. In order to select $1/\xi = 2^t$ points in a single iteration, we need to stack $\mathcal{M}_i$'s for $i \in [1, \ldots, t-3]$.

Repetitive arrangement $\mathcal{R}_i$: Once the algorithm reaches the last multiplier layer $\mathcal{M}_{t-3}$, it will select $1/\xi$ points during the following iteration. The repetitive arrangement allows us to continue adding these many points on every iteration, while only increasing the number of nearest-enemy points by a constant. This arrangement consists of (1) a first layer with a blue center $c_{2i}$ surrounded by $1/\xi$ red points $r_{ij}$ around the unit disk circumference, and (2) a second layer with red center $c_{2i+1}$ and blue points $b_{ij}$ in the circumference (see Figure 4.3d). Once the first layer is added all in a single iteration, during the following iteration $c_{2i}$ adds $c_{2i+1}$, and each $r_{ij}$ adds $b_{ij}$. The size of each $\mathcal{R}_i$ is $2(1+1/\xi) = \mathcal{O}(1/\xi)$, and it contributes with 4 points to the total number of nearest-enemy points. Now, we stack $\mathcal{O}(1/\xi)$ such arrangements $\mathcal{R}_i$ for $i \in [t-2, \ldots, 1/\xi]$, such that we obtain the desired ratio between selected points and number of nearest-enemy points of the training set.

The training set is then defined as $P = \mathcal{B} \bigcup_{i=1}^{t-3} \mathcal{M}_i \bigcup_{i=t-2}^{1/\xi} \mathcal{R}_i \cup \mathcal{F}$, where $\mathcal{F}$ is a set of points designed to fix the centroids of $P$. These extra points are located far enough from the remaining points of $P$, and are carefully placed such that the centroids of $P$ are all the points of $\mathcal{B}$, plus a blue and white point from $\mathcal{F}$. Additionally, all the points of $\mathcal{F}$ should be closer to it's corresponding class centroid than to any enemy centroid, and they should increase the number of nearest-enemy points by a constant. This can be done with $\mathcal{O}(n)$ extra points.

All together, by adding up the corresponding terms, the ratio between the size of FCNN and $\kappa$ (the number of nearest-enemy points of $P$) is $\mathcal{O}(1/\xi)$. Therefore, there exists a training set in 3-dimensional Euclidean space for which FCNN selects $\mathcal{O}(\kappa/\xi)$ for any $\xi < 1/8$. □

These results show that adding points in batch on every iteration of FCNN prevents the algorithm to have any guarantee on the size of its selected subset, in terms of either $k$ or $\kappa$. However, this design choice is not key for any of the features of the algorithm, and can therefore be avoided.

### 4.3.3 SFCNN

The SFCNN algorithm (or *Single* FCNN) is a modified version of FCNN such that only *one* single representative is added to the subset $R$ on each iteration of the algorithm. Surprisingly, such a simple change in the selection process allows us to successfully analyze the size of SFCNN in terms of $\kappa$, and even prove that it computes a tight approximation of the minimum cardinality consistent subset of $P$ on general metrics. These results can be seen as corollaries of the more general Theorems 5.16 and 5.17, which are presented later in Chapter 5.

**Algorithm 5:** SFCNN

**Input:** Initial training set $P$
**Output:** Consistent subset $R \subseteq P$

$R \leftarrow \phi$
$S \leftarrow \text{centroids}(P)$
**while** $S \neq \phi$ **do**
  $R \leftarrow R \cup \{\text{Choose one point of } S\}$
  $S \leftarrow \phi$
  **foreach** $p \in R$ **do**
    $S \leftarrow S \cup \{\text{rep}(p, \text{voren}(p, R, P))\}$
**return** $R$

Following similar arguments to the ones described for the upper-bound proved on CNN (see Theorem 4.3), the following theorem proofs a comparable upper-bound for SFCNN in terms of $\kappa$ and $\gamma$.

**Theorem 4.6.** *The* SFCNN *algorithm selects* $\mathcal{O}(\kappa \log \frac{1}{\gamma})$ *points.*

*Proof.* Similarly to the upper-bound proof of CNN, this result follows by a charging argument on each nearest-enemy point in the training set. Consider one such point $p \in \{\text{ne}(r) \mid r \in P\}$ and a value $\sigma \in [\gamma, 1]$. We define $R_{p,\sigma}$ to be the subset of points selected by SFCNN whose nearest-enemy is $p$, and whose distance to $p$ is between $\sigma$ and $2\sigma$. That is, $R_{p,\sigma} = \{r \in R \mid \text{ne}(r) = p \wedge \mathsf{d}(r, p) \in [\sigma, 2\sigma)\}$. Clearly, these subsets define a partitioning of $R$ when considering all nearest-enemy points of $P$, and values of $\sigma = \gamma\, 2^i$ for $i = \{0, 1, 2, \ldots, \lceil \log \frac{1}{\gamma} \rceil\}$.

Consider any two points $a, b \in R_{p,\sigma}$ in these subsets. Assume *w.l.o.g.* that point $a$ was selected by the algorithm before point $b$ (*i.e.*, in a prior iteration). We show that $\mathsf{d}(a, b) \geq \sigma$. By contradiction, assume that $\mathsf{d}(a, b) < \sigma$, which immediately implies that $a$ and $b$ belong to the same class. Moreover, recalling that $b$'s nearest-enemy in $P$ is $p$, at distance $\mathsf{d}(b, p) \geq \sigma$, this implies that $b$ is closer to $a$ than to any enemy in $R$. Therefore, by the definition of the *voren* function, $b$ could never be selected by SFCNN, which is a contradiction.

This proves that $\mathsf{d}(a,b) \geq \sigma$. Now, just as with CNN, using a simple packing argument based on $d$-dimensional Euclidean balls, we have that $|R'_{p,\sigma}| \leq 5^d$. Altogether, by counting over all the $R_{p,\sigma}$ sets for every nearest-enemy in the training set and values of $\sigma$, the size of $R$ is upper-bounded by $|R| \leq \kappa \lceil \log 1/\gamma \rceil 5^{d+1}$. Assuming $d$ to be constant, this completes the proof. □

## 4.4 Selective Subsets

Another common criterion used for nearest-neighbor condensation is known as *selectiveness* [34]. A subset $R \subseteq P$ is said to be *selective* if and only if for every point $p \in P$ its nearest-neighbor in $R$ is closer to $p$ than its nearest-enemy in $P$. Clearly selectiveness implies consistency, as the nearest-enemy distance in $R$ of any point of $P$ is at least its nearest-enemy distance in $P$.

Similarly to the previous section, this section studies several algorithms that compute selective subsets of $P$, along with a formal worst-case analysis on the sizes of their selected subsets. Namely, these are the NET, MSS, RSS, and VSS algorithms.

### 4.4.1 NET

The NET algorithm [36] was proposed as an approximation algorithm for the problem of finding minimum cardinality consistent subsets. Their paper also presents almost matching hardness lower-bounds for this problem. While this algorithm is proposed for computing consistent subsets of $P$, it can easily be shown that its selection is actually selective. Hence, the NET algorithm is described and analyzed in this section, and not in section 4.3.

The algorithm is fairly simple, and works by computing a $\gamma$-net of $P$ where $\gamma$ is the minimum nearest-enemy distance in $P$. Evidently, this subset must be selective. Moreover, the authors of the paper prove that this algorithm computes a tight approximation of the minimum cardinality consistent subsets, being the first algorithm to show such guarantees. However, in practice, $\gamma$ tends to be small, which results in subsets of much higher cardinality than needed. To overcome this issue, the authors proposed a post-processing pruning technique to further reduce the selected subset, which is formally described in Algorithm 6. But even with this extra pruning, NET is often outperformed on typical training sets (*w.r.t.* both runtime and selection size) by the more practical heuristics studied in this chapter.

**Theorem 4.7.** *The* NET+PRUNE *algorithm selects* $\mathcal{O}(\kappa \log \frac{1}{\gamma})$ *points.*

*Proof.* Proving this result follows basically the same arguments described on the proofs of Theorems 4.3 and 4.6. Similarly, we must define the sets $R_{p,\sigma}$ as the points selected by the NET+PRUNE algorithm whose nearest-enemy is some point $p \in P$, and its nearest-enemy distance is between $[\sigma, 2\sigma)$. The main difference lies on the lower-bound on the distance between any two points $a, b \in R_{p,\sigma}$ in such subsets. In this case, we can only guarantee that $\mathsf{d}(a,b) \geq \sigma/2 - \gamma$. Even though this slightly complicates the packing argument, we can still argue that there exists a constant

**Algorithm 6:** NET+PRUNE

**Input:** Initial training set $P$
**Output:** Selective subset $R \subseteq P$
$R \leftarrow$ Compute a $\gamma$-net of $P$
**foreach** $i \in \{1, 0, \dots, \lfloor \log \gamma \rfloor\}$ **do**
  **foreach** $p \in R$ *with* $\mathsf{d}_{\text{ne}}(p, R) \geq 2^{i+1}$ **do**
    $R \leftarrow R \setminus \{q \in R \mid q \neq p \wedge \mathsf{d}(p, q) < 2^i - \gamma\}$
**return** $R$

$c$ such that $|R_{p,\sigma}| \leq c^d$. Therefore, this implies that NET+PRUNE selects a subset with at most $\kappa \lceil \log \frac{1}{\gamma} \rceil\, c^d$ points, which completes the proof. □

### 4.4.2 MSS

The MSS algorithm (or *Modified Selective Subset*) has been considered a state-of-the-art algorithm for computing selective subsets [17], due to its good performance in practice and its $\mathcal{O}(n^2)$ worst-case runtime.

The selection process of the algorithm can be simply described as follows: for every $p \in P$, MSS selects the point with smallest nearest-enemy distance contained inside the nearest-enemy ball of $p$. Clearly, this approach computes a selective subset of $P$, which by definition, is order-independent. Unfortunately, the selection criteria of MSS can be too strict, requiring one particular point to be added for each point $p \in P$. Note that any point inside the nearest-enemy ball of $p$ suffices for achieving selectiveness. In practice, this can lead to much larger subsets than needed. This intuition is formalized in the following theorem, where we show how MSS can select a subset of unbounded size as a function of either $\kappa$ or $k$.

**Theorem 4.8.** *There exists a training set $P \subset \mathbb{R}^d$ in Euclidean space, with constant number of classes, nearest-enemy points, and border points, such that* MSS *selects $\Omega(1/\xi)$ points, for any $0 < \xi < 1$.*

*Proof.* Recall that for each point in $P$, the MSS algorithm selects the point inside its nearest-enemy ball with smallest nearest-enemy distance. Given a parameter $0 < \xi < 1$, we construct a training set in 1-dimensional Euclidean space, as illustrated in Figure 4.4a. Create two points $r_1$ and $r_2$, and assign them to the class of *red* points. *w.l.o.g.* the distance between these two points is 1. Let $\vec{u}$ be the unit vector from $r_1$ to $r_2$, create additional points $b_i = r_1 + \frac{i\xi}{4}\vec{u}$ for $i = \{1, 2, \dots, 3/\xi\}$. Assign all $b_i$ points to the class of *blue* points. The set of all these points constitute the training set $P$. It is easy to prove that $P$ has only four nearest-enemy points and four border points, corresponding to $r_1, r_2, b_1$ and $b_{3/\xi}$.

Let's discuss which points are added by MSS for each point in $P$ (see Figure 4.4b). For points $r_1$ and $r_2$, the only points inside their nearest-enemy balls are themselves, so both $r_1$ and $r_2$ belong to the subset selected by MSS. For points $b_i$ with $i \leq 2/\xi$, the point with smallest nearest-enemy distance contained inside

**Algorithm 7:** MSS

**Input:** Initial training set $P$

**Output:** Selective subset $R \subseteq P$

Let $\{p_i\}_{i=1}^{n}$ be the points of $P$ sorted increasingly *w.r.t.* their nearest-enemy distance $\mathsf{d}_{\mathrm{ne}}(p_i)$

$R \leftarrow \phi$

$S \leftarrow P$

**foreach** $p_i \in P$, *where* $i = 1 \ldots n$ **do**

  add $\leftarrow$ *false*

  **foreach** $p_j \in P$, *where* $j = i \ldots n$ **do**

    **if** $p_j \in S \ \wedge \ \mathsf{d}(p_j, p_i) < \mathsf{d}_{\mathrm{ne}}(p_j)$ **then**

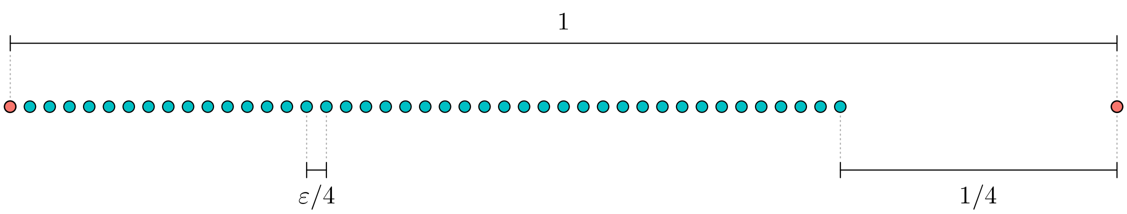
      $S \leftarrow S \setminus \{p_j\}$

      add $\leftarrow$ *true*

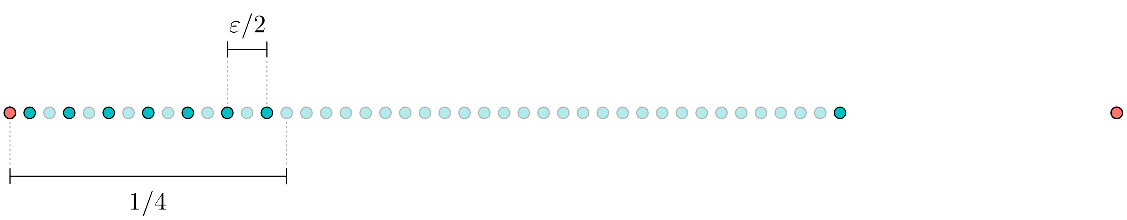
  **if** add **then**

    $R \leftarrow R \cup \{p_i\}$

**return** $R$




(a) Initial training set of collinear points, where both the number of nearest-enemy points and the number of border points equal to 4. That is, $\kappa = k = 4$.




(b) Subset of points computed by MSS from the original training set (fully colored points belong to the subset, while faded points do not). The size of the subset is $\Omega(1/\xi)$.

Figure 4.4: Unbounded example for MSS with respect to $\kappa$ and $k$.

their nearest-enemy ball is $b_1$, which is also added to the subset. Now, consider the points $b_i$ with $2/\xi < i < 5/2\xi$. Let $j = i - 2/\xi$, it is easy to prove that the point with smallest nearest-enemy distance inside the nearest-enemy ball of $b_i$ is $b_{2j+1}$ (see Figure 4.4b). Therefore, this implies that the number of points selected by MSS equals $5/2\xi - 2/\xi = 1/2\xi = \Omega(1/\xi)$. □

### 4.4.3 RSS

We propose the RSS algorithm (or *Relaxed Selective Subset*) with the idea of relaxing the selection process of MSS, while still computing a selective subset. For any given point $p \in P$ in the training set, while MSS requires to add the point with smallest nearest-enemy distance inside the nearest-enemy ball of $p$, in RSS any point inside the nearest-enemy ball $p$ suffices. The idea is rather simple (see Algorithm 8). Points of $P$ are examined in increasing order with respect to their nearest-enemy distance, and we add any point whose nearest-enemy ball contains no point previously added by the algorithm. This tends to select points close to the decision boundaries of $P$ (see Figure 1.1g), as points far from the boundary are examined later in the selection process, and are more likely to already contain points inside their nearest-enemy ball.

---

**Algorithm 8:** RSS

---

**Input:** Initial training set $P$

**Output:** Selective subset $R \subseteq P$

$R \leftarrow \phi$

Let $\{p_i\}_{i=1}^{n}$ be the points of $P$ sorted increasingly *w.r.t.* their nearest-enemy distance $\mathsf{d}_{\mathrm{ne}}(p_i)$

**foreach** $p_i \in P$, *where* $i = 1 \ldots n$ **do**

  **if** $\mathsf{d}_{\mathrm{nn}}(p_i, R) \geq \mathsf{d}_{\mathrm{ne}}(p_i)$ **then**

    $R \leftarrow R \cup \{p_i\}$

**return** $R$

---

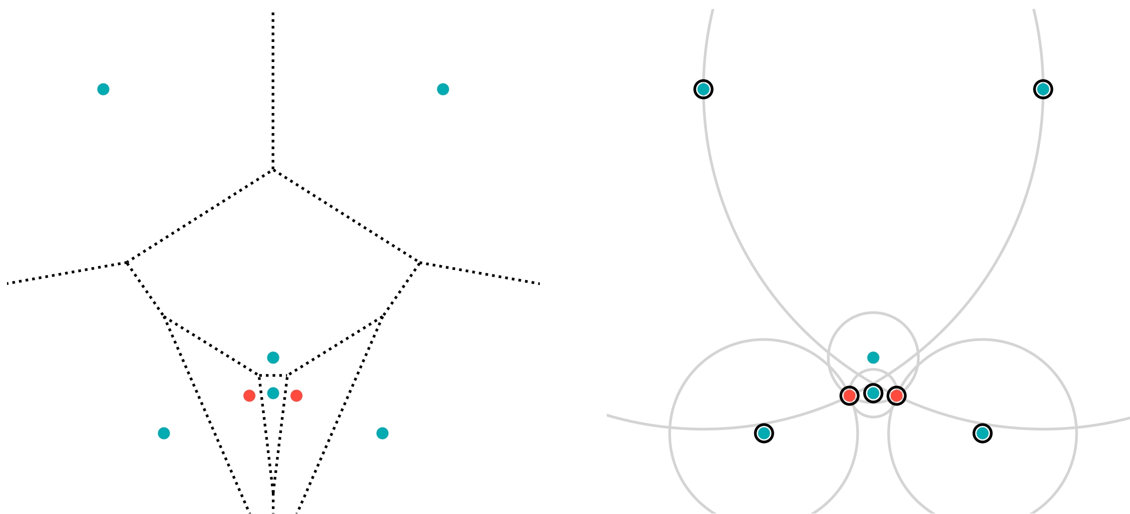
Just like MSS, RSS is order-independent and computes a selective subset of $P$ in $\mathcal{O}(n^2)$ worst-case time. Its selectiveness is evident, as every point in $P$ is either added to the subset, or has a point in the subset inside its nearest-enemy ball. Its order-independence follows from the initial sorting step of the algorithm. Now, the time complexity of RSS can be analyzed as follows. The initial step requires $\mathcal{O}(n^2)$ time for computing the nearest-enemy distances of each point in $P$, plus additional $\mathcal{O}(n \log n)$ time for sorting the points according to such distances. The main loop iterates through each point in $P$, and searches their nearest-neighbor in the current subset, incurring into additional $\mathcal{O}(n^2)$ time using a simple linear search. All together, the worst-case time complexity of the algorithm is quadratic.

**Theorem 4.9.** *The* RSS *algorithm selects* $\mathcal{O}(\kappa)$ *points.*

*Proof.* The proof follows by a charging argument on each nearest-enemy point of $P$. Consider a nearest-enemy point $p \in P$, and let $R_p$ be the set of points selected by RSS such that $p$ is their nearest-enemy. Let $a, b \in R_p$ be two such points, and *w.l.o.g.* say that $\mathsf{d}_{\mathrm{ne}}(a) \leq \mathsf{d}_{\mathrm{ne}}(b)$. By construction of the algorithm, we also know that $\mathsf{d}(a,b) \geq \mathsf{d}_{\mathrm{ne}}(b)$. Now, consider the triangle $\triangle pab$. Clearly, $\overline{pa}$ is the largest side of the triangle, making the angle $\angle apb \geq \pi/3$. This means that the angle between any two points in $R_p$ with respect to $p$ is at least $\pi/3$.

By a standard packing argument, this implies that $|R_p| = \mathcal{O}((3/\pi)^{d-1})$. Finally, we obtain that the number of points selected by RSS is $\sum_p |R_p| = \kappa\, \mathcal{O}((3/\pi)^{d-1})$. $\square$

Additionally, we prove that the size of the subset selected by RSS can be bounded as a function of $\kappa$ and the dimensionality of $P$. Moreover, we show that RSS computes an approximation of both the consistent and selective subsets of minimum cardinality. These results come as corollaries of Theorems 5.10 and 5.11, which will be described in later sections. However, different parameters from $\kappa$ can be used to bound the selection size of condensation algorithms: consider $k$, the number of border points in the training set $P$. From the example illustrated in Figure 4.5, we know that RSS can select more points than $k$ (see Figure 4.5b). Repeating such arrangement forces RSS to select $\Omega(k + 1/\xi)$ points.



(a) Basic point arrangement in $\mathbb{R}^2$, along with the Voronoi diagram induced by such points.

(b) RSS selection outlined, along with the nearest-enemy balls of each point in the arrangement.

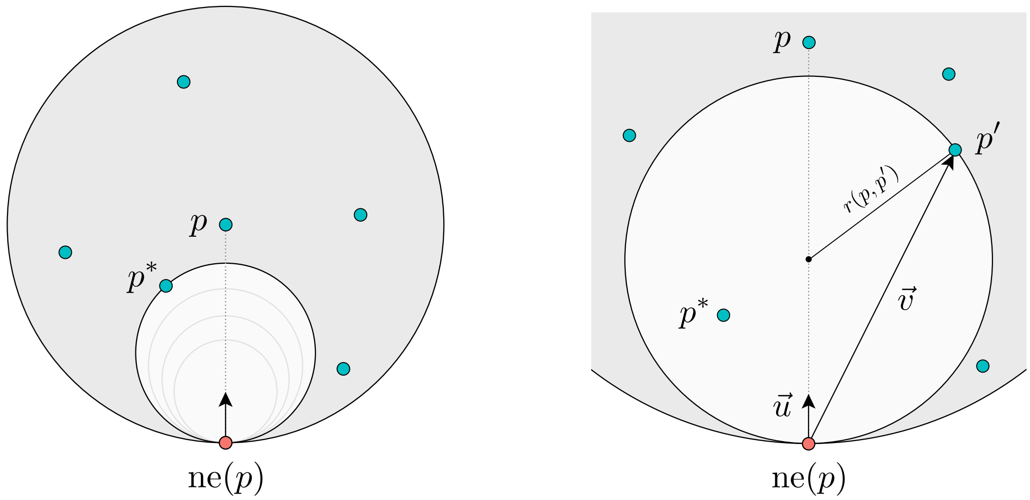
Figure 4.5: Example where RSS selects $k + 1$ points.

### 4.4.4 VSS

We now propose the VSS algorithm (or *Voronoi Selective Subset*). This new algorithm comes as a modification of RSS, based on an observation from the proof of the following lemma. As stated in Chapter 2, we are able to prove that every nearest-enemy is also a border point of $P$, formalized here.

**Lemma 4.10.** *Any nearest-enemy point of a point in $P$ is also a border point of $P$.*

*Proof.* Take any point $p \in P$. Consider the *empty* ball of maximum radius, tangent to point $\mathrm{ne}(p)$, and with center in the line segment between $p$ and $\mathrm{ne}(p)$. Being maximal, this ball is tangent to another point $p^* \in P$ (see Figure 4.6a). Clearly, $p^*$ is inside the nearest-enemy ball of $p$, which implies that $p$ and $p^*$ belong to the same class, making $p^*$ and $\mathrm{ne}(p)$ enemies. By the empty ball property, this means that both $p^*$ and $\mathrm{ne}(p)$ are border points of $P$. □




(a) The largest empty ball tangent to $\mathrm{ne}(p)$ and center in $\overline{p\,\mathrm{ne}(p)}$, is also tangent to some point $p^*$, making $p^*$ and $\mathrm{ne}(p)$ border points.

(b) How to compute the radius of a ball with center in the line segment between $p$ and $\mathrm{ne}(p)$, and tangent to both $\mathrm{ne}(p)$ and $p'$.

Figure 4.6: Relation between nearest-enemy points and border points.

Therefore, from Lemma 4.10 we know that in $d$-dimensional Euclidean space, the number of nearest-enemy points of $P$ is at most the number of border points of $P$. That is, $\kappa \leq k$. Moreover, this result can be easily extended to $\ell_p$ metric spaces with $p \geq 2$. While this result implies an easy extension of the upper-bounds found for RSS, CNN, SFCNN, and NET+Prune, now in terms of $k$ instead of $\kappa$, it is unclear if the other factors in those upper-bounds can be improved.

Coming back to VSS, this proof opens an alternative idea for condensation. In order to prove Lemma 4.10, we show that there exist at least one border point inside the nearest-enemy ball of any point $p \in P$. Therefore, by selecting such border points, we can guarantee that the resulting subset is selective and its size is $\leq k$.

We call this algorithm VSS (see Algorithm 9 for a formal description). Essentially, we show this algorithm computes a selective subset of $P$ of size at most $k$. By construction, for any point in $p \in P$ the algorithm selects one border point inside the nearest-enemy ball of $p$, which implies that the resulting subset is selective, and contains no more than $k$ points.

**Theorem 4.11.** *The* VSS *algorithm selects at most $k$ points.*

Finally, we describe an efficient implementation of VSS, showing this algorithm can be implemented to run in quadratic time. Recall that for every point $p \in P$, the algorithm selects a border point inside its nearest-enemy ball. *w.l.o.g.* implement VSS to compute the point $p^*$ that minimizes the radius of an empty ball tangent

**Algorithm 9:** VSS

**Input:** Initial training set $P$
**Output:** Selective subset $R \subseteq P$

$R \leftarrow \phi$
Let $\{p_i\}_{i=1}^n$ be the points of $P$ sorted increasingly *w.r.t.* their nearest-enemy distance $\mathsf{d}_{\text{ne}}(p_i)$
**foreach** $p_i \in P$, *where* $i = 1 \ldots n$ **do**
  **if** $\mathsf{d}_{\text{nn}}(p_i, R) \geq \mathsf{d}_{\text{ne}}(p_i)$ **then**
    Find a border point that lies inside the nearest-enemy ball of $p_i$ and add it to $R$
**return** $R$

to both $\text{ne}(p)$ and $p^*$, and center in the line segment between $p$ and $\text{ne}(p)$. For any given point $p'$ inside the nearest-enemy ball of $p$, denote $r(p, p')$ to be the radius of the ball tangent to $p'$ and $\text{ne}(p)$ and center in the line segment between $p$ and $\text{ne}(p)$. As illustrated in Figure 4.6b, let vectors $\vec{u} = \frac{p - \text{ne}(p)}{\|p - \text{ne}(p)\|}$ and $\vec{v} = p' - \text{ne}(p)$, the radius of this ball can be derived from the formula $r(p, p') = \|\vec{v} + r(p, p')\vec{u}\|$ as $r(p, p') = \vec{v} \cdot \vec{v} / 2\vec{u} \cdot \vec{v}$. As $p^*$ is defined as the point that minimizes such radius, a simple scan over the points of $P$ suffices to identify the corresponding $p^*$ for any point $p \in P$. Therefore, this implies that VSS can be computed in $\mathcal{O}(n^2)$ worst-case time.

## 4.5 Experimental Comparison

Historically, the importance of some condensation algorithms rely on their performance in practice, despite the lack of theoretical guarantees. Therefore, a natural question is how the algorithms proposed in this chapter compare to existing ones when evaluated in real-world training sets.

Thus, to get a clearer impression of the relevance of these results in practice, we performed experimental trials on several training sets, both synthetically generated and widely used benchmarks. First, we consider 21 training sets from the UCI *Machine Learning Repository*[1] which are commonly used in the literature to evaluate condensation algorithms [18]. These consist of a number of points ranging from 150 to 58000, in $d$-dimensional Euclidean space with $d$ between 2 and 64, and 2 to 26 classes. We also generated some synthetic training sets, containing $10^5$ uniformly distributed points, in 2 to 3 dimensions, and 3 classes. All training sets used in these experimental trials are summarized in Table 4.2. The implementation of the algorithms, training sets used, and raw results, are publicly available[2].

We test seven different condensation algorithms, namely CNN, FCNN, SFCNN, MSS, RSS, VSS, and NET. To compare their results, we consider their runtime and

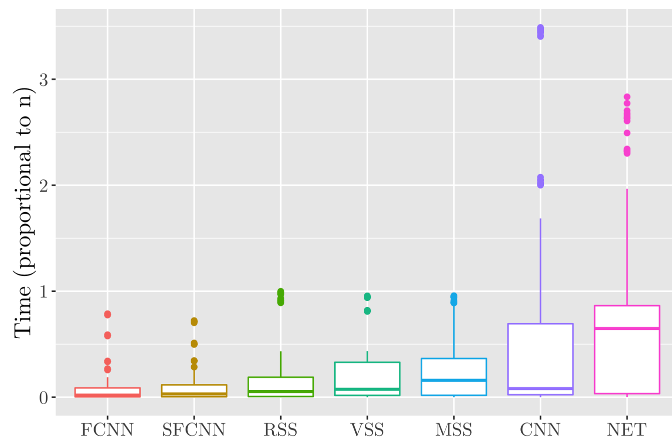
[1] https://archive.ics.uci.edu/ml/index.php
[2] https://github.com/afloresv/nnc/

| Training set | $n$ | $d$ | $c$ | $\kappa$ (%) |
|---|---|---|---|---|
| banana | 5300 | 2 | 2 | 811 (15.30%) |
| cleveland | 297 | 13 | 5 | 125 (42.09%) |
| glass | 214 | 9 | 6 | 87 (40.65%) |
| iris | 150 | 4 | 3 | 20 (13.33%) |
| iris2d | 150 | 2 | 3 | 13 (8.67%) |
| letter | 20000 | 16 | 26 | 6100 (30.50%) |
| magic | 19020 | 10 | 2 | 5191 (27.29%) |
| monk | 432 | 6 | 2 | 300 (69.44%) |
| optdigits | 5620 | 64 | 10 | 1245 (22.15%) |
| pageblocks | 5472 | 10 | 5 | 429 (7.84%) |
| penbased | 10992 | 16 | 10 | 1352 (12.30%) |
| pima | 768 | 8 | 2 | 293 (38.15%) |
| ring | 7400 | 20 | 2 | 2369 (32.01%) |
| satimage | 6435 | 36 | 6 | 1167 (18.14%) |
| segmentation | 2100 | 19 | 7 | 398 (18.95%) |
| shuttle | 58000 | 9 | 7 | 920 (1.59%) |
| thyroid | 7200 | 21 | 3 | 779 (10.82%) |
| twonorm | 7400 | 20 | 2 | 1298 (17.54%) |
| wdbc | 569 | 30 | 2 | 123 (21.62%) |
| wine | 178 | 13 | 3 | 37 (20.79%) |
| wisconsin | 683 | 9 | 2 | 35 (5.12%) |
| v-100000-2-3-15 | 100000 | 2 | 3 | 1909 (1.90%) |
| v-100000-2-3-5 | 100000 | 2 | 3 | 788 (0.78%) |
| v-100000-3-3-15 | 100000 | 3 | 3 | 7043 (7.04%) |
| v-100000-3-3-5 | 100000 | 3 | 3 | 3738 (3.73%) |
| v-100000-4-3-15 | 100000 | 4 | 3 | 13027 (13.02%) |
| v-100000-4-3-5 | 100000 | 4 | 3 | 10826 (10.82%) |
| v-100000-5-3-15 | 100000 | 5 | 3 | 22255 (22.25%) |
| v-100000-5-3-5 | 100000 | 5 | 3 | 17705 (17.70%) |

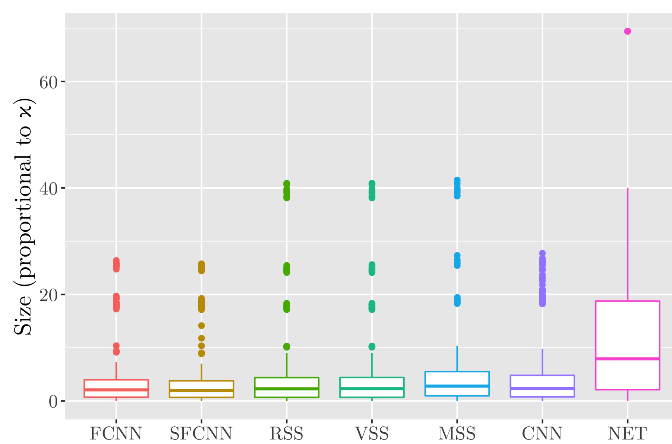
Table 4.2: Training sets used to evaluate the performance of condensation algorithms. Indicates the number of points $n$, dimensions $d$, classes $c$, nearest-enemy points $\kappa$ (also in percentage *w.r.t.* $n$).

the size of the selected subset. Clearly, these values might differ greatly on training sets whose size are too distinct. Therefore, before comparing the raw results, these are normalized. The runtime of an algorithm for a given training set is normalized by dividing it by $n$, the size of the training set. The size of the selected subset is normalized by dividing it by $\kappa$, the number of nearest-enemy points in the training set, which characterizes the complexity of the boundaries between classes.

Figures 4.7a and 4.7b summarize the experimental results. Evidently, the performance of SFCNN is equivalent to the original FCNN algorithm, both in terms of runtime and the size of their selected subsets, showing that the proposed modification does not affect the behavior of the algorithm in real-world training sets. Both FCNN and SFCNN outperform other condensation algorithms in terms of runtime, while their subset size is comparable in all cases, with the exception of the NET algorithm.




(a) Running time.




(b) Size of the selected subsets.

Figure 4.7: Evaluating the studied condensation algorithms: CNN, FCNN, SFCNN, NET, MSS, RSS, and VSS.

# Chapter 5: $\varepsilon$-Coresets

## 5.1 Introduction

There are obvious parallels between the problem of nearest-neighbor condensation and the concept of *coresets* in geometric approximation [57–60]. Intuitively, a *coreset* is small subset of the original data, that well approximates some statistical properties of the original set. Coresets have previously been applied to many problems in machine learning, such as clustering and neural network compression [61–64]. Additionally, coresets have been used towards achieving more efficient classification techniques, as evidenced by the recent results on coresets for the SVM classifier [65].

While the other chapters of this book deal with training sets in $d$-dimensional Euclidean space (*i.e.*, $P \subset \mathbb{R}^d$), this chapter deals with the more generic concept of metric spaces. Therefore, we assume a training set $P$ consisting of $n$ points in a metric space $(\mathcal{X}, \mathsf{d})$, with domain $\mathcal{X}$ and distance function $\mathsf{d} : \mathcal{X}^2 \to \mathbb{R}^+$. Evidently, $\mathsf{d}$ must hold the properties of *identity*, *symmetry*, and *triangle inequality*. The rest remains the same. That is, $P$ is partitioned into a finite set of *classes* by associating each point $p \in P$ with a *label* $l(p)$, indicating the class to which it belongs.

Given an *unlabeled* query point $q \in \mathcal{X}$, the *exact* nearest-neighbor rule predicts $q$'s class using the class of its closest point in $P$ according to metric $\mathsf{d}$. That is, it assigns $q$ to the class $l(\mathrm{nn}(q))$, where $\mathrm{nn}(q) = \arg\min_{p \in P} \mathsf{d}(q, p)$.

So far, in Chapters 3 and 4 we have assumed that $q$'s nearest-neighbor is always computed exactly. However, it is common practice that these queries are instead computed approximately, leveraging known techniques for efficient nearest-neighbor search like Approximate Voronoi Diagrams [19, 20], Locality-Sensitive Hashing [21], and Hierarchical Navigable Small Worlds graphs [22]. In this context, we are given an approximation parameter $\varepsilon \in [0, 1]$ and an unlabeled query point $q \in \mathcal{X}$, and the $\varepsilon$-approximate nearest-neighbor rule assigns $q$ to the class of some point $p$, where $p$ is any point of $P$ such that $\mathsf{d}(q, p) \leq (1 + \varepsilon)\, \mathsf{d}_{\mathrm{nn}}(q)$. Therefore, in both this and the following chapters, we discuss different techniques to achieve boundary-sensitive approaches (*i.e.*, dependent on $\kappa$ and $k$ instead of $n$) for approximate nearest-neighbor classification.

This chapter presents the first approach proposed to compute coresets for nearest-neighbor classification, leveraging its resemblance to the problem of nearest-neighbor condensation. These results have been published in [66].

## Preliminaries

Similarly to Chapter 4, we define notation relevant to understanding the results presented. Given any point $q \in \mathcal{X}$ in the metric space, its nearest-neighbor, denoted $\mathrm{nn}(q)$, is the closest point of $P$ according the the distance function $\mathsf{d}$. The distance from $q$ to its nearest-neighbor is denoted by $\mathsf{d}_{\mathrm{nn}}(q, P)$, or simply $\mathsf{d}_{\mathrm{nn}}(q)$ when $P$ is clear. Given a point $p \in P$ from the training set its nearest-neighbor in $P$ is $p$ itself. Additionally, any point of $P$ whose label differs from $p$'s is called an *enemy* of $p$. The closest such point is called $p$'s *nearest-enemy*, and the distance to this point is called $p$'s *nearest-enemy distance*. These are denoted as $\mathrm{ne}(p)$ and $\mathsf{d}_{\mathrm{ne}}(p, P)$, respectively. Evidently, we can simplify $\mathsf{d}_{\mathrm{ne}}(p, P)$ as $\mathsf{d}_{\mathrm{ne}}(p)$ when it is clear.

Unsurprisingly, the size of a coreset for nearest-neighbor classification should depend on the spatial characteristics of the points of different classes in the training set. For example, it should be much easier to find a small coreset for two spatially well separated clusters than for two classes that have a high degree of overlap. Again, we use the number of nearest-enemy points of $P$, denoted as $\kappa$, to model the intrinsic complexity of the problem of nearest-neighbor classification.

Other intrinsic characteristics of $P$ will also come handy when describing these coresets. Through a suitable uniform scaling, we may assume that the *diameter* of $P$ (that is, the maximum distance between any two points in the training set) is 1. Then, the *spread* of $P$, denoted as $\Delta$, is the ratio between the largest and smallest distances in $P$. Similarly, recall the definition of the *margin* of $P$, denoted $\gamma$, as the smallest nearest-enemy distance in $P$. Clearly, we can see that $1/\gamma \leq \Delta$.

Additionally, a metric space $(\mathcal{X}, \mathsf{d})$ is said to be *doubling* [5] if there exist some bounded value $\lambda$ such that any metric ball of radius $r$ can be covered with at most $\lambda$ metric balls of radius $r/2$. Its *doubling dimension* is the base-2 logarithm of $\lambda$, denoted as $\mathrm{ddim}(\mathcal{X}) = \log \lambda$. Throughout this chapter, we assume that $\mathrm{ddim}(\mathcal{X})$ is a constant, which means that multiplicative factors depending on $\mathrm{ddim}(\mathcal{X})$ may be hidden in our asymptotic notation. Many natural metric spaces of interest are doubling, including $d$-dimensional Euclidean space whose doubling dimension is $\Theta(d)$. An important property of doubling spaces, that will come useful throughout this chapter, is that for any subset $R \subseteq \mathcal{X}$ with spread $\Delta_R$, the size of $R$ is upper-bounded by $|R| \leq \lceil \Delta_R \rceil^{\mathrm{ddim}(\mathcal{X})+1}$.

## Contributions

In this chapter, we introduce the concept of a coreset for classification with the nearest-neighbor rule, which provides approximate guarantees on correct classification for all query points. We demonstrate their existence, analyze their size, and discuss approaches for their efficient computation.

We say that a subset $R \subseteq P$ is an *$\varepsilon$-coreset for the nearest-neighbor rule* on $P$, if and only if for every query point $q \in \mathcal{X}$, the class of its exact nearest-neighbor in $R$ is the same as the class of some $\varepsilon$-approximate nearest-neighbor of $q$ in $P$ (see Section 5.2 for definitions). Recalling the concepts of $\kappa$ and $\gamma$ introduced in the preliminaries, here is our main result:

**Theorem 5.1.** *Given a training set $P$ in a doubling metric space $(\mathcal{X}, \mathsf{d})$, there exist an $\varepsilon$-coreset for the nearest-neighbor rule of size $\mathcal{O}(\kappa \log \frac{1}{\gamma} (1/\varepsilon)^{\mathrm{ddim}(\mathcal{X})+1})$, and this coreset can be computed in subquadratic worst-case time.*

The following summarizes of the principal results presented in the remaining sections, which all together are leveraged to prove the main theorem.

- We extend the criteria used for nearest-neighbor condensation, and identify sufficient conditions to guarantee the correct classification of any query point after condensation. These conditions are the ones that describe our coreset construction.
- We prove that finding minimum-cardinality subsets with this new criteria is NP-hard. Moreover, we prove it is even hard to approximate within practical factors.
- We provide quadratic-time approximation algorithms with provable upper-bounds on the sizes of their selected subsets, and we show that the running time of one such algorithm can be improved to be subquadratic. This subquadratic-time algorithm is the first with such worst-case runtime for the problem of nearest-neighbor condensation.

## 5.2 Coreset Characterization

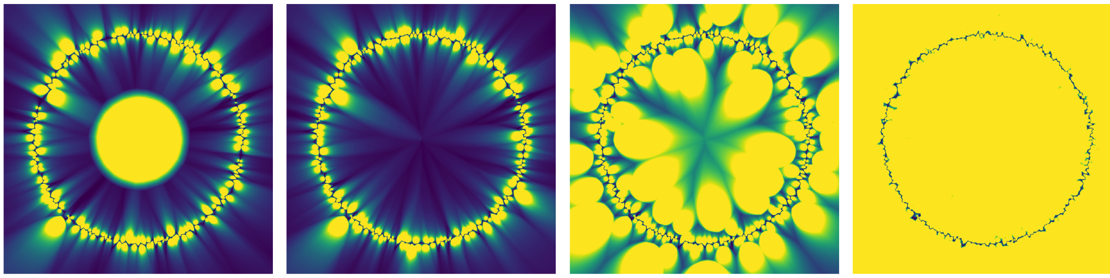
In practice, many applications usually rely its efficiency on computing nearest-neighbors not exactly, but rather approximately. Given an approximation parameter $\varepsilon \geq 0$, an *$\varepsilon$-approximate* nearest-neighbor or $\varepsilon$-ANN query returns any point whose distance from the query point is within a factor of $(1+\varepsilon)$ times the true nearest-neighbor distance.

Intuitively, a query point should be easier to classify if its nearest-neighbor is significantly closer than its nearest-enemy. This intuition can be formalized through the concept of the *chromatic density* [51] of a query point $q \in \mathcal{X}$ with respect to a set $R \subseteq P$, defined as:

$$\delta(q, R) = \frac{\mathsf{d}_{\mathrm{ne}}(q, R)}{\mathsf{d}_{\mathrm{nn}}(q, R)} - 1. \tag{5.1}$$

Clearly, if $\delta(q, R) > \varepsilon$ then $q$ will be correctly classified[1] by an $\varepsilon$-ANN query over $R$, as all possible candidates for the approximate nearest-neighbor belong to the same class as $q$'s true nearest-neighbor. However, as evidenced in Figures 5.1a and 5.1b, one side effect of existing condensation algorithms is a significant reduction in the chromatic density of query points. Consequently, we propose new criteria and algorithms that maintain high chromatic densities after condensation, which are then leveraged to build coresets for the nearest-neighbor rule.

[1] By *correct classification*, we mean that the classification is the same as the classification that results from applying the nearest-neighbor rule exactly on the entire training set $P$.



(a) FCNN (b) RSS (c) 0.1-RSS (d) 0.5-RSS

Figure 5.1: Heatmap of *chromatic density* values of points in $\mathbb{R}^2$ *w.r.t.* the subsets computed by different condensation algorithms: FCNN, RSS, and $\alpha$-RSS (see Figure 1.1). *Yellow* ● corresponds to chromatic density values $\geq 0.5$, while *blue* ● corresponds to 0. Evidently, $\alpha$-RSS helps maintaining high chromatic density values when compared to standard condensation algorithms.

### 5.2.1 Approximation-Sensitive Condensation

The decision boundaries of the nearest-neighbor rule (that is, points $q$ such that $\mathsf{d}_{\mathrm{ne}}(q, P) = \mathsf{d}_{\mathrm{nn}}(q, P)$) are naturally characterized by points that separate clusters of points of different classes. As illustrated in Figures 1.1c-1.1g, condensation algorithms tend to select such points. However, this behavior implies a significant reduction of the chromatic density of query points that are far from such boundaries, as can be seen in Figures 5.1a-5.1b with the selection of the FCNN and RSS algorithms.

A natural way to define an approximate notion of consistency is to ensure that all points in $P$ are correctly classified by ANN queries over the condensed subset $R$. Given a condensation parameter $\alpha \geq 0$, we define a subset $R \subseteq P$ to be:

$$\alpha\text{-consistent} \;\; \text{if } \forall\, p \in P,\;\; \mathsf{d}_{\mathrm{nn}}(p, R) < \mathsf{d}_{\mathrm{ne}}(p, R)/(1+\alpha).$$

$$\alpha\text{-selective} \;\;\;\; \text{if } \forall\, p \in P,\;\; \mathsf{d}_{\mathrm{nn}}(p, R) < \mathsf{d}_{\mathrm{ne}}(p, P)/(1+\alpha).$$

It is easy to see that the standard forms arise as special cases when $\alpha = 0$. These new condensation criteria imply that $\delta(p, R) > \alpha$ for every $p \in P$, meaning that $p$ is correctly classified using an $\alpha$-ANN query on $R$. Note that any $\alpha$-selective subset is also $\alpha$-consistent. Such subsets always exist for any $\alpha \geq 0$ by taking $R = P$. Current condensation algorithms cannot guarantee either $\alpha$-consistency or $\alpha$-selectiveness unless $\alpha$ is equal to zero. Therefore, the central algorithmic challenge is how to efficiently compute such sets whose sizes are significantly smaller than $P$. We propose new algorithms to compute such subsets, which showcase how to maintain high chromatic density values after condensation, as evidenced in Figures 5.1c and 5.1d. This empirical evidence is matched with theoretical guarantees for $\alpha$-consistent and $\alpha$-selective subsets, described in the following section.

### 5.2.2 Guarantees on Classification Accuracy

These newly defined criteria for nearest-neighbor condensation enforce lower-bounds on the chromatic density of any point of $P$ after condensation. However, this doesn't immediately imply having similar lower-bounds for unlabeled query points of $\mathcal{X}$. In this section, we prove useful bounds on the chromatic density of query points, and characterize sufficient conditions to correctly classify some of these query points after condensation.

Intuitively, the chromatic density determines how easy it is to correctly classify a query point $q \in \mathcal{X}$. We show that the "ease" of classification of $q$ after condensation (*i.e.*, $\delta(q, R)$) depends on both the condensation parameter $\alpha$, and the chromatic density of $q$ before condensation (*i.e.*, $\delta(q, P)$). This result is formalized in the following lemma:

**Lemma 5.2.** *Let $q \in \mathcal{X}$ be a query point, and $R$ an $\alpha$-consistent subset of $P$, for $\alpha \geq 0$. Then, $q$'s chromatic density with respect to $R$ is:*

$$\delta(q, R) > \frac{\alpha\, \delta(q, P) - 2}{\delta(q, P) + \alpha + 3}.$$

*Proof.* The proof follows by analyzing $q$'s nearest-enemy distance in $R$. To this end, consider the point $p \in P$ that is $q$'s nearest-neighbor in $P$. There are two possible cases:

Case 1: If $p \in R$, clearly $\mathsf{d}_{\text{nn}}(q, R) = \mathsf{d}_{\text{nn}}(q, P)$. Additionally, it is easy to show that after condensation, $q$'s nearest-enemy distance can only increase: *i.e.*, $\mathsf{d}_{\text{ne}}(q, P) \leq \mathsf{d}_{\text{ne}}(q, R)$. This implies that $\delta(q, R) \geq \delta(q, P)$.

Case 2: If $p \notin R$, we can upper-bound $q$'s nearest-neighbor distance in $R$ as follows:

Since $R$ is an $\alpha$-consistent subset of $P$, we know that there exists a point $r \in R$ such that $\mathsf{d}(p, r) < \mathsf{d}_{\text{ne}}(p, R)/(1+\alpha)$. By the triangle inequality and the definition of nearest-enemy, $\mathsf{d}_{\text{ne}}(p, R) \leq \mathsf{d}(p, \text{ne}(q, R)) \leq \mathsf{d}(q, p) + \mathsf{d}_{\text{ne}}(q, R)$. Additionally, applying the definition of chromatic density on $q$ and knowing that $\mathsf{d}_{\text{ne}}(q, P) \leq \mathsf{d}_{\text{ne}}(q, R)$, we have $\mathsf{d}(q, p) = \mathsf{d}_{\text{nn}}(q, P) \leq \mathsf{d}_{\text{nn}}(q, R) = \mathsf{d}_{\text{ne}}(q, R)/(1 + \delta(q, P))$. Therefore:

$$\begin{aligned}\mathsf{d}_{\text{nn}}(q, R) &\leq \mathsf{d}(q, r) \leq \mathsf{d}(q, p) + \mathsf{d}(p, r) \\ &< \mathsf{d}(q, p) + \frac{\mathsf{d}(q, p) + \mathsf{d}_{\text{ne}}(q, R)}{1 + \alpha} \leq \left( \frac{\delta(q, P) + \alpha + 3}{(1 + \alpha)(1 + \delta(q, P))} \right) \mathsf{d}_{\text{ne}}(q, R).\end{aligned}$$

Finally, from the definition of $\delta(q, R)$, we have:

$$\delta(q, R) = \frac{\mathsf{d}_{\text{ne}}(q, R)}{\mathsf{d}_{\text{nn}}(q, R)} - 1 > \frac{(1 + \alpha)(1 + \delta(q, P))}{\delta(q, P) + \alpha + 3} - 1 = \frac{\alpha\, \delta(q, P) - 2}{\delta(q, P) + \alpha + 3}. \qquad \square$$

The above result can be leveraged to define a coreset, in the sense that an exact result on the coreset corresponds to an approximate result on the original set. As previously defined, we say that a set $R \subseteq P$ is an $\varepsilon$-*coreset for the nearest-neighbor* *rule* on $P$, if and only if for every query point $q \in \mathcal{X}$, the class of $q$'s exact nearest-neighbor in $R$ is the same as the class of any of its $\varepsilon$-approximate nearest-neighbors in $P$.

**Lemma 5.3.** *Any $\varepsilon$-coreset for the nearest-neighbor rule is an $\alpha$-consistent subset, for $\alpha \geq 0$.*

*Proof.* Consider any $\varepsilon$-coreset $C \subseteq P$ for the nearest-neighbor rule on $P$. Since the approximation guarantee holds for any point in $\mathcal{X}$, it holds for any $p \in P \setminus C$. We know $p$'s nearest-neighbor in the original set $P$ is $p$ itself, thus making $\mathsf{d}_{\text{nn}}(p, P)$ zero. This implies that $p$ must be correctly classified by a nearest-neighbor query on $C$, that is, $\mathsf{d}_{\text{nn}}(p, C) < \mathsf{d}_{\text{ne}}(p, C)$, which is the definition of $\alpha$-consistency for any $\alpha \geq 0$. □

**Theorem 5.4.** *Any $2/\varepsilon$-selective subset is an $\varepsilon$-coreset for the nearest-neighbor rule.*

*Proof.* Let $R$ be an $\alpha$-selective subset of $P$, where $\alpha = 2/\varepsilon$. Consider any query point $q \in \mathcal{X}$ in the metric space. It suffices to show that its nearest-neighbor in $R$ is of the same class as any $\varepsilon$-approximate nearest-neighbor in $P$. To this end, consider $q$'s chromatic density with respect to both $P$ and $R$, denoted as $\delta(q, P)$ and $\delta(q, R)$, respectively. We identify two cases:

Case 1 (Correct-Classification guarantee): If $\delta(q, P) \geq \varepsilon$.
Consider the bound derived in Lemma 5.2. Since $\alpha \geq 0$, and by our assumption that $\delta(q, P) \geq \varepsilon > 0$, setting $\alpha = 2/\varepsilon$ implies that $\delta(q, R) > 0$. This means that the nearest-neighbor of $q$ in $R$ belongs to the same class as the nearest-neighbor of $q$ in $P$. Intuitively, this guarantees that $q$ is correctly classified by the nearest-neighbor rule in $R$.

Case 2 ($\varepsilon$-Approximation guarantee): If $\delta(q, P) < \varepsilon$.
Let $p \in P$ be $q$'s nearest-neighbor in $P$, thus $\mathsf{d}(q, p) = \mathsf{d}_{\text{nn}}(q, P)$. Since $R$ is $\alpha$-selective, there exists a point $r \in R$ such that $\mathsf{d}(p, r) = \mathsf{d}_{\text{nn}}(p, R) < \mathsf{d}_{\text{ne}}(p, P)/(1+\alpha)$. Additionally, by the triangle inequality and the definition of nearest-enemies, we have

$$\mathsf{d}_{\text{ne}}(p, P) \leq \mathsf{d}(p, \text{ne}(q, P)) \leq \mathsf{d}(p, q) + \mathsf{d}(q, \text{ne}(q, P)) = \mathsf{d}_{\text{nn}}(q, P) + \mathsf{d}_{\text{ne}}(q, P).$$

From the definition of chromatic density, $\mathsf{d}_{\text{ne}}(q, P) = (1 + \delta(q, P))\, \mathsf{d}_{\text{nn}}(q, P)$. Together, these inequalities imply that $(1+\alpha)\, \mathsf{d}(p, r) \leq (2 + \delta(q, P))\, \mathsf{d}_{\text{nn}}(q, P)$. Therefore:

$$\mathsf{d}_{\text{nn}}(q, R) \leq \mathsf{d}(q, r) \leq \mathsf{d}(q, p) + \mathsf{d}(p, r) \leq \left(1 + \frac{2 + \delta(q, P)}{1 + \alpha}\right) \mathsf{d}_{\text{nn}}(q, P).$$

Now, assuming $\delta(q, P) < \varepsilon$ and setting $\alpha = 2/\varepsilon$, imply that $\mathsf{d}_{\text{nn}}(q, R) < (1+\varepsilon)\, \mathsf{d}_{\text{nn}}(q, P)$. Therefore, the nearest-neighbor of $q$ in $R$ is an $\varepsilon$-approximate nearest-neighbor of $q$ in $P$.

Cases 1 and 2 imply that setting $\alpha$ to $2/\varepsilon$ is sufficient to ensure that the nearest-neighbor rule classifies any query point $q \in \mathcal{X}$ with the class of one of its valid $\varepsilon$-approximate nearest-neighbors in $P$. Therefore, $R$ is an $\varepsilon$-coreset for the nearest-neighbor rule on $P$. □

So far, we have assumed that nearest-neighbor queries over $R$ are computed exactly, as this is the standard notion of coresets. However, it is reasonable to compute nearest-neighbors approximately even for $R$. How should the two approximations be combined to achieve a desired final degree of accuracy? Consider another approximation parameter $\xi$, where $0 \leq \xi < \varepsilon$. We say that a set $R \subseteq P$ is an $(\xi, \varepsilon)$-*coreset* for the approximate nearest-neighbor rule on $P$, if and only if for every query point $q \in \mathcal{X}$, the class of any of $q$'s $\xi$-approximate nearest-neighbor in $R$ is the same as the class of any of its $\varepsilon$-approximate nearest-neighbors in $P$. The following result generalizes Theorem 5.4 to accommodate for $\xi$-ANN queries after condensation.

**Theorem 5.5.** *Any $\alpha$-selective subset is an $(\xi, \varepsilon)$-coreset for the approximate nearest-neighbor rule when $\alpha = \Omega(1/(\varepsilon - \xi))$.*

*Proof.* This follows from similar arguments to the ones described in the proof of Theorem 5.4. Instead, here we set $\alpha = (\varepsilon\xi + 3\xi + 2)/(\varepsilon - \xi)$. Let $R$ be an $\alpha$-selective subset of $P$, and $q \in \mathcal{X}$ any query point in the metric space, consider the same two cases:

Case 1 (Correct-Classification guarantee): If $\delta(q, P) \geq \varepsilon$.

Consider the bound derived in Lemma 5.2. By our assumption that $\delta(q, P) \geq \varepsilon > 0$, and since $\alpha \geq 0$, the following inequality holds true:

$$\delta(q, R) > \frac{\alpha\, \delta(q, P) - 2}{\delta(q, P) + \alpha + 3} \geq \frac{\alpha\varepsilon - 2}{\varepsilon + \alpha + 3}$$

Based on this, it is easy to see that the assignment of $\alpha = (\varepsilon\xi + 3\xi + 2)/(\varepsilon - \xi)$ implies that $\delta(q, R) > \xi$, meaning that any of $q$'s $\xi$-approximate nearest-neighbors in $R$ belong to the same class as $q$'s nearest-neighbor in $P$. Intuitively, this guarantees that $q$ is correctly classified by the $\xi$-ANN rule in $R$.

Case 2 ($\varepsilon$-Approximation guarantee): If $\delta(q, P) < \varepsilon$.

The assignment of $\alpha$ implies that $\mathsf{d}_{\mathrm{nn}}(q, R) < \frac{1+\varepsilon}{1+\xi}\, \mathsf{d}_{\mathrm{nn}}(q, P)$. This means that an $\xi$-ANN query for $q$ in $R$, will return one of $q$'s $\varepsilon$-approximate nearest-neighbors in $P$.

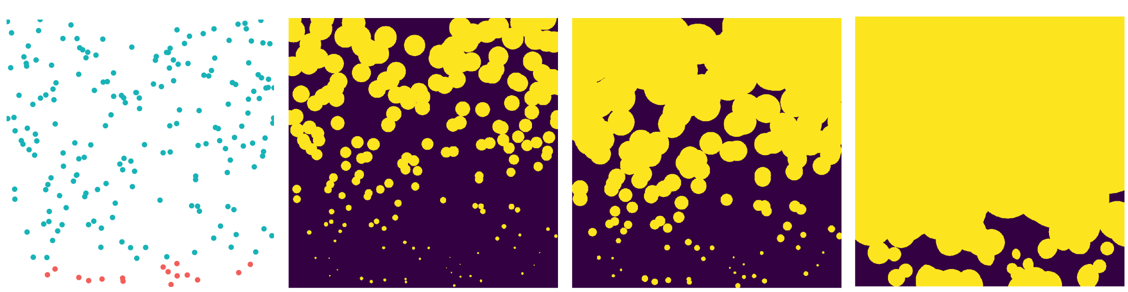
All together, this implies that $R$ is an $(\xi, \varepsilon)$-coreset for the nearest-neighbor rule on $P$. □

In contrast with standard condensation criteria, these new results provide guarantees on either approximation or the correct classification, of any query point in the metric space. This is true even for query points that were "hard" to classify with the entire training set, formally defined as query points with low chromatic density. Consequently, Theorems 5.4 and 5.5 show that $\alpha$ must be set to large values if we hope to provide any sort of guarantees for these query points. However, better results can be achieved by restricting the set of points that are guaranteed to be correctly classified. This relates to the notion of *weak* coresets, which provide approximation guarantees only for a subset of the possible queries. Given $\beta \geq 0$, we define $\mathcal{Q}_\beta$ as the set of query points in $\mathcal{X}$ whose chromatic density with respect to $P$ is at least $\beta$ (*i.e.*, $\mathcal{Q}_\beta = \{q \in \mathcal{X} \mid \delta(q, P) \geq \beta\}$). The following result describes the trade-off between $\alpha$ and $\beta$ to guarantee the correct classification of query points in $\mathcal{Q}_\beta$ after condensation.

**Theorem 5.6.** *Any $\alpha$-consistent subset is a weak $\varepsilon$-coreset for the nearest-neighbor rule for queries in $\mathcal{Q}_\beta$, for $\beta = 2/\alpha$. Moreover, all query points in $\mathcal{Q}_\beta$ are correctly classified.*

The proof of this theorem is rather simple, and follows the same arguments outlined in Case 1 of the proof of Theorem 5.4. Basically, we use Lemma 5.2 to show that for any query point $q \in \mathcal{Q}_\beta$, $q$'s chromatic density after condensation is greater than zero if $\alpha\beta \geq 2$. Note that $\varepsilon$ plays no role in this result, as the guarantee on query points of $\mathcal{Q}_\beta$ is of correct classification (*i.e.*, the class of its *exact* nearest-neighbor in $P$), rather than an approximation.

The trade-off between $\alpha$ and $\beta$ is illustrated in Figure 5.2. From an initial training set $P \subset \mathbb{R}^2$ (Figure 5.2a), we show the regions of $\mathbb{R}^2$ that comprise the sets $\mathcal{Q}_\beta$ for $\beta = 2/\alpha$, using $\alpha = \{0.1, 0.2, \sqrt{2}\}$ (Figures 5.2b-5.2d). While evidently, increasing $\alpha$ guarantees that more query points will be correctly classified after condensation, this example demonstrates a phenomenon commonly observed experimentally: most query points lie far from enemy points, and thus have high chromatic density with respect to $P$. Therefore, while Theorem 5.4 states that $\alpha$ must be set to $2/\varepsilon$ to provide approximation guarantees on all query points, Theorem 5.6 shows that much smaller values of $\alpha$ are sufficient to provide guarantees on some query points, as evidenced in the example in Figure 5.2.



(a) Set $P$ (200 pts) (b) $\mathcal{Q}_{2/\alpha}$ for $\alpha = 0.1$ (c) $\mathcal{Q}_{2/\alpha}$ for $\alpha = 0.2$ (d) $\mathcal{Q}_{2/\alpha}$ for $\alpha = \sqrt{2}$

Figure 5.2: Depiction of the $\mathcal{Q}_\beta$ sets for which any $\alpha$-consistent subset is weak coreset ($\beta = 2/\alpha$). Query points in the *yellow* ● areas are inside $\mathcal{Q}_\beta$, and thus correctly classified after condensation. Query points in the *blue* ● areas are not in $\mathcal{Q}_\beta$, and have no guarantee of correct classification.

These results establish a clear connection between the problem of condensation and that of finding coresets for the nearest-neighbor rule, and provides a roadmap to prove Theorem 5.1. This is the first characterization of sufficient conditions to correctly classify any query point in $\mathcal{X}$ after condensation, and not just the points in $P$ (as the original consistency criteria implies). In the following section, these existential results are matched with algorithms to compute $\alpha$-selective subsets of $P$ of bounded cardinality.

## 5.3 Coreset Computation

### 5.3.1 Hardness Results

Define Min-$\alpha$-CS to be the problem of computing an $\alpha$-consistent subset of minimum cardinality for a given training set $P$. Similarly, let Min-$\alpha$-SS be the corresponding optimization problem for $\alpha$-selective subsets. Following known results from standard condensation [10–12], when $\alpha$ is set to zero, the Min-0-CS and Min-0-SS problems are both known to be NP-hard. Being special cases of the general problems just defined, this implies that both Min-$\alpha$-CS and Min-$\alpha$-SS are NP-hard.

Here we present results related to the hardness of approximation of both problems, along with simple algorithmic approaches with tight approximation factors.

**Theorem 5.7.** *The* Min-$\alpha$-CS *problem is* NP*-hard to approximate in polynomial time within a factor of* $2^{(\mathrm{ddim}(\mathcal{X})\log((1+\alpha)/\gamma))^{1-o(1)}}$.

The full proof is omitted, as it follows from a modification of the hardness bounds proof for the Min-0-CS problem described in [36], which is based on a reduction from the *Label Cover* problem. Proving Theorem 5.7 involves a careful adjustment of the distances in this reduction, so that all the points in the construction have chromatic density at least $\alpha$. Consequently, this would imply that the minimum nearest-enemy distance is reduced by a factor of $1/(1+\alpha)$, explaining the resulting bound for Min-$\alpha$-CS.

The NET algorithm [36] can also be generalized to compute $\alpha$-consistent subsets of $P$ as follows. We define $\alpha$-NET as the algorithm that computes a $\gamma/(1+\alpha)$-net of $P$, where $\gamma$ is the smallest nearest-enemy distance in $P$. The covering property of nets [67] implies that the resulting subset is $\alpha$-consistent, while the packing property suggests that its cardinality is $\mathcal{O}\left(((1+\alpha)/\gamma)^{\mathrm{ddim}(\mathcal{X})+1}\right)$, implying a tight approximation to the Min-$\alpha$-CS problem.

**Theorem 5.8.** *The* Min-$\alpha$-SS *problem is* NP*-hard to approximate in polynomial time within a factor of* $(1-o(1))\ln n$ *unless* $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{\log\log n})$.

*Proof.* The result follows from the hardness of another related covering problem: the minimum *dominating set* [68–70]. We describe a simple L-reduction from any instance of this problem to an instance of Min-$\alpha$-SS, which preserves the approximation ratio.

1. Consider any instance of minimum dominating set, consisting of the graph $G = (V, E)$.

2. Generate a new edge-weighted graph $G'$ as follows:
   Create two copies of $G$, namely $G_\mathsf{r} = (V_\mathsf{r}, E_\mathsf{r})$ and $G_\mathsf{b} = (V_\mathsf{b}, E_\mathsf{b})$, of *red* and *blue* nodes respectively. Set all edge-weights of $G_\mathsf{r}$ and $G_\mathsf{b}$ to be 1. Finally, connect each red node $v_\mathsf{r}$ to its corresponding blue node $v_\mathsf{b}$ by an edge $\{v_\mathsf{r}, v_\mathsf{b}\}$ of weight $1 + \alpha + \xi$ for a sufficiently small constant $\xi > 0$. Formally, $G'$ is defined as the edge-weighted graph $G' = (V', E')$ where the set of nodes is $V' = V_\mathsf{r} \cup V_\mathsf{b}$, the set of edges is $E' = E_\mathsf{r} \cup E_\mathsf{r} \cup \{\{v_\mathsf{r}, v_\mathsf{b}\} \mid v \in V\}$, and an edge-weight function $w : E' \to \mathbb{R}^+$ where $w(e) = 1$ iff $e \in E_\mathsf{r} \cup E_\mathsf{b}$, and $w(e) = 1 + \alpha + \xi$ otherwise.

3. A labeling function $l$ where $l(v) = \mathit{red}$ iff $v \in V_\mathsf{r}$, and $l(v) = \mathit{blue}$ iff $v \in V_\mathsf{b}$.

4. Compute the shortest-path metric of $G'$, denoted as $\mathsf{d}_{G'}$.

5. Solve the Min-$\alpha$-SS problem for the set $V'$, on metric $\mathsf{d}_{G'}$, and the labels defined by $l$.

A dominating set of $G$ consists of a subset of nodes $D \subseteq V$, such that every node $v \in V \setminus D$ is adjacent to a node in $D$. Given any dominating set $D \subseteq V$ of $G$, it is easy to see that the subset $R = \{v_\mathsf{r}, v_\mathsf{b} \mid v \in D\}$ is an $\alpha$-selective subset of $V'$, where $|R| = 2|D|$. Similarly, given an $\alpha$-selective subset $R \subseteq V'$, there is a corresponding dominating set $D$ of $G$, where $|D| \leq |R|/2$, as $D$ can be either $R \cap V_\mathsf{r}$ or $R \cap V_\mathsf{b}$. Therefore, Min-$\alpha$-SS is as hard to approximate as the minimum dominating set problem. □

There is a clear connection between the Min-$\alpha$-SS problem and covering problems, in particular that of finding an optimal hitting set. Given a set of elements $U$ and a family $C$ of subsets of $U$, a *hitting set* of $(U, C)$ is a subset $H \subseteq U$ such that every set in $C$ contains at least one element of $H$. Therefore, let $N_{p,\alpha}$ be the set of points of $P$ whose distance to $p$ is less than $\mathsf{d}_{\mathrm{ne}}(p)/(1 + \alpha)$, then any hitting set of $(P, \{N_{p,\alpha} \mid p \in P\})$ is also an $\alpha$-selective subset of $P$, and vice versa. This simple reduction implies a $\mathcal{O}(n^3)$ worst-case time $\mathcal{O}(\log n)$-approximation algorithm for Min-$\alpha$-SS, based on the classic greedy algorithm for set cover [71, 72]. Call this approach $\alpha$-HSS or *$\alpha$-Hitting Selective Subset*. It follows from Theorem 5.8 that for training sets in general metric spaces, this is the best approximation possible under standard complexity assumptions.

While both $\alpha$-NET and $\alpha$-HSS compute tight approximations of their corresponding problems, their performance in practice does not compare to heuristic approaches for standard condensation (see Section 5.4 for experimental results). Therefore, in the following subsections, we consider two practical algorithms for this problem, namely SFCNN and RSS, and extend them to compute subsets with the newly defined criteria.

### 5.3.2 An Algorithm for $\alpha$-Selective Subsets

For standard condensation, we have already analyzed the RSS algorithm (see Chapter 4) used to compute selective subsets. It runs in quadratic worst-case time and exhibits good performance in practice. The selection process of this algorithm is heuristic in nature and can be described as follows: beginning with an empty set, the points in $p \in P$ are examined in increasing order with respect to their nearest-enemy distance $\mathsf{d}_{\text{ne}}(p)$. The point $p$ is added to the subset $R$ if $\mathsf{d}_{\text{nn}}(p, R) \geq \mathsf{d}_{\text{ne}}(p)$. It is easy to see that the resulting subset is selective.

Now, we define a generalization called $\alpha$-RSS, to compute $\alpha$-selective subsets of $P$. The condition to add a given point $p \in P$ to the selected subset checks if any previously selected point is closer to $p$ than $\mathsf{d}_{\text{ne}}(p)/(1+\alpha)$, instead of just $\mathsf{d}_{\text{ne}}(p)$. See Algorithm 10 for a formal description, and Figure 5.3 for an illustration. It is easy to see that this algorithm computes an $\alpha$-selective subset, while keeping the quadratic time complexity of the original RSS algorithm.

**Algorithm 10:** $\alpha$-RSS

**Input:** Initial training set $P$ and parameter $\alpha \geq 0$
**Output:** $\alpha$-selective subset $R \subseteq P$

```
R ← φ
Let {p_i}_{i=1}^n be the points of P sorted increasingly w.r.t. their nearest-enemy
  distance d_ne(p_i)
foreach p_i ∈ P, where i = 1 ... n do
    if (1 + α) · d_nn(p_i, R) ≥ d_ne(p_i) then
        R ← R ∪ {p_i}
return R
```

Naturally, we want to analyze the number of points this algorithm selects. The remainder of this section establishes upper-bounds and approximation guarantees of the $\alpha$-RSS algorithm for any doubling metric space, with improved results in the Euclidean space. This proves the result mentioned in Chapter 4 that RSS computes an approximation of the Min-0-CS and Min-0-SS problems.

#### 5.3.2.1 Size in Doubling spaces

First, we consider the case where the underlying metric space $(\mathcal{X}, \mathsf{d})$ of $P$ is doubling. The following results depend on the doubling dimension $\text{ddim}(\mathcal{X})$ of the metric space (which is assumed to be constant), the margin $\gamma$ (the smallest nearest-enemy distance of any point in $P$), and $\kappa$ (the number of nearest-enemy points in $P$).

**Theorem 5.9.** *$\alpha$-RSS computes a tight approximation for the* Min-$\alpha$-CS *problem.*

*Proof.* This follows from a direct comparison to the resulting subset of the $\alpha$-NET algorithm from the previous section. For any point $p$ selected by $\alpha$-NET, let $B_{p,\alpha}$

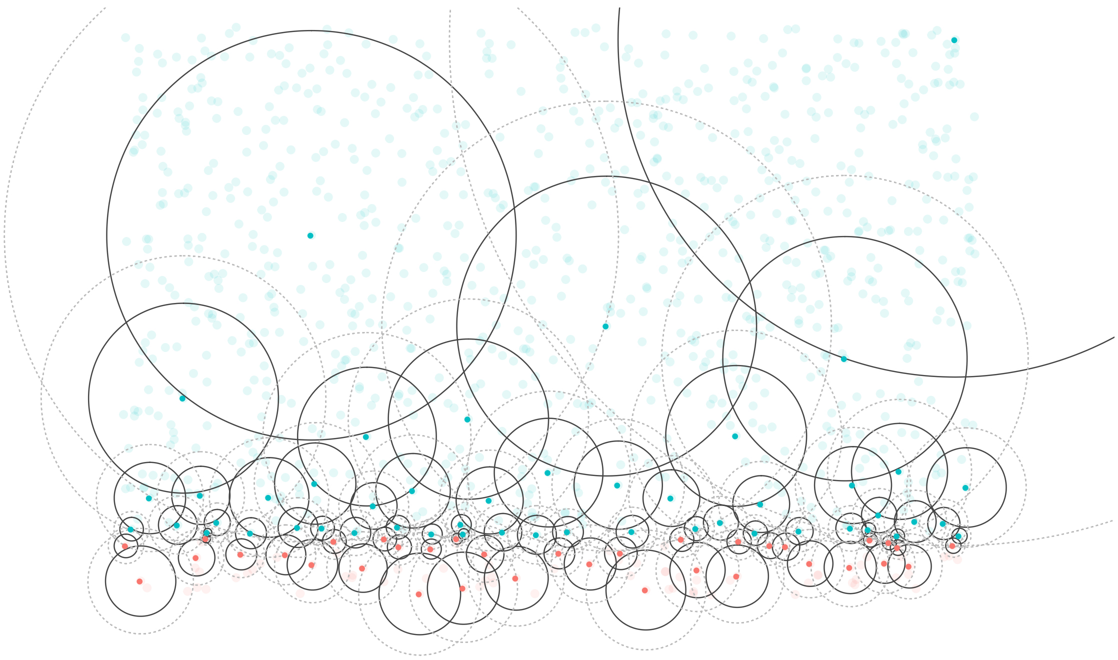


Figure 5.3: Selection of $\alpha$-RSS for $\alpha$=0.5. Faded points are not selected, while selected points are drawn along with a ball of radius $\mathsf{d}_{\text{ne}}(p)$ (dotted outline) and a ball of radius $\mathsf{d}_{\text{ne}}(p)/(1+\alpha)$ (solid outline). A point $p$ is selected if no previously selected point is closer to $p$ than $\mathsf{d}_{\text{ne}}(p)/(1+\alpha)$.

be the set of points of $P$ "covered" by $p$, that is, whose distance to $p$ is at most $\gamma/(1+\alpha)$. By the covering property of $\varepsilon$-nets, this defines a partition on $P$ when considering every point $p$ selected by $\alpha$-NET.

Let $R$ be the set of points selected by $\alpha$-RSS, we analyze the size of $B_{p,\alpha} \cap R$, that is, for any given $B_{p,\alpha}$ how many points could have been selected by the $\alpha$-RSS algorithm. Let $a, b \in B_{p,\alpha} \cap R$ be any two such points, where without loss of generality, $\mathsf{d}_{\text{ne}}(a) \leq \mathsf{d}_{\text{ne}}(b)$. By the selection process of the algorithm, we know that $\mathsf{d}(a,b) \geq \mathsf{d}_{\text{ne}}(b)/(1+\alpha) \geq \gamma/(1+\alpha)$. A simple packing argument in doubling metrics implies that $|B_{p,\alpha} \cap R| \leq 2^{\text{ddim}(\mathcal{X})+1}$. Altogether, we have that the size of the subset selected by $\alpha$-RSS is $\mathcal{O}\left((2(1+\alpha)/\gamma)^{\text{ddim}(\mathcal{X})+1}\right)$. □

**Theorem 5.10.** *$\alpha$-RSS computes an $\mathcal{O}(\log(\min(1+2/\alpha, 1/\gamma)))$-factor approximation for the* Min-$\alpha$-SS *problem. For $\alpha = \Omega(1)$, this is a constant-factor approximation.*

*Proof.* Let $\text{OPT}_\alpha$ be the optimum solution to the Min-$\alpha$-SS problem, *i.e.*, the minimum cardinality $\alpha$-selective subset of $P$. For every point $p \in \text{OPT}_\alpha$ in such solution, define $S_{p,\alpha}$ to be the set of points in $P$ "covered" by $p$, or simply $S_{p,\alpha} = \{r \in P \mid \mathsf{d}(r,p) < \mathsf{d}_{\text{ne}}(r)/(1+\alpha)\}$. Additionally, let $R$ be the set of points selected by $\alpha$-RSS, define $R_{p,\sigma}$ to be the points selected by $\alpha$-RSS which also belong to $S_{p,\alpha}$ and whose nearest-enemy distance is between $\sigma$ and $2\sigma$, for $\sigma \in [\gamma, 1]$. That is,

$R_{p,\sigma} = \{ r \in R \cap S_{p,\alpha} \mid \mathsf{d}_{\mathrm{ne}}(r) \in [\sigma, 2\sigma) \}$. Clearly, these subsets define a partitioning of $R$ for all $p \in \mathrm{OPT}_\alpha$ and values of $\sigma = \gamma\, 2^i$ for $i = \{0, 1, 2, \dots, \lceil \log \frac{1}{\gamma} \rceil\}$.

However, depending on $\alpha$, some values of $\sigma$ would yield empty $R_{p,\sigma}$ sets. Consider some point $q \in S_{p,\alpha}$, we can bound its nearest-enemy distance with respect to the nearest-enemy distance of point $p$. In particular, by leveraging simple triangle-inequality arguments, it is possible to prove that $\frac{1+\alpha}{2+\alpha}\, \mathsf{d}_{\mathrm{ne}}(p) \leq \mathsf{d}_{\mathrm{ne}}(q) \leq \frac{1+\alpha}{\alpha}\, \mathsf{d}_{\mathrm{ne}}(p)$. Therefore, the values of $\sigma$ for which $R_{p,\sigma}$ sets are not empty, are $\sigma = 2^j\, \frac{1+\alpha}{2+\alpha}\, \mathsf{d}_{\mathrm{ne}}(p)$ for $j = \{0, \dots, \lceil \log{(1 + 2/\alpha)} \rceil\}$.

The proof now follows by bounding the size of $R_{p,\sigma}$ which can be achieved by bounding its spread. Thus, lets consider the smallest and largest pairwise distances among points in $R_{p,\sigma}$. Take any two points $a, b \in R_{p,\sigma}$ where without loss of generality, $\mathsf{d}_{\mathrm{ne}}(a) \leq \mathsf{d}_{\mathrm{ne}}(b)$. Note that points selected by $\alpha$-RSS cannot be "too close" to each other; that is, as $a$ and $b$ were selected by the algorithm, we know that $(1+\alpha)\, \mathsf{d}(a,b) \geq \mathsf{d}_{\mathrm{ne}}(b) \geq \sigma$. Therefore, the smallest pairwise distance in $R_{p,\sigma}$ is at least $\sigma/(1+\alpha)$. Additionally, by the triangle inequality, we can bound the maximum pairwise distance using their distance to $p$ as $\mathsf{d}(a,b) \leq \mathsf{d}(a,p) + \mathsf{d}(p,b) \leq 4\sigma/(1+\alpha)$. Then, by the packing properties of doubling spaces, the size of $R_{p,\sigma}$ is at most $4^{\mathrm{ddim}(\mathcal{X})+1}$.

Altogether, for every $p \in \mathrm{OPT}_\alpha$ there are up to $\lceil \log{(\min{(1 + 2/\alpha, 1/\gamma)})} \rceil$ non-empty $R_{p,\sigma}$ subsets, each containing at most $4^{\mathrm{ddim}(\mathcal{X})+1}$ points. In doubling spaces with constant doubling dimension, the size of these subsets is also constant. □

While these results are meaningful from a theoretical perspective, it is also useful to establishing bounds in terms of the geometry of the learning space, which is characterized by the boundaries between points of different classes. Thus, using similar packing arguments as above, we bound the selection size of the algorithm with respect to $\kappa$.

**Theorem 5.11.** $\alpha$-RSS *selects* $\mathcal{O}\Big(\kappa \log \frac{1}{\gamma}\ (1+\alpha)^{\mathrm{ddim}(\mathcal{X})+1}\Big)$ *points.*

*Proof.* This follows from similar arguments to the ones used to prove Theorem 5.10, using an alternative charging scheme for each nearest-enemy point in the training set. Consider one such point $p \in \{\mathrm{ne}(r) \mid r \in P\}$ and a value $\sigma \in [\gamma, 1]$, we define $R'_{p,\sigma}$ to be the subset of points from $\alpha$-RSS whose nearest-enemy is $p$, and their nearest-enemy distance is between $\sigma$ and $2\sigma$. That is, $R'_{p,\sigma} = \{ r \in R \mid \mathrm{ne}(r) = p \wedge \mathsf{d}_{\mathrm{ne}}(r) \in [\sigma, 2\sigma) \}$. These subsets partition $R$ for all nearest-enemy points of $P$, and values of $\sigma = \gamma\, 2^i$ for $i = \{0, 1, 2, \dots, \lceil \log \frac{1}{\gamma} \rceil\}$.

For any two points $a, b \in R'_{p,\sigma}$, the selection criteria of $\alpha$-RSS implies some separation between selected points, which can be used to prove that $\mathsf{d}(a,b) \geq \sigma/(1+\alpha)$. Additionally, we know that $\mathsf{d}(a,b) \leq \mathsf{d}(a,p) + \mathsf{d}(p,b) = \mathsf{d}_{\mathrm{ne}}(a) + \mathsf{d}_{\mathrm{ne}}(b) \leq 4\sigma$. Using a simple packing argument, we have that $|R'_{p,\sigma}| \leq \lceil 4(1+\alpha) \rceil^{\mathrm{ddim}(\mathcal{X})+1}$.

Altogether, by counting all sets $R'_{p,\sigma}$ for each nearest-enemy in the training set and values of $\sigma$, the size of $R$ is upper-bounded by $|R| \leq \kappa\, \lceil \log 1/\gamma \rceil\, \lceil 4(1+\alpha) \rceil^{\mathrm{ddim}(\mathcal{X})+1}$. Based on the assumption that $\mathrm{ddim}(\mathcal{X})$ is constant, this completes the proof. □

As a corollary, this result implies that when $\alpha = 2/\varepsilon$, the $\alpha$-selective subset computed by $\alpha$-RSS contains $\mathcal{O}\left(\kappa \log 1/\gamma \; (1/\varepsilon)^{\mathrm{ddim}(\mathcal{X})+1}\right)$ points. This establishes the size bound on the $\varepsilon$-coreset given in Theorem 5.1, which can be computed using the $\alpha$-RSS algorithm.

#### 5.3.2.2 Size in Euclidean space

In the case where $P \subset \mathbb{R}^d$ lies in $d$-dimensional Euclidean space, the analysis of $\alpha$-RSS can be further improved, leading to a constant-factor approximation of Min-$\alpha$-SS for values of $\alpha \geq 0$, and reduced dependency on the dimensionality of $P$.

**Theorem 5.12.** $\alpha$-RSS *computes an* $\mathcal{O}(1)$*-approximation for the* Min-$\alpha$-SS *problem in* $\mathbb{R}^d$.

*Proof.* Similar to the proof of Theorem 5.10, define $R_p = S_{p,\alpha} \cap R$ as the points selected by $\alpha$-RSS that are "covered" by $p$ in the optimum solution $\mathrm{OPT}_\alpha$. Consider two such points $a, b \in R_p$ where without loss of generality, $\mathsf{d}_{\mathrm{ne}}(a) \leq \mathsf{d}_{\mathrm{ne}}(b)$. By the definition of $S_{p,\alpha}$ we know that $\mathsf{d}(a,p) < \mathsf{d}_{\mathrm{ne}}(a)/(1+\alpha)$, and similarly with $b$. Additionally, from the selection of the algorithm we know that $\mathsf{d}(a,b) \geq \mathsf{d}_{\mathrm{ne}}(b)/(1+\alpha)$. Overall, these inequalities imply that the angle $\angle apb \geq \pi/3$. By a simple packing argument, the size of $R_p$ is bounded by the *kissing number* in $d$-dimensional Euclidean space, or simply $\mathcal{O}((3/\pi)^{d-1})$. Therefore, we have that $|R| \leq \sum_p |R_p| = |\mathrm{OPT}_\alpha| \; \mathcal{O}((3/\pi)^{d-1})$. Assuming $d$ is constant, this completes the proof. □

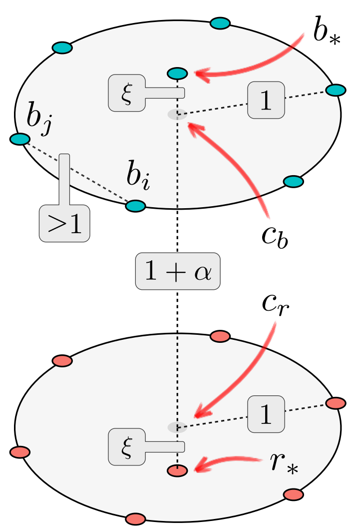



Figure 5.4: Training set where the analysis of the approximation factor of $\alpha$-RSS in $\mathbb{R}^d$ is tight.

This analysis is tight up to constant factors. In Figure 5.4, we illustrate a training set $P$ consisting of *red* and *blue* points in $\mathbb{R}^d$, where $\alpha$-RSS selects $\Theta(c^{d-2} \; |\mathrm{OPT}_\alpha|)$ points. Consider two helper points (which do not belong to $P$) $c_r = 0\vec{u}_d$ and $c_b = (1+\alpha)\vec{u}_d$, where $\vec{u}_d$ is the unit vector parallel to the $d$-th coordinate. Add red points $r_i$ on the surface of the $d-1$ unit ball centered at $c_r$ and perpendicular to $\vec{u}_d$. Similarly with blue points $b_i$ around $c_b$. Finally, add two points $r_* = -\xi\vec{u}_d$ and $b_* = (1+\alpha+\xi)\vec{u}_d$, for a suitable value $\xi$ such that $\|r_* r_i\| < 1$. Clearly, the nearest-enemy distance of all $r_i$ and $b_i$ points is $1+\alpha$, while the one of $r_*$ and $b_*$ is strictly greater than $1+\alpha$. Thus, $\mathrm{OPT}_\alpha = \{r_*, b_*\}$ but $\alpha$-RSS selects $\Theta(c^{d-2})$ points $r_i$ and $b_i$ at distance greater than 1 from each other.

Furthermore, a similar constant-factor approximation can be achieved for any training set $P$ in $\ell_p$ space for $p \geq 3$. This follows analogously to the proof of Theorem 5.12, exploiting the bounds between $\ell_p$ and $\ell_2$ metrics, where $1/\sqrt{d}\,\|v\|_p \leq \|v\|_2 \leq \sqrt{d}\,\|v\|_p$. This would imply that the angle between any two points in $\alpha$-$\mathrm{RSS}_p$ is $\Omega(1/d)$. Therefore, it shows that $\alpha$-RSS achieves an approximation factor of $\mathcal{O}(d^{d-1})$, or simply $\mathcal{O}(1)$ for constant dimension.

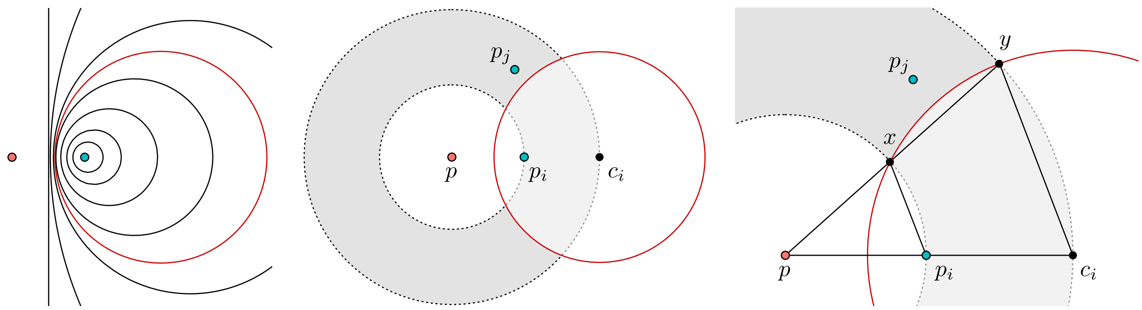
Similarly to the case of doubling spaces, we also establish upper-bounds in terms of $\kappa$ for the selection size of the algorithm in Euclidean space. The following result improves the exponential dependence on the dimensionality of $P$ (from $\mathsf{ddim}(\mathbb{R}^d) = \Theta(d)$ to $d-1$), while keeping the dependency on the margin $\gamma$, which contrast with the approximation factor results.

**Theorem 5.13.** *In Euclidean space $\mathbb{R}^d$, $\alpha$-RSS selects $\mathcal{O}\Big(\kappa \log \frac{1}{\gamma}\ (1+\alpha)^{d-1}\Big)$ points.*

*Proof.* Let $p$ be any nearest-enemy point of $P$ and $\sigma \in [\gamma, 1]$, similarly define $R'_{p,\sigma}$ to be the set of points selected by $\alpha$-RSS whose nearest-enemy is $p$ and their nearest-enemy distance is between $\sigma$ and $b\sigma$, for $b = \frac{(1+\alpha)^2}{\alpha(2+\alpha)}$. Equivalently, these subsets define a partitioning of $R$ for all nearest-enemy points $p$ and values of $\sigma = \gamma\, b^k$ for $k = \{0, 1, 2, \ldots, \lceil \log_b \frac{1}{\gamma} \rceil\}$. Thus, the proof follows from bounding the minimum angle between points in these subsets. For any two such points $p_i, p_j \in R'_{p,\sigma}$, we lower bound the angle $\angle p_i p p_j$. Assume without loss of generality that $\mathsf{d}_{\mathrm{ne}}(p_i) \leq \mathsf{d}_{\mathrm{ne}}(p_j)$. By definition of the partitioning, we also know that $\mathsf{d}_{\mathrm{ne}}(p_j) \leq b\sigma \leq b\,\mathsf{d}_{\mathrm{ne}}(p_i)$. Therefore, altogether we have that $\mathsf{d}_{\mathrm{ne}}(p_i) \leq \mathsf{d}_{\mathrm{ne}}(p_j) \leq b\,\mathsf{d}_{\mathrm{ne}}(p_i)$.

First, consider the set of points whose distance to $p_i$ is $(1+\alpha)$ times their distance to $p$, which defines a multiplicative weighted bisector [73] between points $p$ and $p_i$, with weights equal to 1 and $1/(1+\alpha)$ respectively. This is characterized as a $d$-dimensional ball (see Figure 5.5a) with center $c_i = (p_i - p)\,b + p$ and radius $\mathsf{d}_{\mathrm{ne}}(p_i)\,b/(1+\alpha)$. Thus $p$, $p_i$ and $c_i$ are collinear, and the distance between $p$ and $c_i$ is $\mathsf{d}(p, c_i) = b\,\mathsf{d}_{\mathrm{ne}}(p_i)$. In particular, let's consider the relation between $p_j$ and such bisector. As $p_j$ was selected by the algorithm after $p_i$, we know that $(1+\alpha)\,\mathsf{d}(p_j, p_i) \geq \mathsf{d}_{\mathrm{ne}}(p_j)$ where $\mathsf{d}_{\mathrm{ne}}(p_j) = \mathsf{d}(p_j, p)$. Therefore, clearly $p_j$ lies either outside or in the surface of the weighted bisector between $p$ and $p_i$ (see Figure 5.5b).

For angle $\angle p_i p p_j$, we can frame the analysis to the plane defined by $p$, $p_i$ and $p_j$. Let $x$ and $y$ be two points in this plane, such that they are the intersection points between the weighted bisector and the balls centered at $p$ of radii $\mathsf{d}_{\mathrm{ne}}(p_i)$ and $b\,\mathsf{d}_{\mathrm{ne}}(p_i)$ respectively (see Figure 5.5c). By the convexity of the weighted bisector between $p$ and $p_i$, we can say that $\angle p_i p p_j \geq \min(\angle xpp_i, \angle ypc_j)$. Now, consider the triangles $\triangle pxp_i$ and $\triangle pyc_i$. By the careful selection of $b$, these triangles are both isosceles and similar. In particular, for $\triangle pxp_i$ the two sides incident to $p$ have




(a) Multiplicatively weighted bisectors for different weights.

(b) Position of point $p_j$ *w.r.t.* the weighted bisector between points $p$ and $p_i$.

(c) The intersection points $x$ and $y$ between the weighted bisector and the limit balls of $R_{p,\sigma}$.

Figure 5.5: Construction for the analysis of the minimum angle between two points in $R'_{p,\sigma}$ *w.r.t.* some nearest-enemy point $p \in P$. Let points $p_i, p_j \in R'_{p,\sigma}$, we analyze the angle $\angle p_i p p_j$.

length equal to $\mathsf{d}_{\mathrm{ne}}(p_i)$, and the side opposite to $p$ has length equal to $\mathsf{d}_{\mathrm{ne}}(p_i)/(1+\alpha)$. For $\triangle pyc_i$, the side lengths are $b\,\mathsf{d}_{\mathrm{ne}}(p_i)$ and $\mathsf{d}_{\mathrm{ne}}(p_i)\,b/(1+\alpha)$. Therefore, the angle $\angle p_i p p_j \geq \angle x p p_i \geq 1/(1+\alpha)$.

By a simple packing argument based on this minimum angle, we have that the size of $R'_{p,\sigma}$ is $\mathcal{O}((1+\alpha)^{d-1})$. All together, following the defined partitioning, we have that:

$$|R| = \sum_{p} \sum_{k=0}^{\lceil \log_b \frac{1}{\gamma} \rceil} |R'_{p,b^k}| \leq \kappa \left\lceil \log_b \frac{1}{\gamma} \right\rceil \mathcal{O}\left((1+\alpha)^{d-1}\right)$$

For constant $\alpha$ and $d$, the size of $\alpha$-RSS is $\mathcal{O}(\kappa \log \frac{1}{\gamma})$. Moreover, when $\alpha$ is zero $\alpha$-RSS selects $\mathcal{O}(\kappa\, c^{d-1})$, matching the bound presented in Chapter 4 for RSS in Euclidean space. □

#### 5.3.2.3 Subquadratic Implementation

Here we present a subquadratic implementation for the $\alpha$-RSS algorithm, which completes the proof of our main result, Theorem 5.1. Prior to this result , among algorithms for nearest-neighbor condensation, FCNN and SFCNN achieve the best worst-case time complexity, running in $\mathcal{O}(nm)$ time, where $m = |R|$ is the size of the selected subset.

The $\alpha$-RSS algorithm consists of two main stages: computing the nearest-enemy distances of all points in $P$ (and sorting the points based on these), and the selection process itself. The first stage requires a total of $n$ nearest-enemy queries, plus additional $\mathcal{O}(n \log n)$ time for sorting. The second stage performs $n$ nearest-neighbor queries on the current selected subset $R$, which needs to be updated $m$ times. In both cases, using exact nearest-neighbor search would degenerate into linear search due to the *curse of dimensionality*. Thus, the first and second stage of the algorithm would need $\mathcal{O}(n^2)$ and $\mathcal{O}(nm)$ worst-case time respectively.

These bottlenecks can be overcome by leveraging approximate nearest-neighbor techniques. Clearly, the first stage of the algorithm can be improved by computing nearest-enemy distances approximately, using as many ANN structures as classes there are in $P$, which is considered to be a small constant. Therefore, by also applying a simple brute-force search for nearest-neighbors in the second stage, result (i) of the next theorem follows immediately. Moreover, by combining this with standard techniques for static-to-dynamic conversions [74], we have result (ii) below. Denote this variant of $\alpha$-RSS as $(\alpha, \xi)$-RSS, for a parameter $\xi \geq 0$.

**Theorem 5.14.** *Given a data structure for $\xi$-ANN searching with construction time $t_c$ and query time $t_q$ (which potentially depend on $n$ and $\xi$), the $(\alpha, \xi)$-RSS variant can be implemented with the following worst-case time complexities, where $m$ is the size of the selected subset.*

*(i)* $\mathcal{O}\left(t_c + n\left(t_q + m + \log n\right)\right)$

*(ii)* $\mathcal{O}\left(\left(t_c + n\, t_q\right) \log n\right)$

More generally, if we are given an additional data structure for dynamic $\xi$-ANN searching with construction time $t'_c$, query time $t'_q$, and insertion time $t'_i$, the overall running time will be $\mathcal{O}\left(t_c + t'_c + n\left(t_q + t'_q + \log n\right) + m\, t'_i\right)$. Indeed, this can be used to obtain (ii) from the static-to-dynamic conversions [74], which propose an approach to convert static search structures into dynamic ones. These results directly imply implementations of $(\alpha, \xi)$-RSS with subquadratic worst-case time complexities, based on ANN techniques [43, 75] for low-dimensional Euclidean space, and using techniques like LSH [50] that are suitable for ANN in high-dimensional Hamming and Euclidean spaces. More generally, subquadratic runtimes can be achieved by leveraging techniques [76] for dynamic ANN search in doubling spaces.

**Algorithm 11:** $(\alpha, \varepsilon)$-RSS

**Input:** Initial training set $P$ and parameters $\alpha, \varepsilon \geq 0$
**Output:** $\alpha$-selective subset $R \subseteq P$

```
R ← φ
Let {p_i}_{i=1}^n be the points of P sorted increasingly w.r.t. their approximate
   nearest-enemy distance d_ne(p_i, ε)
foreach p_i ∈ P, where i = 1...n do
    if (1 + α)(1 + ε) · d_nn(p_i, R, ε) ≥ d_ne(p_i, ε) then
        R ← R ∪ {p_i}
return R
```

It remains unclear how these new implementations of the algorithm would deal with "uncertainty". That is, such implementation schemes for $\alpha$-RSS would incur an approximation error (of up to $1 + \xi$) on the computed distances: either only during the first stage if (i) is implemented, or during both stages if (ii) or the dynamic-structure scheme are implemented. The uncertainty introduced by these approximate queries, imply that in order to guarantee finding $\alpha$-selective subsets, we must modify the condition for adding point during the second stage of the algorithm. Let $\mathsf{d}_{\mathrm{ne}}(p, \xi)$ denote the $\xi$-approximate nearest-enemy distance of $p$ computed in the first stage, and let $\mathsf{d}_{\mathrm{nn}}(p, R, \xi)$ denote the $\xi$-approximate nearest-neighbor distance of $p$ over points of the current subset (computed in the second stage). Then, $(\alpha, \xi)$-RSS adds a point $p$ into the subset if $(1+\xi)(1+\alpha)\, \mathsf{d}_{\mathrm{nn}}(p, R, \xi) \geq \mathsf{d}_{\mathrm{ne}}(p, \xi)$.

By similar arguments to the ones described in Section 5.3.2, size guarantees can be extended to $(\alpha, \xi)$-RSS. First, the size of the subset selected by $(\alpha, \xi)$-RSS, in terms of the number of nearest-enemy points in the set, would be bounded by the size of the subset selected by $\hat{\alpha}$-RSS with $\hat{\alpha} = (1+\alpha)(1+\xi)^2 - 1$. Additionally, the approximation factor of $(\alpha, \xi)$-RSS in both doubling and Euclidean metric spaces would increase by a factor of $\mathcal{O}((1+\xi)^{2(\mathrm{ddim}(\mathcal{X})+1)})$.

This completes the proof of Theorem 5.1.

**Lemma 5.15.** *There exist a data structure for dynamic $\xi$-*ANN *queries in sets $P$ in $d$-dimensional Euclidean space, that can be constructed in $t'_c = \mathcal{O}(n \log n)$ time, queried in $t'_q = \mathcal{O}(\log n + 1/\xi^{d-1})$ time, and where points of $P$ can be inserted in $t'_i = \mathcal{O}(\log n)$ time.*

Together with Theorem 5.14 described above, this lemma implies that there is a variant of $\alpha$-RSS for Euclidean space that runs in $\mathcal{O}(n \log n + n/\xi^{d-1})$ time. Such data structure can be build from a standard BBD tree [42, 77] as follows. First, construct the tree from the entire set $P$, thus taking $t'_c = \mathcal{O}(n \log n)$ time. However, each node of the tree has some additional data: a boolean flag indicating if the subtree rooted at such node contains a point of the "active" subset $R$. Initially, all flags are set to *false*, making the initial active subset being empty. To add a point $p \in P$ to the active subset $R$, all the flags from the root of the tree to the leaf node containing $p$ must be set to *true*, thus making the insertion time $t'_i = \mathcal{O}(\log n)$. Finally, an $\xi$-ANN query on such tree would perform as usual, only avoiding to visit nodes whose flag is set to *false*, yielding a query time of $t'_q = \mathcal{O}(\log n + 1/\xi^{d-1})$.

### 5.3.3 An Algorithm for $\alpha$-Consistent Subsets

Even thought the main result of this chapter relies on the computation of $\alpha$-selective subsets, Theorem 5.6 shows that even $\alpha$-consistency is enough to guarantee the correct classification of certain query points. In practice, FCNN [16] has been acknowledged as the most efficient algorithm for computing consistent subsets. From the results presented in Chapter 4, we know that while FCNN cannot be upper-bounded in terms of $k$ or $\kappa$, the simple modification of this algorithm called SFCNN can be successfully upper-bounded. Therefore, in this section, we discuss a simple extension of this algorithm in order to compute $\alpha$-consistent subsets.

Recall the workings of both the FCNN and SFCNN algorithms, which select points iteratively as follows. First, the subset $R$ is initialized with one point per class (*e.g.*, the centroids of each class). During every iteration, the algorithm identifies all the points in $P$ that are incorrectly classified with the current $R$, or simply, those whose nearest-neighbor in $R$ is of different class. This is formalized as the *voren* function, defined for every point $p \in R$ as follows:

$$\text{voren}(p, R, P) = \{q \in P \mid \text{nn}(q, R) = p \wedge l(q) \neq l(p)\}$$

This function identifies all the enemies of $p$ whose nearest-neighbor in $R$ is $p$ itself. The only difference between the original FCNN algorithm and the modified SFCNN appears next. While FCNN adds one point per each $p \in R$ in a batch[2], potentially doubling the size of $R$, SFCNN adds only *one* point per iteration. Then, both algorithms terminate when no other points can be added (*i.e.*, all voren$(p, R, P)$ are empty), implying that $R$ is consistent.

We can now extend both algorithms to compute $\alpha$-consistent subsets, namely $\alpha$-FCNN and $\alpha$-SFCNN, by redefining the *voren* function. The idea is simple: to identify those points whose nearest-neighbor in $R$ is $p$, such that are either enemies of $p$, or whose chromatic density with respect to $R$ is less than $\alpha$. This is formally defined as follows:

$$\text{voren}(\alpha, p, R, P) = \{q \in P \mid \text{nn}(q, R) = p \wedge (l(q) = l(p) \Rightarrow \delta(q, R) < \alpha)\}$$

By plugging this function into the algorithms (see Algorithm 12), it is easy to show that the resulting subsets are $\alpha$-consistent. Moreover, this can be easily implemented to run in $\mathcal{O}(nm)$ worst-case time, where $m$ is the final size of $R$, extending the implementation scheme described in the paper where FCNN was initially proposed [16].

**Algorithm 12:** $\alpha$-SFCNN

**Input:** Initial training set $P$ and parameter $\alpha \geq 0$
**Output:** $\alpha$-consistent subset $R \subseteq P$

```
1 R ← φ
2 S ← centroids(P)
3 while S ≠ φ do
4     R ← R ∪ {Choose one point of S}
5     S ← φ
6     foreach p ∈ R do
7         S ← S ∪ {rep(p, voren(α, p, R, P))}
8 return R
```

Finally, leveraging the analysis described in Section 4.3.3, together with the proofs of Theorems 5.9 and 5.11, we upper-bound the selection size of the $\alpha$-SFCNN algorithm. The proofs of the next results depend on the following observation. Let $a, b \in R$ be two points selected by $\alpha$-SFCNN, where $\mathsf{d}_{\text{ne}}(a), \mathsf{d}_{\text{ne}}(b) \geq \beta$ for some $\beta \geq 0$, it is easy to show that $\mathsf{d}(a, b) \geq \beta/(1 + \alpha)$. This follows from a fairly simple

[2]For FCNN, line 4 of Algorithm 12 updates $R$ by adding all the points in $S$, instead of only one point of $S$.

argument: to the contrary, suppose that $\mathsf{d}(a, b) < \beta/(1+\alpha)$, which would imply that $a$ and $b$ belong to the same class. Without loss of generality, point $a$ was added to $R$ before point $b$. Note that after adding point $a$ to $R$, the chromatic density of $b$ *w.r.t.* $R$ is $\delta(b, R) > \alpha$, which contradicts the statement that $b$ could be added to $R$.

**Theorem 5.16.** $\alpha$-SFCNN *computes a tight approximation for the* MIN-$\alpha$-CS *problem.*

This result follows by similar arguments as the proof of Theorem 5.9. By considering any two points $a, b \in B_{p,\alpha} \cap R$, we know that $\mathsf{d}(a, b) \geq \gamma/(1+\alpha)$ as $\gamma$ is the smallest nearest-enemy distance in $P$. This implies $\alpha$-SFCNN can select up to $2^{\text{ddim}(\mathcal{X})+1}$ times more points as the $\alpha$-NET algorithm, which yields the proof.

**Theorem 5.17.** $\alpha$-SFCNN *selects* $\mathcal{O}\left(\kappa \log \frac{1}{\gamma}\ (1+\alpha)^{\text{ddim}(\mathcal{X})+1}\right)$ *points.*

Similarly, this result can be proven using the same arguments outlined to prove Theorem 5.11. After partitioning the selection of $\alpha$-SFCNN into $\mathcal{O}(\kappa \log 1/\gamma)$ subsets, consider any two points $a, b$ in one of these subsets, where $\mathsf{d}_{\text{ne}}(a), \mathsf{d}_{\text{ne}}(b) \in [\sigma, 2\sigma)$, for some $\sigma \in [\gamma, 1]$. Therefore, we can show that $\mathsf{d}(a, b) \geq \sigma/(1+\alpha)$, which implies that each subset in the partitioning contains at most $\lceil 4(1+\alpha) \rceil^{\text{ddim}(\mathcal{X})+1}$ points. This yields the proof.

## 5.4 Experimental Comparison

In order to get a clearer impression of the relevance of these results in practice, we performed experimental trials on several training sets, both synthetically generated and widely used benchmarks. First, we consider 21 training sets from the UCI *Machine Learning Repository*[3] which are commonly used in the literature to evaluate condensation algorithms [18]. These consist of a number of points ranging from 150 to 58000, in $d$-dimensional Euclidean space with $d$ between 2 and 64, and 2 to 26 classes. We also generated some synthetic training sets, containing $10^5$ uniformly distributed points, in 2 to 3 dimensions, and 3 classes. All training sets used in these experimental trials are summarized in Table 4.2. The implementation of the algorithms, training sets used, and raw results, are publicly available[4].

These experimental trials compare the performance of different condensation algorithms when applied to vastly different training sets. We use two measures of comparison on these algorithms: their runtime in the different training sets, and the size of the subset selected. Clearly, these values might differ greatly on training sets whose size are too distinct. Therefore, before comparing the raw results, these are normalized. The runtime of an algorithm for a given training set is normalized by dividing it by $n$, the size of the training set. The size of the selected subset is normalized by dividing it by $\kappa$, the number of nearest-enemy points in the training set, which characterizes the complexity of the boundaries between classes.

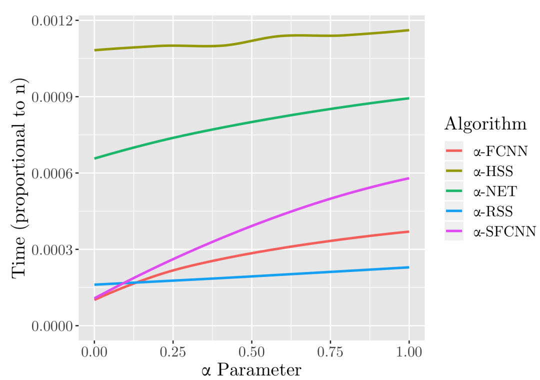
[3] https://archive.ics.uci.edu/ml/index.php

[4] https://github.com/afloresv/nnc/

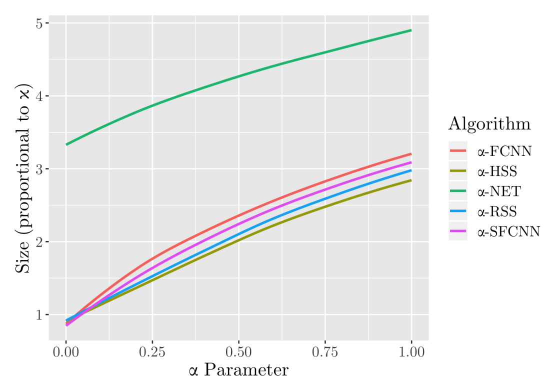
### 5.4.1 Algorithm Comparison

The first experiment evaluates the performance of the five algorithms discussed in this chapter: $\alpha$-RSS, $\alpha$-FCNN, $\alpha$-SFCNN, $\alpha$-HSS, and $\alpha$-NET. The evaluation is carried out by varying the value of the $\alpha$ parameter from 0 to 1, to understand the impact of increasing this parameter. The implementation of $\alpha$-HSS uses the well-known greedy algorithm for set cover [71], and solves the problem using the reduction described in Section 5.3.1. In the other hand, recall that the original NET algorithm (for $\alpha = 0$) implements an extra pruning technique to further reduce the training set after computing the $\gamma$-net [36]. For a fair comparison, we implemented the $\alpha$-NET algorithm with a modified version of this pruning technique that guarantees that the selected subset is still $\alpha$-selective.

The results show that $\alpha$-RSS outperforms the other algorithms in terms of running time by a big margin, and irrespective of the value of $\alpha$ (see Figure 5.6a). Additionally, the number of points selected by $\alpha$-RSS, $\alpha$-FCNN, and $\alpha$-SFCNN is comparable to $\alpha$-HSS, which guarantees the best possible approximation factor in general metrics, while $\alpha$-NET is significantly outperformed.




(a) Running time.

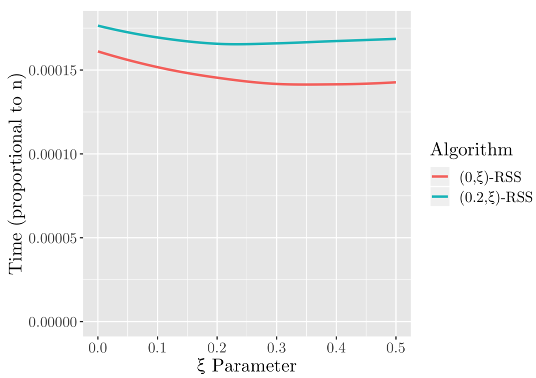



(b) Size of the selected subsets.

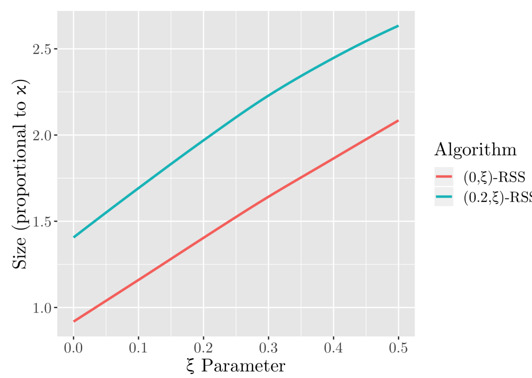
Figure 5.6: Comparison $\alpha$-RSS, $\alpha$-FCNN, $\alpha$-SFCNN, $\alpha$-NET, and $\alpha$-HSS, for different values of $\alpha$.

### 5.4.2 Subquadratic Approach

Using the same experimental framework, we evaluate performance of the subquadratic implementation $(\alpha, \xi)$-RSS described in Section 5.3.2.3. In this case, we change the value of parameter $\xi$ to assess its effect on the running time and selection size over the algorithm, for two different values of $\alpha$ (see Figure 5.7). The results show an expected increase of the number of selected points, while significantly improving its running time.




(a) Running time.




(b) Size of the selected subsets.

Figure 5.7: Evaluating the effect of increasing the parameter $\xi$ on $(\alpha, \xi)$-RSS for $\alpha = \{0, 0.2\}$.

# Chapter 6: Chromatic Approximate Voronoi Diagram

## 6.1 Introduction

As evidenced from the previous chapters, a common approach towards dealing with nearest-neighbor classification, is to reduce this problem to the one of nearest-neighbor search. That is, taking the initial training set $P$ and using any out-of-the-box solution for computing nearest-neighbors of $P$, like AVDs [19, 20], LSH [21], and HNSW graphs [22]. Assuming this reduction for nearest-neighbor classification, there are only two ways of having more efficient queries. The first, by preprocessing the training set via some condensation algorithm or by computing an $\varepsilon$-coresets, as described in Chapters 3 to 5. The second option is to improve the complexity of nearest-neighbor search techniques (usually involving approximate solutions), which is a known and extensive line of research.

In this chapter we explore an alternative approach towards efficient nearest-neighbor classification. The idea then is to avoid reducing this problem to the one of nearest-neighbor search, but instead proposing an approach that directly computes the predicted class. Therefore, our approach would bypass the preprocessing of the training set, and build a tailor-made data structure for approximate nearest-neighbor classification. Formally, we are given training set $P$ in a metric space $(\mathcal{X}, \mathsf{d})$ and an approximation parameter $0 < \varepsilon \leq \frac{1}{2}$, and the goal is to construct a data structure so that given any query point $q \in \mathcal{X}$, it is possible to efficiently classify $q$ according to any valid $\varepsilon$-approximate nearest-neighbor of $q$ in $P$. Throughout, we take domain $\mathcal{X}$ to be the $d$-dimensional Euclidean space $\mathbb{R}^d$, distance function $\mathsf{d}$ to be the $L_2$ norm, and we assume that the dimension $d$ is a fixed constant, independent of $n$ and $\varepsilon$.

These results have been published in [78].

## Related Work

When working with *$\varepsilon$-approximate nearest-neighbor searching* ($\varepsilon$-ANN), the objective is to compute a point whose distance from the query point is within a factor of $1 + \varepsilon$ of the true nearest-neighbor. This problem, referred to as "standard ANN" throughout this chapter, has been widely studied. In *chromatic $\varepsilon$-ANN search* the objective is to return just the class (or more visually, the "color") of any such point [51]. We refer to this as *$\varepsilon$-classification*, and it is the focus of this chapter.

Clearly, chromatic ANN queries can be reduced to standard ANN queries. Hence, most of the efficiency improvements in nearest-neighbor classification have arisen from improvements to the standard ANN problem. While standard ANN has been well studied in high-dimensional spaces (see, e.g., [21, 22, 50]), in constant-dimensional Euclidean space, the most efficient data structures involve variants of the *Approximate Voronoi Diagram* (or AVD) (see [19, 20, 43, 75]). Mount *et al.* [51] proposed a data structure specifically tailored for $\varepsilon$-classification. Unfortunately, this work was based on older technology, and its results are not competitive when compared to the most recent advances on standard ANN search via AVDs.

All previous results have query and space complexities that depend on $n$, the total size of the training set $P$. In many cases, a much smaller portion of the training set may suffice to correctly $\varepsilon$-classify queries. In the exact setting, these smaller set of points would be the set of border points of $P$, of size $k$, which are the ones that define the boundaries between classes. Moreover, the concept of border points can be generalized in the context of $\varepsilon$-classification (see Section 6.2 for a formal definition). Thus, denote $k_\varepsilon$ as the number of $\varepsilon$-border points, where $k \leq k_\varepsilon \leq n$. Ideally, we would like the query and space complexities of answering chromatic $\varepsilon$-ANN queries to depend on $k_\varepsilon$ instead of $n$.

There are different approaches to achieve this goal via training set reduction, as described in Chapters 3 to 5. In general, the idea in such case would be to select a subset $R \subseteq P$, which is then used to build a standard AVD to answer $\varepsilon$-ANN queries over $R$. However, depending on the approach, one obtains different subset sizes and classification guarantees. From the condensation heuristics described in Chapter 4, we know that it is possible to compute subsets $R$ of size $\mathcal{O}(k)$ in $\mathcal{O}(n^2)$ time. However, when AVDs are built from these subsets, the resulting data structures are likely to introduce classification errors, especially for query points that should be easily $\varepsilon$-classified (as described in Chapter 5). Thus, while often used in practice, these approaches do not guarantee that chromatic $\varepsilon$-ANN queries are answered correctly. The second approach would involve computing a coreset for $\varepsilon$-classification, as defined in Chapter 5. Recall that a coreset $R$ guarantees that every query point will be correctly classified when assigning the class of the point of $R$ returned by the AVD. That is, for any query $q \in \mathbb{R}^d$, the point of $R$ returned by the AVD belongs to the same class as one of $q$'s $\varepsilon$-approximate nearest-neighbors in $P$. Unfortunately, the size of the described coreset can be as large as $\mathcal{O}((k \log \Delta)/\varepsilon^{d-1})$, where $\Delta$ is the spread of $P$.

## Contributions

From the previous section, we have seen that existing approaches for $\varepsilon$-classification achieve only one of the following goals:

- The size of the resulting data structure is dependent only on $\varepsilon$, $k_\varepsilon$ (the number of $\varepsilon$-border points) and $d$, while being independent from $n$ and $\Delta$.
- It guarantees correct $\varepsilon$-classification for any query point.

The main result of this chapter is an approach that achieves both goals. We propose a new data structure built specifically to answer chromatic $\varepsilon$-ANN queries over the training set $P$, which we call a *Chromatic AVD*. Given any query point

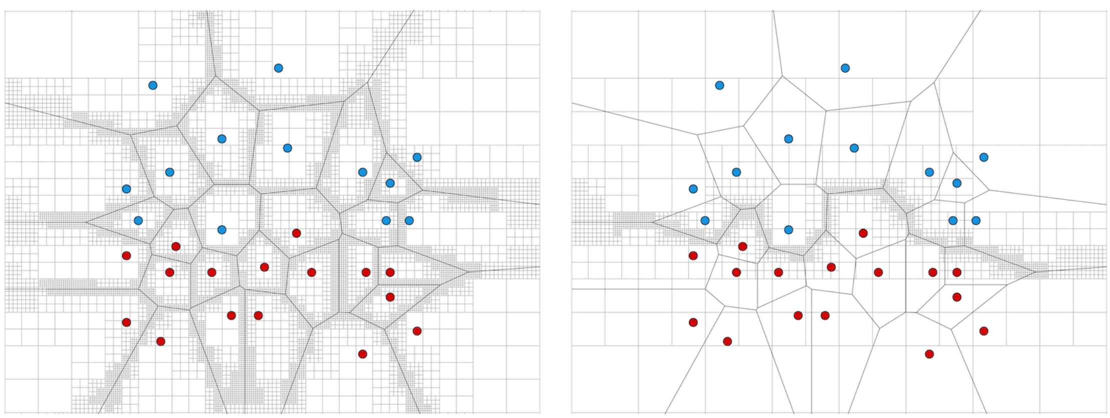
$q \in \mathbb{R}^d$, this data structure returns the class to be assigned to $q$, which matches the class of at least one of $q$'s $\varepsilon$-approximate nearest-neighbors in $P$. More generally, our data structure returns a set of classes such that there is an $\varepsilon$-approximate nearest-neighbor of $q$ from each of these classes.

Therefore, the Chromatic AVD can be used to correctly $\varepsilon$-classify any query point. The main result of this work is summarized in the following theorem, expressed in the form of a space-time tradeoff based on a parameter $\gamma$.

**Theorem 6.1.** *Given a training set $P$ of $n$ labeled points in $\mathbb{R}^d$, an error parameter $0 < \varepsilon \leq \frac{1}{2}$, and a separation parameter $2 \leq \gamma \leq \frac{1}{\varepsilon}$. Let $k_\varepsilon$ be the number of $\varepsilon$-border points of $P$. There exists a data structure for $\varepsilon$-classification, called Chromatic AVD, with:*

$$\textit{Query time: } \mathcal{O}\left(\log(k_\varepsilon \gamma) + \frac{1}{(\varepsilon\gamma)^{\frac{d-1}{2}}}\right) \qquad \textit{Space: } \mathcal{O}\left(k_\varepsilon \gamma^d \log \frac{1}{\varepsilon}\right).$$

*Which can be constructed in time $\widetilde{\mathcal{O}}\left(\left(n + k_\varepsilon/(\varepsilon\gamma)^{\frac{3}{2}(d-1)}\right)\gamma^d \log \frac{1}{\varepsilon}\right)$.*



(a) Standard AVD [19, 20, 43, 75].  (b) Chromatic AVD.

Figure 6.1: Examples of the space partitioning achieved by any standard AVD, compared to the Chromatic AVD data structure proposed in this chapter. Our approach subdivides the space around the boundaries defined by the $\varepsilon$-border points, while ignoring other boundaries.

By setting $\gamma$ to either of its extreme values, we obtain the following query times and space complexities.

**Corollary 6.2.** *The separation parameter $\gamma$ describes the tradeoffs between the query time and space complexity of the Chromatic AVD. This yields the following results:*

$$\textit{If } \gamma = 2 \quad \longrightarrow \quad \textit{Query time: } \mathcal{O}\left(\log k_\varepsilon + \frac{1}{\varepsilon^{\frac{d-1}{2}}}\right) \qquad \textit{Space: } \mathcal{O}\left(k_\varepsilon \log \frac{1}{\varepsilon}\right).$$

$$\textit{If } \gamma = \frac{1}{\varepsilon} \quad \longrightarrow \quad \textit{Query time: } \mathcal{O}\left(\log \frac{k_\varepsilon}{\varepsilon}\right) \qquad \textit{Space: } \mathcal{O}\left(\frac{k_\varepsilon}{\varepsilon^d}\right).$$

The approach towards constructing this data structure is hybrid, combining a quadtree-induced partitioning of space (leveraging similar techniques to the ones used for standard AVDs), with the construction of coresets for only some cells of this partition. All other cells can be discarded, and a new quadtree can be built with only the remaining cells. The final size of the tree is bounded in terms of $k_\varepsilon$. This technique allows us to maintain coresets in the most critical regions of space, and thus, avoiding the dependency on the spread of $P$.

## 6.2 Preliminary Ideas and Intuition

First, we need to introduce some preliminary definitions and notations that are relevant to the results presented in the remaining of the chapter. Given any point $q \in \mathbb{R}^d$, denote its nearest-neighbor as $\mathrm{nn}(q)$, and the distance between them by $\mathsf{d}_{\mathrm{nn}}(q) = \mathsf{d}(q, \mathrm{nn}(q))$.

Additionally, let's introduce a few concepts and related properties that will prove useful in the construction of the Chromatic AVD. These are Well-Separated Pair Decompositions [79] (WSDPs), Quadtrees [48, 80], and Approximate Voronoi Diagrams [19, 20, 43, 75] (AVDs).

Well-Separated Pair Decompositions: Given the point set $P$, and a separation factor $\sigma > 2$, we say that two sets $X, Y \subseteq P$ are *well separated* if they can be enclosed within two disjoint balls of radius $r$, such that the distance between the centers of these balls is at least $\sigma r$. We say that $X$ and $Y$ form a *dumbbell*, where both sets are the *heads* of this dumbbell. Consider the line segment that connects the centers of both balls, and let $z$ and $\ell$ be the center and length of this line segment, respectively (*i.e.*, the center and the length of the dumbbell). The following properties hold when $\sigma > 4$, for $x \in X$ and $y \in Y$:

$$\mathsf{d}(x, z) < \ell \qquad \ell < 2\mathsf{d}(x, y) \qquad \ell > \mathsf{d}(x, y)/2.$$

Furthermore, a well-separated pair decomposition of $P$ is defined as a set $\mathcal{D} = \{(X_i, Y_i)\}_i$ where every $X_i$ and $Y_i$ are well separated, and for every two distinct points $p_1, p_2 \in P$ there exists a unique pair $\mathcal{P} = (X, Y) \in \mathcal{D}$ such that $p_1 \in X$ and $p_2 \in Y$, or vice-versa. It is known how to construct a WSPD of $P$ with $\mathcal{O}(\sigma^d n)$ pairs in $\mathcal{O}(n \log n + \sigma^d n)$ time.

Quadtrees: These are tree data structures that provide a hierarchical partition of space. Each node in this tree consists of a $d$-dimensional hypercube, where non-leaf nodes partition its corresponding hypercube into $2^d$ equal parts. The root of this tree corresponds to the $[0, 1]^d$ hypercube. We will use a variant of this structure called a *balanced box-decomposition* tree (BBD tree) [42]. Such data structure satisfies the following properties:

1. Given a point set $P$, such a tree can be built in $\mathcal{O}(n \log n)$ time, having space $\mathcal{O}(n)$ such that each leaf node contains at most one point of $P$.

2. Given a collection $\mathcal{U}$ of $n$ quadtree boxes in $[0,1]^d$, such a tree can be built in $\mathcal{O}(n \log n)$ time, having $\mathcal{O}(n)$ nodes such that the subdivision induced by its leaf cells is a refinement of the subdivision induced by the Quadtree boxes in $\mathcal{U}$.

3. Given the trees from **1** or **2**, it is possible to determine the leaf cell containing any arbitrary query point $q$ in $\mathcal{O}(\log n)$ time.

Approximate Voronoi Diagram: Generally, AVDs are quadtree-based data structures that can be used to efficiently answer ANN queries. The partitioning of space induced by this data structure is often generated from a WSPD of $P$. Additionally, every leaf cell $w$ of this quadtree has an associated set of $\varepsilon$-representatives $R_w$ that has the following property: for any query point $q \in w$, at least one point in $R_w$ is one of $q$'s $\varepsilon$-approximate nearest-neighbors in $P$.

New Ideas and Intuitions. Consider the space partitioning induced by a standard AVD, as previously described. By construction, any leaf cell $w$ of this partition has an associated set of $\varepsilon$-representatives $R_w$. Evidently, for the purposes of $\varepsilon$-classification, the most important information related to this leaf cell comes from the classes of the points in $R_w$, and not necessarily the points themselves.

This leads to an initial approach to simplify an AVD. We distinguish between two types of leaf cells, based on the points inside their corresponding $\varepsilon$-representative sets. Any leaf cell $w$ is said to be:

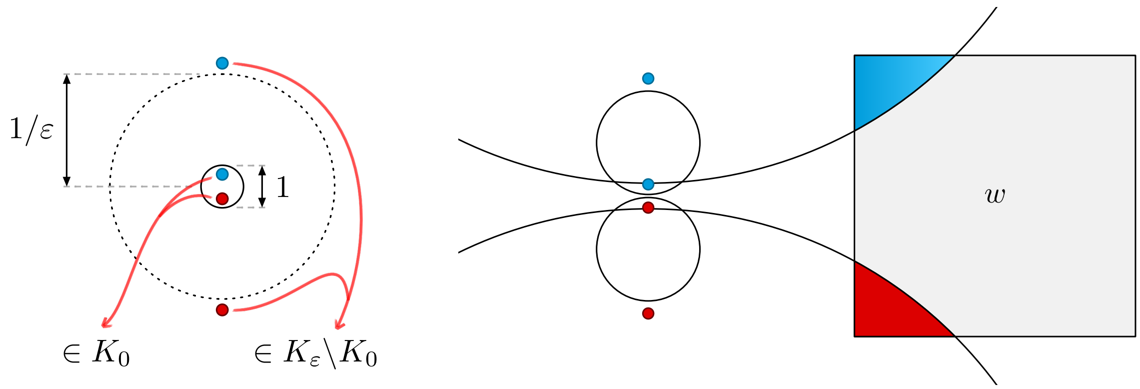
- **Resolved**: If every point in $R_w$ belongs to the same class.
- **Ambiguous**: Otherwise, if at least two points in $R_w$ belong to different classes.

Clearly, there is no need to store the set of $\varepsilon$-representatives of any resolved leaf cell, as instead, we can simply mark the leaf cell $w$ as *resolved with the class* that is shared by all the points in $R_w$. This effectively reduces the space needed for such cells to be constant.

Furthermore, it seems that the bulk of the "work" needed to decide the class of a given query point can be carried out by the ambiguous leaf cells, along with some groupings of resolved leaf cells. The data structure presented in this chapter, called Chromatic AVD, builds upon this hypothesis.

Additionally, we formally define the set of $\varepsilon$-border points of the training set $P$. This set, denoted as $K_\varepsilon$, contains any point $p \in P$ for which there exist some $q \in \mathbb{R}^d$ and $\bar{p} \in P$, such that $p$ and $\bar{p}$ are $\varepsilon$-approximate nearest-neighbors of $q$, and both belong to different classes. Denote $k_\varepsilon = |K_\varepsilon|$ as the number of $\varepsilon$-border points of the training set $P$. Note that $K_\varepsilon \subseteq K_{\varepsilon'}$ if and only if $\varepsilon \leq \varepsilon'$. Additionally, note that $K_0$ defines the set of (exact) border points of $P$, where $k = k_0$.

This generalization of the definition of border points seems better suited to analyze the problem of $\varepsilon$-classification, as illustrated in Figure 6.2. Figure 6.2b shows the $\varepsilon$-approximate bisectors between the two closest and two farthest points (the first two belong to $K_0$, while the others belong to $K_\varepsilon$ but not $K_0$). A hypothetical leaf cell $w$ is sufficiently separated from the only two exact border points, but intersects




(a) (b)

Figure 6.2: Intuition to assume that $K_\varepsilon$ (and not $K_0$) is needed to $\varepsilon$-classify some query points.

the $\varepsilon$-approximate bisectors between the two farthest points. This implies that inside the cell $w$ lie query points that *can only* be $\varepsilon$-classified with one class, and others with the other class, forcing this cell to be ambiguous. This suggests that $K_0$ is insufficient to account for the necessary complexity of $\varepsilon$-classification.

## 6.3 Chromatic AVD Construction

In this section, we describe our method for constructing the proposed Chromatic AVD. The following overview outlines the necessary steps followed to construct this data structure.

- *The Build step* (Section 6.3.1): Consists of building an initial quadtree-based subdivision of space, designed specifically to achieve the properties described in Lemma 6.3.
- *The Reduce step* (Section 6.3.2): Seeks to identify the leaf cells of the initial subdivision that are relevant for $\varepsilon$-classification, as well as those that can be ignored or simplified. This process consists of the following substeps.
  - Computing the sets of $\varepsilon$-representatives for every leaf cell of the initial quadtree.
  - Based on these sets, marking the leaf cells as either ambiguous or resolved.
  - Selecting those leaf cells which are relevant for $\varepsilon$-classification.
  - Building a new quadtree-based subdivision using the previously selected leaf cells.

### 6.3.1 The Build Step

We begin by constructing the tree $T_{\text{init}}$ using similar methods as the ones used to construct a standard AVD. Thus, the first step is to compute a well-separated pair decomposition $\mathcal{D}$ of $P$ using a constant separation factor of $\sigma > 4$. While the standard construction would use all pairs in this decomposition, for the purpose of the Chromatic AVD, we filter $\mathcal{D}$ to only keep bichromatic pairs. Denote $\mathcal{D}' \subseteq \mathcal{D}$ to be the set of bichromatic pairs in $\mathcal{D}$, where a pair $\mathcal{P} \in \mathcal{D}$ is said to be bichromatic if and only if the dumbbell heads separate points of different classes. Note that $\mathcal{D}'$ can be computed similarly to $\mathcal{D}$, using a simple modification of the well-known algorithm for computing WSPDs [79] (the details are left to the reader).

Next, we compute an initial set of quadtree boxes $\mathcal{U}(\mathcal{P})$ for every pair in $\mathcal{D}'$ as follows. This construction depends on two constants $c_1$ and $c_2$ whose assignment will be described later in this section. For $0 \leq i \leq \lceil \log (c_1\, 1/\varepsilon) \rceil$, we define $b_i(\mathcal{P})$ as the ball centered at $z$ of radius $r_i = 2^i \ell$. Thus, this set of balls involves radius values ranging from $\ell$ to $\Theta(\ell/\varepsilon)$. For each such ball $b_i(\mathcal{P})$, let $\mathcal{U}_i(\mathcal{P})$ be the set of quadtree boxes of size $r_i/(c_2\gamma)$ that overlap the ball. Let $\mathcal{U}(\mathcal{P})$ denote the union of all these boxes over all the $\mathcal{O}(\log 1/\varepsilon)$ values of $i$.

After performing this process on every pair of the filtered decomposition $\mathcal{D}'$, take the union of all these boxes denoted as $\mathcal{U} = \bigcup_{\mathcal{P} \in \mathcal{D}'} \mathcal{U}(\mathcal{P})$. Finally, build the tree $T_{\text{init}}$ from the set of quadtree boxes $\mathcal{U}$, leveraging property 2 of quadtrees described in Section 6.2. Additionally, for each class $i$ in the training set, build an auxiliary tree $T^i_{\text{aux}}$ from the point set $P_i$ (*i.e.*, the points of $P$ of that are labeled with class $i$), using property 1 of quadtrees. These auxiliary trees will be used together with $T_{\text{init}}$ in order to build our final tree $T$, the Chromatic AVD.

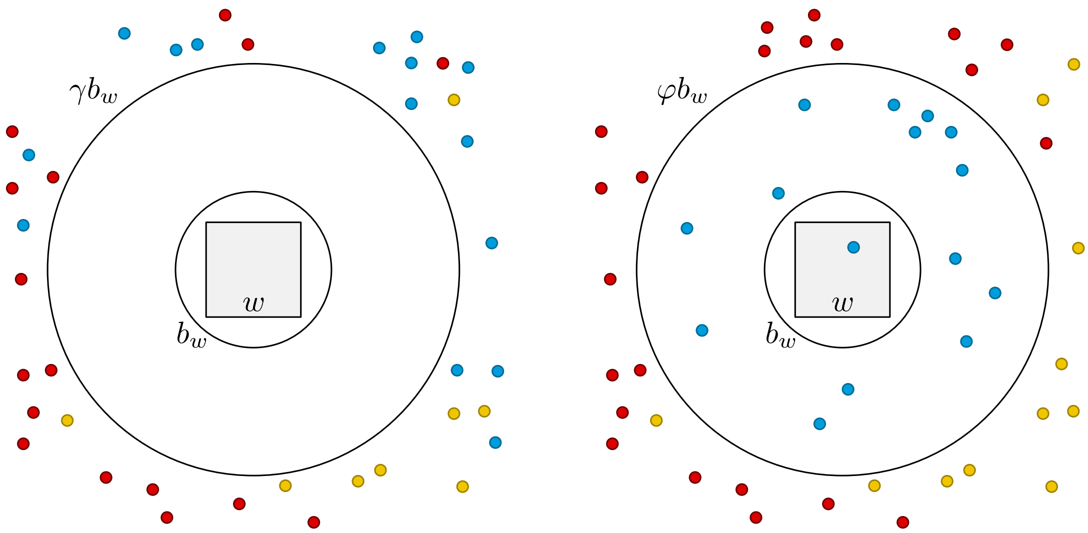
While the standard AVD construction satisfies that all resulting leaf cells of the tree have certain separation properties from the points of set $P$, the same is not true for tree $T_{\text{init}}$. However, the following result describes a relaxed notion of the separation properties, now based on the classes of the points, which are achieved by $T_{\text{init}}$.

**Lemma 6.3** (Chromatic Separation Properties)**.** *Given two separation parameters $\gamma > 2$ and $\varphi > 3$, every leaf cell $w$ of the tree $T_{\text{init}}$ satisfies at least one of the following separation properties, where $b_w$ is the minimum enclosing ball of $w$:*

- *(i) $P \cap \gamma b_w$ is empty (see Figure 6.3a), and hence $b_w$ is concentrically $\gamma$-separated from $P$.*
- *(ii) The cell $w$ can be resolved with the classes present inside $P \cap \varphi b_w$ (see Figure 6.3b).*

*Proof.* Let $w$ be any leaf cell of $T_{\text{init}}$, with center $c_w$ and side length $s_w$, where its (minimum) enclosing ball $b_w$ has radius $r_w = \sqrt{d}/2\, s_w$ and shares the center $c_w$. Additionally, let $x_i \in P_i$ be a 1-approximate nearest-neighbor of $c_w$ among the points of $P$ of class $i$. In other words, for each class of points we use the auxiliary trees $T^i_{\text{aux}}$ to compute a 1-approximate nearest-neighbor of $c_w$. A few cases unfold from here:

The first case is rather simple. If $4\gamma\varphi b_w \cap \{x_i\}_i = \varnothing$, knowing that the points $x_i$ are 1-approximate nearest-neighbors of $c_w$, this implies that the ball $2\gamma\varphi b_w$ is empty (*i.e.*, we know that $2\gamma\varphi b_w \cap P = \varnothing$). Clearly, this means the the first separation property holds for $w$.




(a) No points inside $\gamma b_w$. (b) Leaf cell $w$ can be resolved.

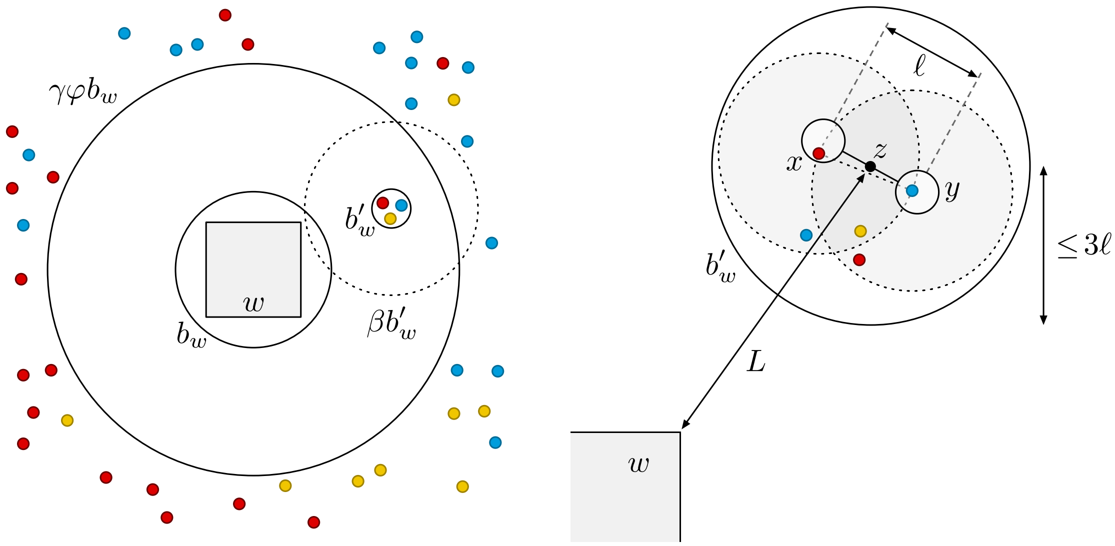
Figure 6.3: Basic separation properties achieved by during the construction of the Chromatic AVD.

Consider the case when $|4\gamma\varphi b_w \cap \{x_i\}_i| = 1$, and let $i$ be the class of the point that lies inside $4\gamma\varphi b_w$. Following similar arguments to the previous case, this implies that only points of class $i$ could potentially lie inside of $2\gamma\varphi b_w$. Then, check if $x_i$ lies inside the smaller ball expansion $2\gamma b_w$. If not, we know that $\gamma b_w$ is empty (*i.e.*, $\gamma b_w \cap P = \varnothing$), making the first separation property hold for $w$. Otherwise, we know that $2\gamma b_w$ contains at least one point (*i.e.*, $x_i$), and additionally we know that $2\gamma b_w$ is $\varphi$-separated from points of all other classes but $i$ (as $2\gamma\varphi b_w$ only contains points of class $i$). Given that $\varphi > 3$, the nearest neighbor of every query point inside $2\gamma b_w$ has class $i$. Therefore, $w$ can be resolved with class $i$ (namely, $C_w = \{i\}$), satisfying the second separation property.

Lastly, it is possible that $|4\gamma\varphi b_w \cap \{x_i\}_i| \geq 2$. However, it is possible to show that if this is the case, it immediately implies that every point inside $4\gamma\varphi b_w$ actually lies inside of some ball $b'_w$ which is $\beta$-separated from $w$ (see Figure 6.4a), where $\beta = 1/\varepsilon$. Let $x, y \in 4\gamma\varphi b_w$ be the two points of different classes inside the ball $4\gamma\varphi b_w$ with largest pairwise distance. Thus, it is easy to show that all the points inside $4\gamma\varphi b_w$ lie inside the two balls centered at $x$ and $y$ with radii equal to $\mathsf{d}(x, y)$, as shown in Figure 6.4b. By definition of the (bichromatic) well-separated pair decomposition $\mathcal{D}'$, there exists a pair $\mathcal{P} \in \mathcal{D}'$ that contains $x$ and $y$ each in one of its dumbbells, with length $\ell$ and center $z$. Now, we define the ball $b'_w$ with center at $z$ and radius $r'_w = \max(\mathsf{d}(z, x), \mathsf{d}(z, y)) + \mathsf{d}(x, y)$. By definition of $\mathcal{P}$, we know that $\mathsf{d}(z, x), \mathsf{d}(z, y) \leq \ell$ and $\mathsf{d}(x, y) \leq 2\ell$, thus making $r'_w \leq 3\ell$. Let $L$ be the distance from $w$ to $z$, we distinguish two cases based on the relationship between $L$ and $\ell$:

- $L > c_1\beta\ell$. Consider the distance that separates the ball $b'_w$ from the cell $w$.

$$\mathsf{d}(w, b'_w) = L - r'_w > c_1\beta\ell - r'_w \geq (c_1\beta/3 - 1)\, r'_w$$




(a) (b)

Figure 6.4: It is possible that points of $\geq 2$ classes lie inside of $\gamma\varphi b_w$. However, this case can be reduced to the two separation properties illustrated in Figure 6.3.

Since $\beta = 1/\varepsilon$, for all sufficiently large constants $c_1 \geq 3(1+\varepsilon)$, the distance $\mathsf{d}(w, b'_w)$ exceeds $\beta r'_w$ which implies that $b'_w$ is concentrically $\beta$-separated from $w$.

- $L \leq c_1\beta\ell$. We will show that this case cannot occur, since otherwise the dumbbell $\mathcal{P}$ would have caused $w$ to be split, contradicting the assumption that it is a leaf cell of $T_{\text{init}}$. Since $x, y$, and $w$ are all contained in the ball $4\gamma\varphi b_w$ whose center lies within $w$, we have both that $\mathsf{d}(x, w) \leq 4\gamma\varphi r_w$, and $\ell < 2\mathsf{d}(x, y) \leq 2(8\gamma\varphi r_w) = 16\gamma\varphi r_w$. Thus, by the triangle inequality, we have:

$$L \leq \mathsf{d}(x, y) + \mathsf{d}(x, w) < \ell + 4\gamma\varphi r_w < 16\gamma\varphi r_w + 4\gamma\varphi r_w = 20\gamma\varphi r_w$$

  Because $L \leq c_1\beta\ell$, it follows from our construction that there is at least one ball $b_i(\mathcal{P})$ (with $0 \leq i \leq \lceil \log c_1\beta \rceil$) that overlaps $w$. Let $b$ denote the smallest such ball, and let $r$ denote its radius. By the construction, we have that $r \leq \max(\ell, 2L)$. Since our construction generates all quadtree boxes of size $r/(c_2\gamma)$ that overlap $b$, it follows that $s_w \leq r/(c_2\gamma)$. Thus, we have:

$$r_w = s_w\frac{\sqrt{d}}{2} \leq \frac{r\sqrt{d}}{2c_2\gamma} \leq \frac{\max(\ell, 2L)\sqrt{d}}{2c_2\gamma} < \frac{20\gamma\varphi r_w\sqrt{d}}{c_2\gamma} = \frac{20\varphi r_w\sqrt{d}}{c_2}$$

  Choosing $c_2 \geq 20\varphi\sqrt{d}$ yields the desired contradiction.

Finally, if $|4\gamma\varphi b_w \cap \{x_i\}_i| \geq 2$, we know all points inside $4\gamma\varphi b_w$ are $\beta$-separated from $w$. We can now proceed similarly to the previous case, by checking if one of the computed 1-approximate nearest-neighbors lies inside the ball $2\gamma b_w$. If

$2\gamma b_w \cap \{x_i\}_i = \varnothing$ we know that $\gamma b_w$ is empty (*i.e.*, $\gamma b_w \cap P = \varnothing$), making the first separation property hold for $w$. Otherwise, note that $b'_w$ is completely contained inside $2\gamma(1+\varepsilon)b_w$. Given that $\varphi > 3$, it is possible to show that for any query point in $w$, all points in $b'_w$ are valid $\varepsilon$-approximate nearest neighbors. This implies that we can resolve $w$ with the class of any of the points inside of $b'_w$, thus satisfying the second separation property. In particular, we mark $w$ as resolved with every class present in the inner cluster $b'_w$, namely, $C_w = \{l(p) \mid \forall\, p \in b'_w \cap P\} = \{i \mid x_i \in 4\gamma\varphi b_w\}$. □

### 6.3.2 The Reduce Step

From the initial partitioning as described in Lemma 6.3, every leaf cell $w$ of $T_{\text{init}}$ is either concentrically $\gamma$-separated from $P$ (*i.e.*, $\gamma b_w \cap P = \varnothing$), or it is already marked as resolved. For every leaf cell $w$ in the first case, we will compute a set of $\varepsilon$-representatives by leveraging the *concentric ball lemma* (see Lemma 5.1 in [75]). It states that there exists a set $R_w$ of $\varepsilon$-representatives for $w$ of size $\mathcal{O}(1/(\varepsilon\gamma)^{\frac{d-1}{2}})$, and provides a way to compute such set.

Instead of directly applying this result, we use it to compute a set of $\varepsilon/3$-representatives for any leaf cell $w$ that is yet unresolved. Essentially, this leads to the same asymptotic upper-bound on the size of $R_w$, meaning that $|R_w| = \mathcal{O}(1/(\varepsilon\gamma)^{\frac{d-1}{2}})$. Once $R_w$ is computed, we can proceed to mark $w$ as either resolved or ambiguous as follows.

*Procedure to Mark Leaf Cells:* For every leaf cell $w$, this procedure marks $w$ as either resolved or ambiguous, following a few defined cases that unfold from the contents of the set $R_w$ of points selected as representatives for $w$. Let $r^-_w = \varepsilon\,(1-\gamma)\,r_w/3$.

1. If all the points in $R_w$ belong to the same class.
   For every point $p \in R_w$ and class $i \in C$, compute a 1-approximate nearest-neighbor of $p$ among the points of $P_i$, denoted as the point $x_{p,i}$. If $\mathsf{d}(p, x_{p,i}) < r^-_w$, then add $x_{p,i}$ to $R_w$. It is easy to show that $x_{p,i}$ would also be an $\varepsilon$-representative for $w$. Repeat this for every point originally in $R_w$, and every class in the training set.
   (a) If any point $x_{p,i}$ was added to $R_w$, proceed with *Case* 2.
   (b) Otherwise, mark $w$ as resolved with the class of the points in $R_w$. Namely, let $i$ be the class of every point in $R_w$, then $C_w = \{i\}$.
2. If $R_w$ contains points of more than one class.
   Before proceeding, we will do some basic pruning of the set $R_w$. For every class $i$, compute a *net* among the points of $R_w$ of class $i$, using a radius of $r^-_w$ to compute the net, and replace the points of class $i$ in $R_w$ with the computed net. It is easy to see that the remaining points of $R_w$ are a set of $\varepsilon$-representatives of $w$, and that every two points in $R_w$ of the same class are at distance at least $r^-_w$.

(a) If the diameter of $R_w$ is less than $r_w^-$, it is easy to prove that all the points in $R_w$ are $\varepsilon$-representatives of any point inside $b_w$. Therefore, $w$ can be labeled as resolved with the class of all of the points in $R_w$. That is, $C_w = \{l(p) \mid \forall\ p \in R_w\}$.

(b) If the diameter is greater than or equal to $r_w^-$, $w$ is marked as ambiguous.

Let $\mathcal{A}$ and $\mathcal{R}$ be the sets of ambiguous and resolved leaf cells of $T_{\text{init}}$, respectively. We will use some of these cells to build the Chromatic AVD, while ignoring the remaining cells.

Consider the set of resolved leaf cells $\mathcal{R}$, we partition this set into two subsets $\mathcal{R}_b$ and $\mathcal{R}_i$ (named *boundary* and *interior* resolved leaf cells, respectively). We say a resolved leaf cell $w_1$ belongs to $\mathcal{R}_b$, if and only if there exists another resolved leaf cell $w_2$ adjacent to $w_1$, such that $C_{w_2} \setminus C_{w_1} \neq \varnothing$. Every other resolved leaf cell belongs to $\mathcal{R}_i$ (*i.e.*, $\mathcal{R}_i = \mathcal{R} \setminus \mathcal{R}_b$). Note that both sets $\mathcal{R}_b$ and $\mathcal{R}_i$ can be identified by a simple traversal over the leaf cells of $T_{\text{init}}$, using linear time in the size of the tree[1].

Finally, we build a new tree $T$ from the set of ambiguous and boundary resolved leaf cells $\mathcal{A} \cup \mathcal{R}_b$. By well-known construction methods of quadtrees, as described in Section 6.2, the leaf cells of $T$ either belong to $\mathcal{A} \cup \mathcal{R}_b$, or are "Steiner" leaf cells added during the construction of $T$ that cover the remainer of the space that is uncovered by $\mathcal{A} \cup \mathcal{R}_b$.

**Lemma 6.4.** *For any leaf cell $w$ in the tree $T$ such that $w \notin \mathcal{A} \cup \mathcal{R}_b$, $w$ must cover a space that is also covered by a collection of leaf cells of $T_{\text{init}}$, all of which are resolved with the same set of classes $C_w$.*

*Proof.* This becomes apparent from the construction of $T$. In the new tree $T$, consider any leaf cell $w$ of $T$ that is not part of $\mathcal{A} \cup \mathcal{R}_b$ (*i.e.*, a "Steiner" leaf cell added during the construction of the tree). Now, recall that the leaf cells of both $T$ and $T_{\text{init}}$ are a partitioning of (the same) space, which means that we can define $\mathcal{W}_w = \{w' \in T_{\text{init}} \mid w \cap w' \neq \varnothing\}$ as the collection of leaf cells of $T_{\text{init}}$ that cover the same space covered by $w$.

Now, for any fixed $w$ of $T$, it is easy to see that any two leaf cells $w_1, w_2 \in \mathcal{W}_w$ must be resolved with the same set of classes. Otherwise, at least one of these two would be part of the set $\mathcal{R}_b$, which would be a contradiction to the fact that $w$ is a "Steiner" leaf cell of $T$. Therefore, any query inside $w$ can be answered with the classes $C_w = C_{w_1} = C_{w_2}$, and this can be determined during preprocessing by a single query on $T_{\text{init}}$ (*e.g.*, finding the leaf cell of $T_{\text{init}}$ that contains the center $c_w$ of $w$ is sufficient to know the contents of $C_w$). $\square$

This implies that after building tree $T$, and with some extra preprocessing to resolve the "Steiner" leaf cells of the tree, we can use the resulting data structure to correctly answer chromatic $\varepsilon$-approximate nearest-neighbor queries over the training

[1] Two leaf cells are adjacent if and only if a vertex of one of the cells "touches" the other cell. This implies that the number of adjacency relations (*i.e.*, edges in the implicit graph where the leaf cells are the nodes) is $\mathcal{O}(2^d\, m)$, where $m$ is the number of leaf cells of the tree $T_{\text{init}}$.

set $P$. In other words, $T$ can be used to answer $\varepsilon$-classification queries over $P$. We call this data structure $T$ the Chromatic AVD.

**Lemma 6.5.** *The construction of $T$ takes $\widetilde{\mathcal{O}}\left(n\gamma^d \log \frac{1}{\varepsilon}\right)$ time.*

*Proof.* Let's analyze the total time needed to build our Chromatic AVD, namely the tree $T$, by analyzing the time required to perform each step of the construction.

- Building $T_{\text{init}}$ has essentially the same complexity of building any standard AVD [19, 20, 43, 75]. This means that constructing $T_{\text{init}}$ takes $\mathcal{O}(m \log m)$ time, where $m = n\gamma^d \log \frac{1}{\varepsilon}$. Note that during the construction, while computing the set of $\varepsilon$-representatives of each leaf cell, each leaf cell can already be marked as either ambiguous or resolved.

- Building the auxiliary trees $T^i_{\text{aux}}$ for every class $i$, takes $\mathcal{O}(n \log n)$ time, as the number of classes of $P$ is considered to be a constant. Recall that because these trees are only used to for 1-ANN queries, they only need to be standard Quadtrees, and not AVDs.

- Identifying the set $\mathcal{R}_b$ requires a traversal over the leaf-level partitioning of the space, which is linear in terms of the number of cells. Therefore, this step requires $\mathcal{O}(m)$ time.

- Once the sets of ambiguous and boundary resolved leaf cells are identified, namely, the sets $\mathcal{A}$ and $\mathcal{R}_b$, the final tree $T$ can be built. Roughly, this step takes $\mathcal{O}(m \log m)$ time.

- Finally, we must resolve each "Steiner" leaf cell of $T$, which can be done by a single query over $T_{\text{init}}$, each taking $\mathcal{O}(\log m)$ time. Thus, this step takes $\mathcal{O}\left(m \log m\right)$ total time.

All together, the total construction time is dominated by the first step. Therefore, the time required to construct $T$ is $\mathcal{O}(m \log m) = \widetilde{\mathcal{O}}(m) = \widetilde{\mathcal{O}}\left(n\gamma^d \log \frac{1}{\varepsilon}\right)$. □

## 6.4 Tree-size Analysis

### 6.4.1 Initial Size Bounds

Define the set of important leaf cells $\mathcal{I}$ of the tree $T_{\text{init}}$ as those leaf cells $w$ for which there exists two $\varepsilon$-border points inside $\rho\gamma b_w$ for some constant $\rho$, such that the distance between these points is lower-bounded by $\Omega(\varepsilon\gamma r_w)$. Formally, we define this set as $\mathcal{I} = \{w \in T_{\text{init}} \mid \exists p_1, p_2 \in \rho\gamma b_w \cap K_\varepsilon, \mathsf{d}(p_1, p_2) = \Omega(\varepsilon\gamma r_w)\}$.

**Lemma 6.6.** *The number of important leaf cells of $T_{\text{init}}$ is $|\mathcal{I}| = \mathcal{O}\big(k_\varepsilon \gamma^d \log \frac{1}{\varepsilon}\big)$.*

*Proof.* This proof follows from a charging argument on the set $K_\varepsilon$ of $\varepsilon$-border points of $P$. More specifically, consider a well-separated pair decomposition $\mathcal{D}''$ of $K_\varepsilon$ with constant separation factor of $\sigma > 4$, the charging scheme assigns every important leaf cell $w \in \mathcal{I}$ to a pair of $\mathcal{D}''$. Recall that $\mathcal{D}''$ can generally be consider to have $\mathcal{O}(k_\varepsilon)$ pairs, where $k_\varepsilon = |K_\varepsilon|$. It is important to note that both $K_\varepsilon$ and $\mathcal{D}''$ need not be computed.

Given that $w \in \mathcal{I}$, we know there exist two points $p_1, p_2 \in \rho\gamma b_w \cap K_\varepsilon$. Let $\mathcal{P} \in \mathcal{D}''$ be the pair of $\mathcal{D}''$ that contains both $p_1$ and $p_2$, each in one of its dumbbell heads. We then charge $w$ to the pair $\mathcal{P}$. Denote $z$ and $\ell$ to be the center and length of $\mathcal{P}$, respectively, we know the following. First, note that the distance from $c_w$ (the center of $w$) to $z$ is $\mathsf{d}(c_w, z) \leq \rho\gamma r_w + \ell$. Additionally, we know that $\ell \geq \mathsf{d}(p_1, p_2)/2 = \Omega(\varepsilon\gamma r_w)$ by the properties of WSPDs described in Section 6.2. Therefore, this implies that the ratio $\mathsf{d}(c_w, z)/\ell = \mathcal{O}(1/\varepsilon)$.

Finally, fix some pair $\mathcal{P} \in \mathcal{D}''$ with center $z$ and length $\ell$, and consider all important leaf cells according to their distance to $z$. For any value of $i \in [0, 1, \ldots, \mathcal{O}(\log 1/\varepsilon)]$, consider all leaf cells that can charge $\mathcal{P}$ whose distance to $z$ is between $2^i\ell$ and $2^{i+1}\ell$. From our previous analysis, $r_w \geq \mathsf{d}(c_w, z)/\rho\gamma \geq 2^i\ell/\rho\gamma$. By a simple packing argument, the number of such leaf cells is at most $\mathcal{O}(\gamma^d)$. Thus, a total of $\mathcal{O}(\gamma^d \log 1/\varepsilon)$ cells can charge $\mathcal{P}$. Note that no leaf cell whose distance to $z$ is $\Omega(\ell/\varepsilon)$ can charge $\mathcal{P}$, as it would contradict the fact that both $p_1$ and $p_2$ are separated by a distance of $\Omega(\varepsilon\gamma r_w)$. Finally, the proof follows by knowing that there are at most $\mathcal{O}(k_\varepsilon)$ pairs in $\mathcal{D}''$. □

**Lemma 6.7.** *Every ambiguous leaf cell of $T_{\text{init}}$ is important, namely $\mathcal{A} \subseteq \mathcal{I}$.*

*Proof.* Consider any ambiguous leaf cell $w \in \mathcal{A}$ of the tree $T_{\text{init}}$. Knowing that $w$ is ambiguous implies that there must exist some point $q \in \frac{\gamma}{2} b_w$ for which two of the $\varepsilon$-representatives of $w$ are valid $\varepsilon$-approximate nearest neighbors for $q$, both points belong to different classes, and the distance between them is $\Omega(\varepsilon\gamma r_w)$. Formally, denote these points as $p_1, p_2 \in P$ such that $l(p_1) \neq l(p_2)$, $\mathsf{d}(p_1, p_2) \geq \varepsilon\,(1-\gamma)\, r_w/4$, and $\mathsf{d}(q, p_1), \mathsf{d}(q, p_2) \leq (1+\varepsilon)\, \mathsf{d}_{\text{nn}}(q)$.

We will see now how to bound the distance from $c_w$ to any of these points as a constant factor of $r_w$ (recall that $r_w = \sqrt{d}/2\, s_w$). From the proof of Lemma 6.3 in [75], we know that the ball $c_3\gamma b_w \cap P \neq \varnothing$, for some constant $c_3 \geq 1 + 2c_2/\sqrt{d}$. In other words, $\mathsf{d}_{\text{nn}}(c_w) \leq c_3\gamma\, r_w$. From this, we can say that $\mathsf{d}_{\text{nn}}(q) \leq (\frac{1}{2} + c_3)\gamma r_w$. Applying the triangle inequality yields that $\mathsf{d}(c_w, p_1) \leq \mathsf{d}(c_w, q) + \mathsf{d}(q, p_1) \leq \left(\frac{1}{2} + (1+\varepsilon)(\frac{1}{2} + c_3)\right) \gamma\, r_w$. Similarly, we can achieve the same bound for $\mathsf{d}(c_w, p_2)$.

Therefore, both $p_1, p_2 \in \rho\gamma b_w$ for sufficiently large constant $\rho$ (*i.e.*, $\rho \geq \varepsilon(\frac{1}{2} + c_3) + c_3 + 1$). Knowing also that $\mathsf{d}(p_1, p_2) = \Omega(r_w^-) = \Omega(\varepsilon\gamma r_w)$ yields that the leaf cell $w \in \mathcal{I}$. □

**Lemma 6.8.** *Every boundary resolved leaf cell of $T_{\text{init}}$ is important, namely $\mathcal{R}_b \subseteq \mathcal{I}$.*

*Proof.* Let $w_1 \in \mathcal{R}_b$ be any boundary resolved leaf cell of the tree $T_{\text{init}}$, we know there exists another leaf cell $w_2 \in \mathcal{R}_b$ adjacent to $w_1$, such that there exists some class $i \in C_{w_2} \backslash C_{w_1}$. Let $b_{w_1}$ and $b_{w_2}$ be the corresponding bounding balls of $w_1$ and $w_2$. By definition, any point $q$ on the boundary shared by $w_1$ and $w_2$ has at least one $\varepsilon$-representative from each cell, namely some points $p_1 \in R_{w_1}$ and $p_2 \in R_{w_2}$, where

$l(p_1) \neq i$ and $l(p_2) = i$. Additionally, by similar arguments to the ones described in Lemma 6.7, we know that both $p_1, p_2 \in \rho\gamma b_w$ for sufficiently large constant $\rho$.

Now, we proceed to prove that $\mathsf{d}(p_1, p_2) \geq r_w^-/2$. First, note that if $w_1$ was resolved by the initial marking of leaf cells as described in Lemma 6.3, then $p_2$ must lie outside of $\gamma b_w$. In such cases, clearly $\mathsf{d}(p_1, p_2) \geq r_w^-/2$. The remaining possibility is that $w_1$ was resolved after computing the set of representatives. From the described procedure, in Case 1, we know that if $\mathsf{d}(p_1, p_2) < r_w^-/2$, the point $x_{p_1,i}$ (which is a 1-approximate nearest-neighbor of $p_1$ among points in $P_i$) would hold that $\mathsf{d}(p_1, x_{p_1,i}) < r_w^-$. Hence, $x_{p_1,i}$ should have been added to the set of representatives of $w_1$, contradicting the assumption that $w_1$ is resolved, or that $C_{w_1}$ does not contain $i$. All together, we have that $\mathsf{d}(p_1, p_2) = \Omega(r_w^-) = \Omega(\varepsilon\gamma r_w)$. From the definition of the set of important leaf cells, $w \in \mathcal{I}$. □

Lemmas 6.7 and 6.8 imply that all the leaf cells used to build $T$ belong to the set of important leaf cells (*i.e.*, $\mathcal{A} \cup \mathcal{R}_b \subseteq \mathcal{I}$), whose size is upper-bounded by Lemma 6.6. All together, and leveraging construction methods of quadtrees (see Section 6.2), the size of $T$ is proportional to the total number of leaf cells used to build it, which we now know is $\mathcal{O}(k_\varepsilon \gamma^d \log \frac{1}{\varepsilon})$. However, we also need to account for the set of $\varepsilon$-representatives stored for each ambiguous leaf cell, leading to a worst-case upper-bound of $\mathcal{O}(k_\varepsilon \gamma^d \log \frac{1}{\varepsilon} \cdot 1/(\varepsilon\gamma)^{\frac{d-1}{2}})$ total space to store $T$.

### 6.4.2 Spatial Amortization

The previous result can be improved by applying a technique called *spatial amortization*, described by Arya *et al.* [75]. That is, we can remove the extra $\mathcal{O}(1/(\varepsilon\gamma)^{\frac{d-1}{2}})$ factor from the analysis of the space requirements for $T$.

This will be twofold process, as in order to successfully apply spatial amortization to the analysis of the data structure, we first need to further reduce the set of $\varepsilon$-representatives of every ambiguous leaf cell in the tree. Actually, the new set will no longer be a set of $\varepsilon$-representatives, but it will just be a *weak* $\varepsilon$-coreset for query points inside of each leaf cell.

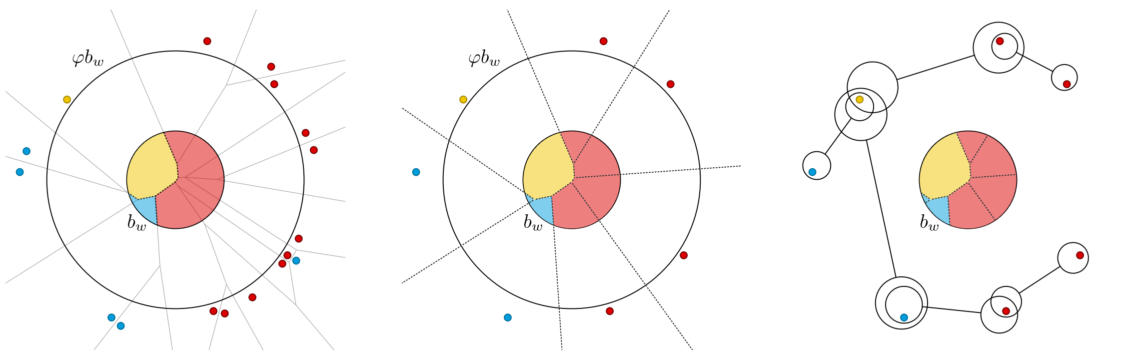
**Lemma 6.9.** *The total space required to store the ambiguous leaf cells of $T$ is $\mathcal{O}\big(k_\varepsilon \gamma^d \log \frac{1}{\varepsilon}\big)$.*

Consider any ambiguous leaf cell $w$ of $T$, and in particular, consider the set $R_w$ of $\varepsilon$-representatives of $w$. By construction, $R_w$ has the property that every point $q \in b_w$ has at least one $\varepsilon$-approximate nearest-neighbor in the set $R_w$. However, note that the opposite is not necessarily true, as not every $p \in R_w$ is an $\varepsilon$-approximate nearest-neighbor of some point in $b_w$. Even worst, while the fact the $w$ is ambiguous indicates that at least two points in $R_w$ belong to $K_\varepsilon$, the remaining points of $R_w$ might not, which in turn prevents the application of a spatial amortization analysis. Overall, this suggests some of the points of $R_w$ might not be necessary to distinguish between the classes that change the classification of points inside $b_w$ (see Figure 6.5a).

This can be resolved as follows. Suppose we have access to the Voronoi diagram of the set of points $R_w$, and consider the boundaries between adjacent cells of this diagram. Any boundary that separates two points of $R_w$ of different classes, and that intersects $b_w$, is relevant to the classification any query point inside $b_w$. Formally, we define the set $R'_w \subseteq R_w$ of border points of $R_w$ as (see Figure 6.5b):

$$R'_w = \{p \in R_w \mid \exists\, q \in b_w, p' \in R_w \text{ such that } l(p) \neq l(p') \wedge \mathsf{d}(q,p) = \mathsf{d}(q,p')\}$$

This new set $R'_w$ has some useful properties. Note that for any query point $q \in b_w$, its (exact) nearest-neighbor in $R'_w$ belongs to the same class as its (exact) nearest-neighbor in $R_w$, which itself is an $\varepsilon$-approximate nearest-neighbor of $q$ among the points of $P$. In other words, $R'_w$ is an $\varepsilon$-coreset for any query point in $b_w$. This implies that we can replace the set of $\varepsilon$-representatives of $w$ with the set $R'_w$. Moreover, this means that by the definition of $\varepsilon$-border points, $R'_w \subseteq K_\varepsilon$. Note that we can use the algorithm described in Chapter 3 to compute $R'_w$ in $\mathcal{O}(|R_w| \cdot |R'_w|^2)$ time, increases the construction time described in Lemma 6.5 by a factor of $k_\varepsilon^2$.




(a) Set of $\varepsilon$-reps $R_w$. (b) Set of border points $R'_w$. (c) Relevant pairs of $\mathcal{D}''$.

Figure 6.5: The set $R_w$ of $\varepsilon$-representatives of $w$ can be reduced to the set $R'_w$. This later set is a subset of $K_\varepsilon$, and can be charged to a proportional number of relevant pairs of $\mathcal{D}''$.

Now, let's proceed with the charging argument over the pairs of the same WSPD $\mathcal{D}''$ used in Lemma 6.6. Instead of only charging $w$ to a single pair (as described in Lemma 6.7), we charge every point stored in $R'_w$ to a pair of $\mathcal{D}''$. Thus, consider the following procedure to find a sufficient number of pairs to charge all the points in $R'_w$, which is derived from a similar procedure proposed in [75]. See Figure 6.5c for an illustrative example.

1. Compute a net of $R'_w$ using radius $r_w^-$, and denote this subset $R''_w$. Given that all the points of $R'_w$ that belong to the same class are already separated by a distance of at least $r_w^-$, we know that $|R''_w| = \Theta(|R'_w|)$, hiding constants[2] that depend exponentially on $d$.

2. Find the two of points of $p_1, p_2 \in R''_w$ with smallest pairwise distance, and consider the pair of $\mathcal{P} \in \mathcal{D}''$ that contains both points $p_1$ and $p_2$, each in one

[2] More specifically, we know that $|R''_w| \geq |R'_w|/\phi^{d-1}c$, where $\phi^{d-1}$ is the kissing number in $d$-dimensional Euclidean space, and $c$ is the number of classes in $P$.

of its dumbbell heads. Note that by having computed a net in the previous step, $\mathsf{d}(p_1, p_2) \geq r_w^-$.

3. Delete one of the two points from $R''_w$ (without lost of generality, delete $p_1$).

4. Charge every point of $R'_w$ that is covered by $p_1$ (*i.e.*, whose distance to $p_1$ is $\leq r_w^-$) to the pair $\mathcal{P}$. By the arguments described in step 1 on the size of $R''_w$, we know that $\mathcal{P}$ receives a charge from $\mathcal{O}(1)$ points of $R'_w$.

5. Repeat steps 2-4 with the remaining points of $R''_w$ until the set is empty.

Evidently, the number of pairs found (and charged) equals $|R''_w| - 1$. All together, we have that the total space required to store all the ambiguous leaf cells is proportional to the sum of charges to every pair of $\mathcal{D}''$. Using the same arguments as Lemma 6.6, this implies that the total storage is $\mathcal{O}(k_\varepsilon \gamma^d \log \frac{1}{\varepsilon})$. This completes the proof of Theorem 6.1.

# Chapter 7: Conclusions

In this dissertation, we have presented different techniques that successfully reduce the query time and space complexities of the nearest-neighbor rule. While usually nearest-neighbor classification would be highly dependent on $n$, the size of the training set $P$, our results provide a clear way to reduce its dependency to $k$. This parameter is defined as the number of border points of $P$, and characterizes the intrinsic complexity of the class boundaries of the training set.

The results presented here open a series of potential directions for future research to further improve the efficiency of the nearest-neighbor rule. While mostly theoretical, these results can help expand the scope of applicability of this classification method for larger real-world use cases. Furthermore, our results on training set reduction algorithms imply their potential application beyond the scope of nearest-neighbor classification, by reducing the data used to train other classification models. While many of the classification guarantees provided here would not hold –with most likeliness– for other classification methods, the upper-bounds on subset sizes would.

In the following sections, we retrace most of the results presented throughout this dissertation, and discuss some of the remaining open problems and limitations.

## 7.1 Training Set Reduction

Most of the results presented in this dissertation can be described as training set reduction approaches. In this case, the goal is to find a subset $R \subseteq P$, whose induced class boundaries resemble the original class boundaries of $P$. This subset is then used by the nearest-neighbor rule to answer any incoming queries, instead of using the full training set. As seen throughout this book, different approaches yield different subsets, and depending on the approach used, the classification guarantees of the nearest-neighbor rule after training set reduction can vary.

### 7.1.1 Boundary Preservation

The goal of boundary preservation algorithms is to compute the set of border points of $P$. Given that by definition, this subset of points fully characterize the class boundaries of $P$, this reduction of the training set does not affect the classification accuracy of the nearest-neighbor rule. That is, the class boundaries remain the same. Moreover, these algorithms achieve the expected dependency reduction from $n$ to $k$, as $k$ is defined as the size of their selected subset.

In Chapter 3, we present an improvement over Eppstein's recent algorithm [8] to compute the set of border points of any training set $P \subset \mathbb{R}^d$, with constant $d$. This improved algorithm reduces the complexity of computing such subset to $\mathcal{O}(nk^2)$ worst-case time, while Eppstein's original algorithm has $\mathcal{O}(n^2 + nk^2)$ runtime. A paper describing this result is under submission, and can be found here [52].

### 7.1.2 Condensation Algorithms

Alternatively, condensation algorithms aim to compute subsets $R \subseteq P$ whose induced class boundaries are "similar" to the original class boundaries, albeit not the same as with boundary preservation algorithms. As formally described in Chapter 2, condensation algorithms select either consistent or selective subsets of $P$, which can only guarantee the correct classification of points in $P$. This is a rather popular line of research [9, 13–17, 34, 38], with many heuristic approaches proposed to compute such subsets. Until our work, theoretical results on these heuristics were scarce.

In Chapter 4, we present the first theoretical results on upper-bounding the subset sizes of condensation algorithms. In particular, we show that FCNN and MSS, which were considered state-of-the-art for the condensation problem, can not be bounded in terms of $k$. Additionally, we propose new quadratic-time algorithms called SFCNN, RSS, and VSS, and prove upper-bounds on their subset sizes as a function of $k$. These upper-bounds are tight up-to constant factors, and supports the good empirical behavior observed for these condensation algorithms. Most of these results were published in a series of papers [54–56].

Despite our efforts, some questions remain open. First, it is unclear whether our upper-bound for the RSS algorithm in terms of $k$ can be improved to have linear dependencies on $d$, as our current result has a term with exponential dependency on $d$. So far, we have been unable to find a matching lower-bound for RSS in terms of $k$, keeping the hope that this improvement is indeed possible. The other important open question revolves around SFCNN. While this algorithm performs really well in practice, we were only able to prove a $\mathcal{O}(k \log(1/\gamma))$ upper-bound on its selected subset (compared with the $\mathcal{O}(k)$ upper-bound on RSS). Thus, it remains open whether the $\log(1/\gamma)$ factor in the upper-bound of SFCNN can be dropped.

### 7.1.3 $\varepsilon$-Coresets

Both *boundary preservation* and *condensation* algorithms make the assumption that the nearest-neighbor rule computes nearest-neighbors exactly. However, efficient data structures for nearest-neighbor search compute nearest-neighbor approximately rather than exactly. That is, given an approximation parameter $\varepsilon \in [0, 1]$, a query point $q \in \mathcal{X}$ can be assigned the class of any point of $P$ whose distance to $q$ is at most $1+\varepsilon$ times the distance from $q$ to its true nearest-neighbor. This relaxed assumption immediately breaks the classification guarantees provided by both types of training set reduction techniques.

Chapter 5 presents a new framework for training set reduction that is sensitive to these approximations, as well as a characterization of $\varepsilon$-coresets for the nearest-neighbor rule. First, we define such an $\varepsilon$-coreset as a subset $R \subseteq P$ where for every query point $q \in \mathcal{X}$ the class of its exact nearest-neighbor in $R$ is the same as the class of a valid $\varepsilon$-approximate nearest-neighbor of $q$ in $P$. In order to compute such subsets, we extended the criteria used for condensation to be approximation-sensitive, and modified the condensation algorithms from Chapter 4 to compute subsets under these new criteria. Finally, this gives us a way to compute $\varepsilon$-coresets of size $\mathcal{O}(k \log \frac{1}{\gamma} (1/\varepsilon)^{\text{ddim}(\mathcal{X})+1})$. These results have been published in the following paper [66].

The biggest shortcoming of this result is the size of the $\varepsilon$-coresets, having high dependencies on $\gamma$ and $\varepsilon$. The lower-bound presented in Lemma 5.3 indicate that our coreset construction might not be optimal, and that the terms on $\gamma$ and $\varepsilon$ could be potentially dropped. However, despite our efforts, we were unable to achieve this.

## 7.2 Chromatic Nearest-Neighbor Search

Unsurprisingly, the standard approach to answer nearest-neighbor classification queries is to reduce them to nearest-neighbor search queries. However, as described in Chapter 6, there exists an alternative approach towards achieving more efficient nearest-neighbor classification. This consists of having algorithms and data structures to directly compute the class of the nearest-neighbor of any given query point; *i.e.*, without computing the nearest-neighbor itself.

To this end, we proposed a tailor-made data structure for approximate nearest-neighbor classification called *Chromatic* AVD. Given a training set $P$ in $d$-dimensional Euclidean space (assuming constant $d$ and the $\ell_2$ metric) and an approximation parameter $\varepsilon \in [0, \frac{1}{2}]$, we construct a quadtree-based partitioning of space to answer any classification query approximately. That is, for any query point $q \in \mathbb{R}^d$ this data structure returns the class of any of $q$'s valid $\varepsilon$-approximate nearest-neighbors in $P$. This data structure is designed as a simplification of state-of-the-art AVDs [20] for approximate nearest-neighbor search, and completely reduces its dependency from $n$ to $k_\varepsilon$, which is defined as the number of $\varepsilon$-border points of $P$, and describes the intrinsic complexity of the $\varepsilon$-approximate class boundaries. These results have been published in the following paper [78].

There are a few possible improvements and extensions that would be worth exploring. First, the query time and space dependencies of the Chromatic AVD are expressed in terms of $k_\varepsilon$ and not $k$, where $k_\varepsilon \geq k$. However, it is unclear if $k_\varepsilon$ can be expressed in terms of $k$, or if the analysis and construction of the Chromatic AVD can be improved to reduce the expressed dependencies to be in terms of $k$. Another interesting direction is trying to obtain query times that are query-sensitive, similar to the result by Mount *et al.* [51] in 1997, where query points with high chromatic density are easier to answer than query points with low chromatic density. Finally, yet another direction would be on extending this data structure to answer $\varepsilon$-approximate $\mathsf{k}$-NN classification queries, for $\mathsf{k} \geq 1$. While there exists some work on this problem [51, 81], all existing results have dependencies on $n$.

# Bibliography


[1] 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.

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

[3] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, January 1967.

[4] 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.

[5] Juha Heinonen. *Lectures on analysis on metric spaces*. Springer Science & Business Media, 2012.

[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.

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

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

[9] P. Hart. The condensed nearest neighbor rule (corresp.). *IEEE Trans. Inf. Theor.*, 14(3):515–516, September 1968.

[10] 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.

[11] 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.

[12] 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.

[13] Godfried Toussaint. Open problems in geometric methods for instance-based learning. In Jin Akiyama and Mikio Kano, editors, *JCDCG*, volume 2866 of *Lecture Notes in Computer Science*, pages 273–283. Springer, 2002.

[14] Godfried Toussaint. Proximity graphs for nearest neighbor decision rules: Recent progress. In *Progress", Proceedings of the 34th Symposium on the INTERFACE*, pages 17–20, 2002.

[15] 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.

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

[17] Ricardo Barandela, Francesc J Ferri, and J Salvador Sánchez. Decision boundary preserving prototype selection for nearest neighbor classification. *International Journal of Pattern Recognition and Artificial Intelligence*, 19(06):787–806, 2005.

[18] Salvador Garcia, Joaquin Derrac, Jose Cano, and Francisco Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(3):417–435, March 2012.

[19] 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.

[20] 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.

[21] Piotr Indyk, Rajeev Motwani, Prabhakar Raghavan, and Santosh Vempala. Locality-preserving hashing in multidimensional spaces. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, page 618–625, New York, NY, USA, 1997. Association for Computing Machinery.

[22] Yu A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. 42(4):824–836, apr 2020.

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

[24] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014.

[25] 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.

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

[27] 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.

[28] Marc Khoury and Dylan Hadfield-Menell. Adversarial training with Voronoi constraints. *CoRR*, abs/1905.01019, 2019.

[29] 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.

[30] 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.

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

[32] Belur V. Dasarathy. Nearest unlike neighbor (nun): an aid to decision confidence estimation. *Optical Engineering*, 34(9):2785–2792, 1995.

[33] 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.

[34] G. L. Ritter, H. B. Woodruff, S. R. Lowry, and T. L. Isenhour. An algorithm for a selective nearest neighbor decision rule. *IEEE Transactions on Information Theory*, 21(6):665–669, 1975.

[35] Richard Nock and Marc Sebban. Sharper bounds for the hardness of prototype and feature selection. In *Proceedings of the 11th International Conference on Algorithmic Learning Theory*, ALT '00, page 224–237, Berlin, Heidelberg, 2000. Springer-Verlag.

[36] Lee-Ad Gottlieb, Aryeh Kontorovich, and Pinhas Nisnevitch. Near-optimal sample compression for nearest neighbors. In *Advances in Neural Information Processing Systems*, pages 370–378, 2014.

[37] Rajesh Chitnis. Refined lower bounds for nearest neighbor condensation. In Sanjoy Dasgupta and Nika Haghtalab, editors, *Proceedings of The 33rd International Conference on Algorithmic Learning Theory*, volume 167 of *Proceedings of Machine Learning Research*, pages 262–281. PMLR, 29 Mar–01 Apr 2022.

[38] Geoffrey W. Gates. The reduced nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 18(3):431–433, 1972.

[39] Ahmad Biniaz, Sergio Cabello, Paz Carmi, Jean-Lou De Carufel, Anil Maheshwari, Saeed Mehrabi, and Michiel Smid. On the minimum consistent subset problem. In *Workshop on Algorithms and Data Structures*, pages 155–167. Springer, 2019.

[40] Kenneth L Clarkson. A randomized algorithm for closest-point queries. *SIAM Journal on Computing*, 17(4):830–847, 1988.

[41] Andrew C Yao and F Frances Yao. A general approach to d-dimensional geometric queries. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 163–168, 1985.

[42] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, November 1998.

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

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

[45] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3887–3896. PMLR, 13–18 Jul 2020.

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

[47] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, 16(2):187–260, jun 1984.

[48] Hanan Samet. *The design and analysis of spatial data structures*, volume 85. Addison-Wesley Reading, MA, 1990.

[49] Hanan Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

[50] Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. Approximate nearest neighbor search in high dimensions. *arXiv preprint arXiv:1806.09823*, 2018.

[51] David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. Chromatic nearest neighbor searching: A query sensitive approach. *Computational Geometry*, 17(3):97 – 119, 2000.

[52] Alejandro Flores-Velazco. Improved search of relevant points for nearest-neighbor classification. *CoRR*, abs/2203.03567, 2022.

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

[54] 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.

[55] 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.

[56] Alejandro Flores-Velazco and David M. Mount. Guarantees on nearest-neighbor condensation heuristics. *Computational Geometry*, 95:101732, 2021.

[57] Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Geometric approximation via coresets. *Combinatorial and computational geometry*, 52:1–30, 2005.

[58] Jeff M. Phillips. Coresets and sketches, 2016.

[59] Dan Feldman. Core-sets: Updated survey. In *Sampling Techniques for Supervised or Unsupervised Tasks*, pages 23–44. Springer, 2020.

[60] Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 291–300, 2004.

[61] Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. *arXiv preprint arXiv:1804.05345*, 2018.

[62] Vladimir Braverman, Dan Feldman, and Harry Lang. New frameworks for offline and streaming coreset constructions, 2016.

[63] Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 569–578, 2011.

[64] Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. *arXiv preprint arXiv:1911.07412*, 2019.

[65] Murad Tukan, Cenk Baykal, Dan Feldman, and Daniela Rus. On coresets for support vector machines, 2020.

[66] Alejandro Flores-Velazco and David M. Mount. Coresets for the nearest-neighbor rule, 2020.

[67] Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM Journal on Computing*, 35(5):1148–1184, 2006.

[68] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[69] Azaria Paz and Shlomo Moran. Non deterministic polynomial optimization problems and their approximations. *Theoretical Computer Science*, 15(3):251–277, 1981.

[70] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM (JACM)*, 41(5):960–981, 1994.

[71] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[72] Petr Slavik. A tight analysis of the greedy algorithm for set cover. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 435–441, New York, NY, USA, 1996. ACM.

[73] F. Aurenhammer and H. Edelsbrunner. An optimal algorithm for constructing the weighted Voronoi diagram in the plane. *Pattern Recognition*, 17(2):251 – 257, 1984.

[74] Jon Louis Bentley and James B. Saxe. Decomposable searching problems i. static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.

[75] 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.

[76] Richard Cole and Lee-Ad Gottlieb. Searching dynamic point sets in spaces with bounded doubling dimension. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 574–583, 2006.

[77] Timothy M. Chan. A minimalist's implementation of an approximate nearest neighbor algorithm in fixed dimensions (2006). *URL: http://tmc.web.engr.illinois.edu/sss.ps.*

[78] 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.

[79] Paul B. Callahan and S. Rao Kosaraju. A decomposition of multidimensional point sets with applications to $k$-nearest-neighbors and $n$-body potential fields. *Journal of the ACM (JACM)*, 42(1):67–90, 1995.

[80] Raphael A. Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.

[81] TWJ van der Horst. Chromatic k-nearest neighbors queries using (near-) linear space. Master's thesis, 2021.