University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Computer Science

Exact Robustness Certification of k-Nearest Neighbor Classifiers

Supervisor
Prof. Francesco Ranzato
University of Padova

Co-supervisor
Marco Zanella
University of Padova

Master Candidate
Ahmad Shakeel

Student ID
1237579

Academic Year
2024-2025

"Computer science is no more about computers than astronomy is about telescopes."

— Edsger Dijkstra

Acknowledgments

This master's degree has been a wonderful experience that allowed me to delve into topics I am deeply passionate about. It has been a journey of intellectual growth and discovery that I will cherish forever.

I would like to express my sincere gratitude to everyone who helped and supported me throughout these years of my master's program. Your guidance and encouragement were invaluable. A special thanks to my beloved family and friends who consistently believed in me, even during challenging times. Your unwavering support gave me the strength to persevere despite all the difficulties.

Last but not least, I would like to express my gratitude to my supervisor Prof. Francesco Ranzato and co-supervisor Marco Zanella for their support and especially patience during the development of this work.

Padova, April 2025 Ahmad Shakeel

Abstract

Machine learning (ML) models are increasingly being applied in a variety of fields, including healthcare, finance, and autonomous transportation. As ML systems are deployed in safety-critical domains, their robustness against adversarial attacks becomes crucial, especially when errors can result in life-threatening consequences. Adversarial examples, which are inputs altered by small perturbations that lead to incorrect outcomes in ML models, pose significant security risks. This paper focuses on certifying the robustness and stability of k-nearest neighbors (k-NN) classifiers against such adversarial attacks. We propose a novel method that models adversarial attacks as a perturbation region around a test sample and analyzes the labels assigned by k-NN to samples within this region. Our approach constructs a directed graph where nodes represent training samples, and edges indicate the relative proximity of these samples to the perturbation region. We perform a graph traversal to collect the most frequent labels from paths consisting of k samples, ensuring that the proximity conditions are satisfied. If only one label is found, we guarantee the stability of k-NN w.r.t. the input sample. Furthermore, if this label matches the ground truth, we conclude that the classifier is also robust. Otherwise, if multiple labels are found, k-NN is neither stable nor robust. We implemented the algorithm in Python and evaluate it on seven datasets commonly used for robustness verification and four datasets for fairness evaluation. Our experimental results demonstrate that the proposed certifier can successfully verify the robustness of more than 90% of the samples under adversarial perturbations, showcasing k-NN's potential as a robust classification algorithm.

Contents

Acknowledgments **v**

Abstract **vii**

List of figures **xi**

List of tables **xi**

List of algorithms **xii**

List of acronyms **xii**

1 Introduction **1**

2 Background **7**
2.1 Notation 7
2.2 Machine Learning 8
2.2.1 Supervised Classification 10
2.2.2 K-NN Classifiers 11
2.2.3 Adversarial Examples 13
2.2.4 Stability and Robustness 15
2.2.5 Individual Fairness 16

3 Related Works **19**
3.1 Theoretical analysis 20
3.2 QP-based analysis 20
3.3 Region-based analysis 21
3.4 Gradient-based analysis 22
3.5 Geometric-based analysis 23
3.6 Abstract interpretation-based analysis 24

4 Methodology **25**
4.1 G-kNN Algorithm 26
4.1.1 Proximity Precedence Graph 27
4.1.2 Nearest Neighbors Selection 28
4.1.3 The Full Algorithm 32

4.2 Exact Robustness Certification 33
4.2.1 Adversarial Proximity Precedence Graph 34
4.2.2 Nearest Neighbors Selection 39
4.2.3 Full Algorithm 45

5 Optimizations **47**
5.1 APP-G Construction Optimizations 48
5.1.1 Dataset partitioning 49
5.1.2 Searching in the BSP Tree 50
5.2 Certifier Optimizations 52

6 Experimental Evaluation **59**
6.1 Datasets 59
6.2 Perturbations 62
6.3 Preprocessing 62
6.4 Results 62
6.4.1 Stability and Robustness Certification 64
6.4.2 Individual Fairness Certification 76

7 Conclusion and Future Works **81**

References **83**

List of Figures

1.1 Example showing the algorithm 4

2.1 Venn diagram showing the relation between artificial intelligence, machine learning and deep learning 8
2.2 3NN over a dataset with 3 classes, which shows how a new point is classified (left), as well as its label after being classified (right) 12

4.1 Example showing how to determine if $\boldsymbol{s}_1 \preccurlyeq_{\boldsymbol{x}} \boldsymbol{s}_2$. 27
4.2 Example showing a PP-G 29
4.3 Example showing how to determine if $\boldsymbol{s}_1 \preccurlyeq^{A}_{\boldsymbol{x}} \boldsymbol{s}_2$. 35
4.4 Example showing a APP-G 38
4.5 Example of non adversarially valid path in a APP-G 40
4.6 Example showing how to assess the adversarial validity of a path in a APP-G 42

6.1 Certified stability and robustness on the whole Australian test set . . . 66
6.2 Certified stability and robustness on the whole BreastCancer test set . 68
6.3 Certified stability and robustness on the whole Diabetes test set 70
6.4 Certified stability and robustness on the whole Fourclass test set 72
6.5 Certified stability and robustness on the whole Letter test set 73
6.6 Certified stability and robustness on the whole Pendigits test set 75
6.7 Certified stability and robustness on the whole SatImage test set 76
6.8 Certified fairness on the whole Adult test set 77
6.9 Certified fairness on the whole Compas test set 78
6.10 Certified fairness on the whole Crime test set 79
6.11 Certified fairness on the whole German test set 80

List of Tables

6.1 Datasets used in experimental evaluations 61
6.2 Accuracy of each dataset for $k \in \{1, 3, 5, 7\}$ 61
6.3 Average certification time 64

List of Algorithms

2.1 k-NN algorithm 13
4.1 CreatePPGraph method 28
4.2 ExtractValidPath method 31
4.3 G-kNN full algorithm 32
4.4 CreateAPPGraph method 39
4.5 Closer method 43
4.6 SelectAdvValidPaths method 44
4.7 Certifier algorithm 45
5.1 BuildBSPTree algorithm 50
5.2 SearchClosePoints algorithm 51
5.3 SearchPartition algorithm 52
5.4 OptCertifier algorithm 55

List of Acronyms

***k*-NN**	*k*-Nearest Neighbors
G-*k*NN	Graph based *k*-Nearest Neighbors
PP-G	Proximity Precedence Graph
APP-G	Adversarial Proximity Precedence Graph
***k*NAVe**	*k*-NN Abstract Verifier
ML	Machine Learning
QP	Quadratic Programming
AI	Artificial Intelligence

1 Introduction

Machine learning (ML) models are finding widespread application in a multitude of fields, ranging from computer vision [30] and natural language processing to finance [36], healthcare [1], and beyond. This widespread usage, particularly when applied in safety-critical domains (e.g., healthcare, autonomous transportation, etc.), where an error can lead to potentially irreversible and life-threatening consequences [3], has led to the emergence of a body of research focused on the robustness and security of these methods. In the context of ML, *robustness* refers to the resilience of the ML system against vulnerabilities that can alter its expected behavior or outcome. One such vulnerability—and the focus of this work—is *adversarial examples*, which are created by applying small perturbations to input samples, making them perceptually indistinguishable from the originals, in order to change the outcome of the ML system. A ML system is robust to an adversarial attack (i.e., when an adversarial example is given as input) if the outcome remains the same as the expected one for the unperturbed input. It is important to note that, by definition, determining the robustness of an adversarial example can only be done when the outcome of the unperturbed sample is known *a priori*. This is only possible during testing, when the ground truth is available. If the outcomes of the unperturbed sample and adversarial example are the same, regardless of accuracy, the ML system is defined as *stable*. So, stability only requires the outcome to be the same, while robustness also imposes the correctness constraint, meaning that the outcome must coincide with the one associated with the test sample.

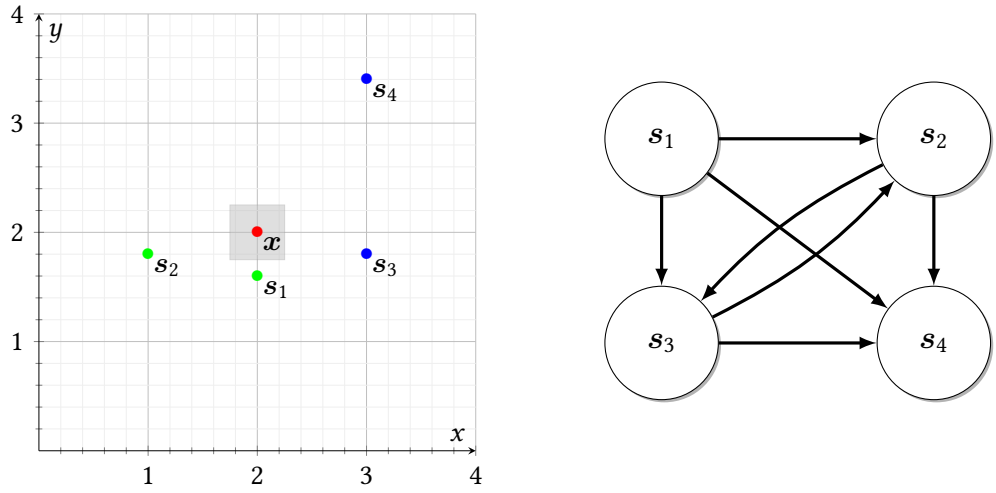
Adversarial examples pose a significant security risk for practical machine learning applications. For example, consider a ML system that receives voice commands. A malevolent agent could create a seemingly innocuous recording, such as a song, that contains hidden voice commands imperceptible to humans but recognizable by the ML system. This recording would be an adversarial example, as the ML system should have ignored the input. Another example of an adversarial attack impacting the reliability of a ML system is in the face recognition domain. An adversarial example in this scenario could be obtained by applying imperceptible changes to a person's face image, so a human observer would recognize their identity correctly, but the ML system would recognize them as a different person. Robustness against adversarial attacks, whether accidental or intentional, is highly desirable for machine learning models. Unfortunately, several studies have demonstrated that ML models are highly susceptible to adversarial examples [17, 55, 24, 29]. Moreover, in [55], the authors showed that adversarial examples exhibit transferability, meaning that an adversarial example designed to affect model M_1 can often affect another model, M_2. This means it is possible to perform an adversarial attack on a ML system without knowing the underlying model. Additionally, in [29], the authors further showed that adversarial examples can also transfer to the physical world, meaning that ML systems operating in the physical world and perceiving data through various sensors (e.g., cameras) are also vulnerable to adversarial examples.

Given the potential risks posed by adversarial attacks, significant research has focused on mitigating these threats through robustness verification [60, 65] and adversarial training [24, 10, 7]. However, much of this research has concentrated on (deep) neural network-based models, given their success and widespread adoption. As a result, relatively little attention has been given to other types of ML models, one of which is k-nearest neighbors [21, 22], also known in literature as k-NN. This method is mainly used for *classification tasks*, where the objective is to assign a discrete label representing a certain category to an input sample. Following the simple intuition that similar data with respect to a similarity metric should be classified similarly, k-NN infers the label for an input sample by taking the most frequent label among the k most similar samples in a given dataset—hence the name k-nearest neighbors. Despite its simplicity, k-NN is widely used in various applications [61, 28], and more recently, it has been combined with deep neural networks to solve tasks such as remote sensing image retrieval [63], machine translation [26], and face recognition [37]. In particular, k-NNs is successfully

applied in various tasks where adversarial attacks must be considered. In chemistry, for instance, 1-NN was used to classify molecular structures using nuclear magnetic resonance spectra [27], while the general version was used to classify sensor array data for two types of chemical warfare agents [46], substances whose toxic properties are meant to kill, injure, or incapacitate human beings. A second example is [56], where k-NN, in combination with three feature selection methods, was used to classify three types of cancer: leukemia, colon, and breast cancer.

In this research, we propose a novel algorithm to certify the robustness and stability of k-NN against adversarial attacks. Our methodology models an adversarial attack as a small region $P(\boldsymbol{x})$ of space around a test sample, $\boldsymbol{x}$, called the *perturbation (or adversarial)* region of $\boldsymbol{x}$ and searches for the labels that k-NN assigns to samples within $P(\boldsymbol{x})$ (i.e., the adversarial examples). To find this labels, our approach constructs a directed graph where the nodes represent the samples in the training set. An edge from sample $\boldsymbol{s}_i$ to $\boldsymbol{s}_j$ indicates that there exists a point within the perturbation region that is closer to $\boldsymbol{s}_i$ than to $\boldsymbol{s}_j$ w.r.t to the Euclidean distance. The method then performs a principled traversal of this graph, starting from the samples with no incident edges, and collects the most frequent labels from paths consisting of k samples. Only the paths whose elements satisfy the relative proximity to the adversarial examples are considered. Specifically, the most frequent labels are extracted from a path $\mathcal{P} = [\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_k]$ only if there exists a point $\boldsymbol{x}'_i \in P(\boldsymbol{x})$ such that $\boldsymbol{s}_1$ is the closest sample to $\boldsymbol{x}'_i$, $\boldsymbol{s}_2$ is the second closest, and so on. To determine the existence of the point $\boldsymbol{x}'_i$, we check for the intersection between the perturbation region and k high-order Voronoi cells [31], where the sites are the samples in $\mathcal{P}$. If there is an intersection, the path $\mathcal{P}$ is considered; otherwise, it is ignored in the traversal of the graph. If this method finds only a single label, we can guarantee with absolute certainty the stability of k-NN for the sample $\boldsymbol{x}$ with respect to $P(\boldsymbol{x})$. Additionally, if this single label coincides with the ground truth of $\boldsymbol{x}$, we can also conclude that the k-NN classifier is robust for the sample $\boldsymbol{x}$. Otherwise, if more than one label are found, k-NN is neither stable nor robust.

We implemented this algorithm in Python and performed an exhaustive experimental evaluation of the final certifier on 7 datasets commonly used for formal robustness verification and on 4 standard datasets for individual fairness verification, achieving promising results in most of these. The experimental evaluation demonstrates that k-NN is a fairly robust classification algorithm, as our certifier was able to certify the robustness of more than 90% of the samples with adversarial perturbations of ±3% in most datasets.



(a) Input sample and the dataset $\mathcal{S}$ with two labels: *blue* and green

(b) Graph constructed by the certifier.

Figure 1.1: Example showing the workings of the algorithm

Illustrative Example. Consider a dataset $\mathcal{S} \subset \mathbb{R}^2$ and the adversarial region $P(\boldsymbol{x})$ of the input sample $\boldsymbol{x} \in \mathbb{R}^2$ with magnitude $\epsilon = 0.25$, as shown in Figure 1.1a. In this scenario, the sample $\boldsymbol{s}_1$ is the closest to any point in $P(\boldsymbol{x})$, while the sample $\boldsymbol{s}_4$ is the farthest. The samples $\boldsymbol{s}_2$ and $\boldsymbol{s}_3$ are equidistant to $P(\boldsymbol{x})$, meaning that there are points in $P(\boldsymbol{x})$ that are closer to $\boldsymbol{s}_2$ than to $\boldsymbol{s}_3$ and vice versa.

Figure 1.1b shows the graph constructed by the certifier. There are no incident edges to $\boldsymbol{s}_1$ since it is the closest sample, and there are no outgoing edges from $\boldsymbol{s}_4$ as it is the farthest. The samples $\boldsymbol{s}_2$ and $\boldsymbol{s}_3$ are adjacent to each other since they are equidistant.

With this graph, the certifier starts the traversal from $\boldsymbol{s}_1$, as it has no incoming edges. Depending on the value of k, it visits the other samples. If $k = 1$, the certifier stops at the first sample and returns the set $\{green\}$. If $k = 2$, the certifier computes the most frequent label within the paths $[\boldsymbol{s}_1, \boldsymbol{s}_2]$ and $[\boldsymbol{s}_1, \boldsymbol{s}_3]$. The most frequent label is *green* for the first path and both colors for the second, due to a tie. Hence, the certifier returns the set $\{green, blue\}$. For $k = 3$, it collects the most frequent label within the paths $[\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3]$ and $[\boldsymbol{s}_1, \boldsymbol{s}_3, \boldsymbol{s}_2]$. In both paths, the most frequent label is *green*, so the certifier returns the singleton $\{green\}$. Notice that the paths $[\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_4]$ and $[\boldsymbol{s}_1, \boldsymbol{s}_3, \boldsymbol{s}_4]$ are not considered, because there are no points in $P(\boldsymbol{x})$ that are closer to $\boldsymbol{s}_4$ than to $\boldsymbol{s}_3$ in the first path, or to $\boldsymbol{s}_2$ in the second. Therefore, these paths are ignored during the

graph traversal. Finally, for $k = 4$, there is only one path, which includes all the samples in $\mathcal{S}$, and again, due to a tie, the certifier returns the set $\{green, blue\}$.
The remainder of this document is organized as follows:

Chapter 2 describes the background needed to better understand the concepts covered in this research work, providing some basic notions about machine learning;

Chapter 3 reviews state-of-the-art methods currently applied to verify major machine learning models and highlights some recent works finding adversarial examples on k-NNs;

Chapter 4 describes our novel method, explaining in detail how it works and why it is exact;

Chapter 6 shows the experimental results obtained by executing our certifier on the reference datasets;

Chapter 7 sums up the contribution of this research work and proposes future improvements to possibly obtain even better performance.

2
Background

This chapter introduces and describes the foundational knowledge necessary to fully understand the topics presented in this thesis. It begins by defining the notations used throughout the subsequent chapters. Following this, it provides an overview of machine learning and supervised classification, including a detailed explanation of the k-NN classifier. Finally, it addresses the safety and fairness issues associated with these methods, and outlines the properties that must be evaluated as a result.

2.1 Notation

An n-dimensional vector space $\mathcal{X} \subseteq \mathbb{R}^n$, called *input feature space*, and a set of classification labels $\mathcal{L} \subset \mathbb{N}$ are assumed. Vectors in this input feature space are denoted with bold letters while natural numbers are denoted with normal letters. Given vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, and a constant $c \in \mathbb{R}^n$:

- $\forall i \in \{1, n\}$ $\boldsymbol{x}_i \in \mathbb{R}$ denotes the i-th component of $\boldsymbol{x}$;
- $\boldsymbol{x} \cdot \boldsymbol{y} \doteq \sum_{i=1}^{n} \boldsymbol{x}_i + \boldsymbol{y}_i$ represents the dot product between two vectors;
- $\boldsymbol{x} + \boldsymbol{y} \doteq (\boldsymbol{x}_1 + \boldsymbol{y}_1, \boldsymbol{x}_2 + \boldsymbol{y}_2, \ldots, \boldsymbol{x}_n + \boldsymbol{y}_n)$ is the vector addition of two vectors;
- $\boldsymbol{x} + c \doteq (\boldsymbol{x}_1 + c, \boldsymbol{x}_2 + c, \ldots, \boldsymbol{x}_n + c)$ denotes the summation between a vector and constant;

- $c \cdot \boldsymbol{x} \doteq (c \cdot \boldsymbol{x}_1, c \cdot \boldsymbol{x}_2, \ldots, c \cdot \boldsymbol{x}_n)$ denote the multiplication between a constant;
- $\|\boldsymbol{x}\|_2 = \sqrt{\boldsymbol{x} \cdot \boldsymbol{x}}$ denotes the ℓ_2 (i.e., Euclidean) norm;
- $\|\boldsymbol{x}\|_\infty = \max\{|\boldsymbol{x}_i| \mid i \in \{1, n\}\}$ is ℓ_∞ (i.e., maximum) norm;

A dataset will be denoted as $\mathcal{S} = \{\boldsymbol{z}^{(i)}\}_{i=1}^n = \{(\boldsymbol{x}^{(i)}, l^{(i)})\}_{i=1}^n \subset \mathcal{Z} = \mathcal{X} \times \mathcal{L}$. Given a dataset $\mathcal{S}$:

- $\mathcal{S}_X$ denotes the samples in the dataset $\mathcal{S}$;
- $\mathcal{S}_L$ denotes the labels in the dataset $\mathcal{S}$;

If not specified otherwise, $\mathcal{S}, \mathcal{T}$ will denote the training set and the test set respectively. Given $\boldsymbol{n} \in \mathbb{R}^n, b \in \mathbb{N}$ let

$$\pi_{\boldsymbol{n},b} = \boldsymbol{n}_1 x_1 + \boldsymbol{n}_2 x_2 + \cdots + \boldsymbol{n}_n x_n + b = 0$$

be the equation of a hyperplane. Then $\pi_{\boldsymbol{n},b}(\boldsymbol{y}) = n_1 \cdot y_1 + n_2 \cdot y_2 + \cdots + n_n \cdot y_n + b$

2.2 Machine Learning

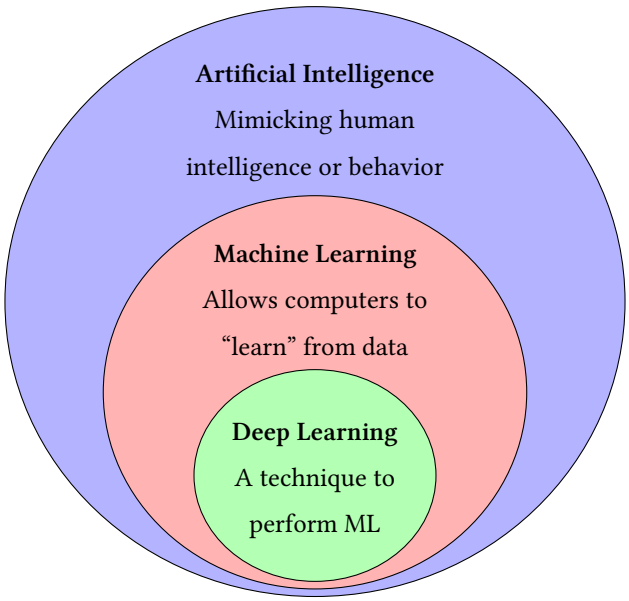


Figure 2.1: Venn diagram showing the relation between artificial intelligence, machine learning and deep learning.

In contemporary discourse, artificial intelligence (AI), machine learning, and deep learning are increasingly prevalent terms. While often confused or used interchangeably, these concepts are, in fact, distinct.

Artificial intelligence is a rapidly expanding field with numerous practical applications and active research areas. As defined by its pioneer, John McCarthy [32]:

"The science and engineering of making intelligent machines, especially intelligent computer programs."

Its main goal is to employ diverse technologies to develop machines and computers capable of replicating cognitive functions associated with human intelligence, thereby automating mind- and time-consuming tasks. AI is particularly suited for solving problems that are intellectually challenging for humans but readily definable in a formal mathematical manner understandable by computers. Conversely, tasks that our minds perform effortlessly, such as recognizing objects, faces, or sounds, categorizing document subjects, or identifying entities (places, titles, names, actions, etc.) in phrases or images, pose a challenge in formalization for computer understanding. This is because we learn these tasks through development and experience. The real challenge arises when precise problem formalization is impossible, when input or output uncertainty exists, and when solutions are excessively complex or inefficient.

Machine learning is a subset of artificial intelligence designed to address these challenges. ML focuses on creating systems or models that learn inherent data patterns and improve performance in specific tasks without explicit programming. These models learn from past experiences or examples to make decisions on new data, contrasting with traditional AI methods where human programmers write rules to transform input data into desired results. The underlying assumption is that a stochastic process governs real-world data, and machine learning methods approximate this process by learning patterns and correlations within available data (a representation of real-world data). The learned process is then used to make predictions on unseen samples. Data is therefore crucial in ML model learning, commonly represented as sets of n-dimensional vectors in $\mathbb{R}^n$ where each vector component is called *attribute* or *feature*. For example, the features of an image, represented as a vector, can be its pixels ranging from 0 to 255. Data and tasks determine which ML method is most suitable and performing for each specific case. Within machine learning, the following learning paradigms are identified:

- **Supervised learning**: data is presented as a set of examples $\bigcup_{i=1}^{n}\{(\boldsymbol{x}^{(i)}, l^{(i)})\}$ known as the training set. Each example is a pair consisting of an input feature vector from $\mathcal{X}$ and its corresponding output, typically a label. Supervised learning is particularly well-suited for classification and regression tasks: a classification function is learned for discrete outputs, while a regression function is learned for continuous outputs. The learned function is then used to predict the output for unseen data.;
- **Unsupervised learning**: data are presented as a set of unlabelled vectors $\bigcup_{i=1}^{n}\{\boldsymbol{x}_i\}$. In this scenario, the outputs represent the inherent data structure, determined by minimizing a cost function. Unsupervised learning is particularly well-suited for clustering and dimensionality reduction tasks;
- **Reinforcement learning**: as for unsupervised learning, data are presented as a set of unlabelled vectors $\bigcup_{i=1}^{n}\{\boldsymbol{x}_i\}$. In contrast, however,A scalar reward signal is used to evaluate these pairs through trial and error, aiming to find the optimal prediction for each input. This approach is employed in domains where systems must adapt to environmental changes, and is thus applicable to problems ranging from control systems to game playing.

There several machine learning methods, and deep learning is one of them, but we will focus on its use for supervised classification tasks, precisely exploiting the k nearest neighbors algorithm. For more in depth introduction to machine learning refer to [2].

2.2.1 Supervised Classification

Classification is the process of assigning a category to a new sample based on its likelihood of belonging to that category. The goal is to learn a mapping from inputs $\boldsymbol{x} \in \mathcal{X}$ to a label $l_{\boldsymbol{x}}$ representing a category, using a labeled training dataset $T = \bigcup_{i=1}^{n}\{(\boldsymbol{x}^{(i)}, l^{(i)})\}$. This task can be formalized as approximating an unknown function $f : \mathcal{X} \rightarrow \wp(\mathcal{L})$, which perfectly predicts the label for a given input (i.e.,$f(\boldsymbol{x}^{(i)}) = l^{(i)} \ \forall \ (\boldsymbol{x}^{(i)}, l^{(i)}) \in T$). This is achieved by selecting a function $h : \mathcal{X} \rightarrow \wp(\mathcal{L})$ from the hypothesis space, which encompasses all possible functions, such that h approximates f as closely as possible on the training data. The number of possible outputs determines the type of supervised classification: *binary classification* occurs when each data instance can be assigned to one of two possible class labels; *multiclass classification* involves more than two class labels, with each instance assigned to only one. *Multilabel classification* addresses cases where a single example can belong to multiple classes. However, this thesis focuses solely on binary and multiclass classification.

2.2.2 K-NN Classifiers

One of the simplest and most trivial forms of learning for classification tasks is rote learning [18], which involves memorizing the entire training data and performing classification only if the attributes of the test sample exactly match those of a training example. This approach has two significant drawbacks: many test samples will remain unclassified due to the lack of an exact match with any training example, and when two or more training examples have identical attributes but different class labels, it becomes impossible to infer the correct label. A possible solution to mitigate these issues is to employ a more refined approach based on similarity rather than strict equality.

The k-nearest neighbors algorithm is a *non-parametric* supervised learning method that leverages data similarity to make classifications or predictions about the grouping of an individual sample, operating under the assumption that similar data should be classified similarly. The term 'non-parametric' indicates that no assumptions are made about the underlying data distribution from which the dataset is sampled. Consequently, the model structure is uniquely determined by the dataset itself, which is highly advantageous in practice, as most real-world datasets do not conform to predefined mathematical models. A k-NN classifier assigns an unknown feature vector $\boldsymbol{x} \in \mathbb{R}^n$ into the class $y_{\boldsymbol{x}}$ to which the majority of its k nearest neighbors belong, thus preventing the algorithm from failing when there is no exact match. In its simplest form, the 1-NN classifier, it assigns a test sample the class of its closest neighbor. The number $k \in \mathbb{N}$ of neighbors, as well as the similarity metric used to compare vectors in $\mathbb{R}^n$, are hyperparameters of this prediction model. The most commonly used similarity metrics are

- **Euclidean distance** (i.e. ℓ_2 norm): This metric computes the straight-line distance between two points of a vector space, and suitable for data that has continuous and numerical attributes with similar scales and ranges;

- **Manhattan distance** (i.e. ℓ_1 norm): This metric computes the distance between two points of a vector space in a grid-like path, and is preferred for data that has discrete and categorical attributes;

- **Cosine similarity**: This metric calculates the cosine of the angle between two vectors of a vector space, and it excels in sparse, high-dimensional data like text or images, focusing on direction rather than magnitude.

As Sebastian Raschka pointed out [43], although technically *plurality voting* is used, the term *majority voting* is almost ubiquitous in literature. The difference between these two terms is that "majority voting" requires a majority of more than 50%, which only works when there are only two classes. When there are multiple classes, e.g. four categories, it is not always necessary a 50% of the vote to make a decision about a class, and it is in fact possible to assign a class label with a vote of more than 25%.

The left picture in Figure 2.2 illustrates a classification performed using a k-NN model with $k = 3$ and Euclidean distance as the similarity metric, applied to a dataset in $\mathbb{R}^2$ with three classes: *red*, *green*, and *blue*. By applying the 3-NN algorithm to every vector in the input space, the right picture depicts the dataset after classification. For an unknown input vector, represented by a gray dot in the left picture, the algorithm computes the 3 nearest samples in the dataset, which are those within the dashed circle, and then infers the most common label among them. Following this strategy, it's possible that two or more labels receive an equal number of votes, resulting in a tie. For example, replacing a red point inside the dashed circle with a blue point would create a tie, as each label would receive exactly 1 vote.

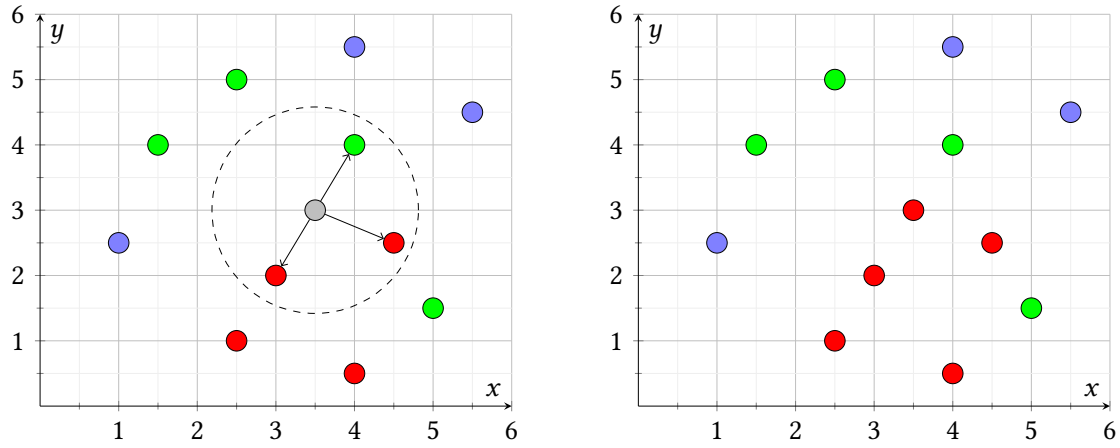


Figure 2.2: 3NN over a dataset with 3 classes, which shows how a new point is classified (left), as well as its label after being classified (right).

As can be seen from the example above, a k-NN model does not need a learning phase, which is instead required in most supervised ML algorithms, because all the examples are stored and entirely used at classification time, a feature that makes k-NN a so-called *lazy* (or *just-in-time*) learning algorithm [8]. While this makes it quite simple to implement, it can potentially result in a high computation time due to the effort of

computing and sorting the distances, especially when so many neighbors must be found. For this reason, k is usually a low value, very often below 9 and in any case always smaller than the square root of the total number of samples in the dataset. Being lazy, on the other hand, makes it perform well in many situations. Under certain reasonable assumptions, a well-known result by Cover and Hart [16] shows that the classification error of the nearest neighbor rule is bounded above by twice the optimal Bayes error. Furthermore, the error of the general k-NN method approaches that of the Bayes error asymptotically and can be used to approximate it.

Algorithm 2.1 provides a high-level summary of the k-NN algorithm for classification tasks. Given a ground truth dataset $T = \{(\boldsymbol{x}_1, l_1), \ldots, (\boldsymbol{x}_N, l_N)\} \subseteq X \times L$, a number of neighbors $k \in \mathbb{N} \setminus \{0\}$, and a distance function $\delta \colon X \times X \to \mathbb{R}_{\geq 0}$, a k-NN classifier is modeled as a total function $C_{T,k,\delta} \colon X \to \wp(L)$, which maps an input sample $\boldsymbol{x} \in X$ into a nonempty set of labels, by first selecting the similarly k samples in D to the input $\boldsymbol{x}$ according to δ metric, and then computing the set of their most frequent labels. Since a tie vote means to output a set including more than one label, we consider sets of labels as co-domain of classifiers.

Algorithm 2.1 k-NN algorithm

Input: D: the dataset, δ: the similarity metric, $\boldsymbol{x}$: The input sample
Output: The possible classifications of the input samples

1: $M, O \leftarrow \varnothing$
2: **for all** $(\boldsymbol{y}, l_{\boldsymbol{y}}) \in D$ **do**
3: $\quad d \leftarrow \delta(\boldsymbol{x}, \boldsymbol{y})$
4: $\quad O \leftarrow O \cup (d, l\boldsymbol{y})$
5: O.Sort()
6: **for all** $i \in \{1, \ldots, k\}$ **do**
7: $\quad M[i] \leftarrow O$.Extract$[i]$
8: **return** $\arg\max_{l \in L} \sum_{(\boldsymbol{y}, l_{\boldsymbol{y}}) \in M \text{ s.t. } l = l_{\boldsymbol{y}}} 1$

2.2.3 Adversarial Examples

Adversarial examples in machine learning are inputs to a model that are intentionally designed to cause the algorithm make mistakes in its predictions, yet appearing to be a legitimate input to a human. In this thesis, we will look at these kinds of inputs in the

context of k-NN classifiers. Given a classifier $C\colon X \to \wp(L)$, adversarial examples can be formally defined as inputs $\bar{\boldsymbol{x}}$ where the difference between $\bar{\boldsymbol{x}}$ and non-adversarial inputs $\boldsymbol{x}$ is minimal under some distance metric $\delta\colon X \times X \to \mathbb{R}_{\geq 0}$, but enough to make C infer a different output. To obtain such an $\bar{\boldsymbol{x}}$, a *perturbation* P is applied to $\boldsymbol{x}$ to cause a variation in the feature values of $\boldsymbol{x}$, defining a potential adversarial region $P(\boldsymbol{x}) \subseteq X$ in which $\bar{\boldsymbol{x}}$ belong. Generally, adversarial examples attempt to satisfy:

$$0 < \delta(\boldsymbol{x}, \bar{\boldsymbol{x}}) \leq \epsilon \text{ such that } C(\boldsymbol{x}) \neq C(\bar{\boldsymbol{x}})$$

where $\epsilon \in \mathbb{R}$ is a (small) constant bounding the magnitude of the perturbation.

Resilience to adversarial perturbations is a key property of a robust machine learning model. Ideally, the classifier should maintain consistent decisions despite minor variations in the input features. While this property is often assumed, it is not always guaranteed. For example, when the distance function is overly simplistic, k-NN can be significantly affected by adversarial perturbations, especially when dealing with feature vectors that have limited precision. In digital images, for instance, each pixel is typically represented with only 8 bits, with values from 0 to 255, discarding all information below $1/255$ of the dynamic range. As a result, if we consider a perturbation $\tau > 0$ smaller than this threshold, it would be unrealistic for the classifier to differentiate between an input vector $\boldsymbol{x}$ and a perturbed vector $\bar{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{y}$, as long as every element of $\boldsymbol{y}$ is smaller than or equal to τ. However, if the distance between vectors is computed by, for example, summing their feature-wise differences, as happens using the Manhattan distance, then $\delta(\boldsymbol{x}, \boldsymbol{x} + \boldsymbol{y})$ is the same whether we add $n\tau$ to the first feature or distribute τ across all features of $\boldsymbol{x}$. This behavior is problematic because, despite the differences in how perturbations affect the features, the distance metric treats them as equally significant, leading to potentially misleading results.

In this study, we model the adversarial region using the well-known ℓ_∞-perturbation [11], which uniformly affects all features. Given an input vector $\boldsymbol{x} \in \mathbb{R}^n$ and a perturbation magnitude $\epsilon \geq 0$, the ℓ_∞-perturbation defines the adversarial region as $P^{\epsilon}_{\infty}(\boldsymbol{x}) \triangleq \{\boldsymbol{w} \in \mathbb{R}^n \mid \max(|\boldsymbol{w}_1 - \boldsymbol{x}_1|, \ldots, |\boldsymbol{w}_n - \boldsymbol{x}_n|) \leq \epsilon\}$, which represents the ℓ_∞-ball of radius ϵ centered at $\boldsymbol{x}$. This approach accounts for all possible combinations of perturbed features, where each feature of the input sample can be perturbed up to the magnitude ϵ.

2.2.4 Stability and Robustness

Classifiers are usually evaluated and compared through multiple metrics. A simple and intuitive metric is *accuracy* on a test set: given ground truth test dataset $T = \{(\boldsymbol{x}_1, l_1), \ldots, (\boldsymbol{x}_n, l_n)\} \subseteq \mathcal{X} \times \mathcal{L}$, the accuracy of a classifier $C\colon X \to \wp(L)$ on T is defined by the ratio:

$$\textsc{Accuracy}(C, T) \triangleq \frac{|\{(\boldsymbol{x}, l_{\boldsymbol{x}}) \in T \mid C(\boldsymbol{x}) = \{l_{\boldsymbol{x}}\}|}{|T|} \tag{2.1}$$

that is the proportion of samples with correct predictions to the total number of samples. In adversarial scenarios, assessing a classification model solely on this standard metric is far from adequate. Although accuracy is useful in assessing the model's overall performance, it does not highlight any safety concerns. We now define two relevant properties in this context, which will be formally certified by our method.

Definition 2.1 (Stability). A classifier $C\colon X \to \wp(L)$ is *stable* on an input $\boldsymbol{x} \in X$ for a given perturbation $P\colon X \to \wp(X)$, denoted by $\textsc{Stable}(C, P, \boldsymbol{x})$, when $\forall \bar{\boldsymbol{x}} \in P(\boldsymbol{x}).\, C(\bar{\boldsymbol{x}}) = \{l\}$ holds for some $l \in L$.

Definition 2.2 (Robustness). A classifier $C\colon X \to \wp(L)$ is *robust* on an input $(\boldsymbol{x}, l_{\boldsymbol{x}}) \in X \times L$ for a given perturbation $P\colon X \to \wp(X)$, denoted by $\textsc{Robust}(C, P, \boldsymbol{x}, l_{\boldsymbol{x}})$, when $\forall \bar{\boldsymbol{x}} \in P(\boldsymbol{x}).\ C(\bar{\boldsymbol{x}}) = \{l_{\boldsymbol{x}}\}$ holds.

Stability means that a classifier does not change its output on a region of similar inputs, and it is orthogonal to accuracy in the sense that it does not require prior knowledge of the ground truth labels. Robustness, on the other hand, requires the classifier to be stable and correct, which means that it must output the same label that the input has in the dataset to which it belongs. It should be noted that for null ℓ_∞-perturbations, i.e. with $\epsilon = 0$, the definitions of accuracy and robustness coincide, leading us to conclude that the latter property expresses accuracy in adversarial scenarios.

As we did in (2.1), we define the stability and robustness of C on some test set $T \subseteq X \times L$ by the ratios:

$$\begin{aligned} \textsc{Stability}(C, T) &\triangleq \frac{|\{(\boldsymbol{x}, l_{\boldsymbol{x}}) \in T \mid \textsc{Stable}(C, P, \boldsymbol{x})\}|}{|T|} \\ \textsc{Robustness}(C, T) &\triangleq \frac{|\{(\boldsymbol{x}, l_{\boldsymbol{x}}) \in T \mid \textsc{Robust}(C, P, \boldsymbol{x}, l_{\boldsymbol{x}})\}}{|T|} \end{aligned} \tag{2.2}$$

It is worth remarking that if we have a tie vote using the k-NN algorithm, $|C(\boldsymbol{x})| > 1$ always holds, and hence C can be neither stable nor robust on $\boldsymbol{x}$ by definition.

2.2.5 Individual Fairness

When it comes to *individual fairness* of ML classifiers, as the name suggests, we want to know if similar inputs will receive a similar class label. The similarity relation on the input space X is expressed in terms of a distance δ and a threshold $\epsilon > 0$ by considering $S_{\delta,\epsilon} \triangleq \{(\boldsymbol{x}, \boldsymbol{y}) \in X \times X \mid \delta(\boldsymbol{x}, \boldsymbol{y}) \leq \epsilon\}$. According to Dwork's work [19], a model is *biased* (or *unfair*) if there is a pair of valid inputs that are close to each other for some distance function δ, but are treated differently by the model, leading to a different outcome, and it is instead *unbiased* (or *fair*) if such a pair does not exist. Consequently, given an input $\boldsymbol{x} \in X$, we say that a classifier $C : X \to \wp(L)$ is fair on $\boldsymbol{x}$ with respect to $S_{\delta,\epsilon}$ when:

$$\forall \boldsymbol{y} \in X.\ (\boldsymbol{x}, \boldsymbol{y}) \in S_{\delta,\epsilon} \Rightarrow C(\boldsymbol{x}) = C(\boldsymbol{y}).$$

Following the above consideration, we now formally define the individual fairness property in adversarial scenarios.

Definition 2.3 (Individual Fairness). Given an input $\boldsymbol{x} \in X$ and perturbation $P \colon X \to \wp(X)$ such that $P_{\delta,\epsilon}(\boldsymbol{x}) \triangleq \{\boldsymbol{y} \in X \mid \delta(\boldsymbol{x}, \boldsymbol{y})\}$ for some distance function δ, a classifier $C : X \to \wp(L)$ is *fair* on $(\boldsymbol{x}, l_{\boldsymbol{x}})$ with respect to $P_{\delta,\epsilon}$, denoted by $\textsc{Fair}(C, P_{\delta,\epsilon}, \boldsymbol{x})$, when $\forall \bar{\boldsymbol{x}} \in P_{\delta,\epsilon}(\boldsymbol{x}).\ C(\boldsymbol{x}) = C(\bar{\boldsymbol{x}})$ holds.

By leveraging on Definition 2.3, we observe that individual fairness boils down to stability, namely, for all inputs $\boldsymbol{x}$, $\textsc{Fair}(C, P_{\delta,\epsilon}, \boldsymbol{x}) \Leftrightarrow \textsc{Stable}(C, P_{\delta,\epsilon}, \boldsymbol{x})$ holds. For the individual fairness metric on a given test set $T \subseteq X \times L$ we therefore have that:

$$\textsc{fairness}(C, T) \triangleq \frac{|\{(\boldsymbol{x}, l_{\boldsymbol{x}}) \in T \mid \textsc{Fair}(C, P_{\delta,\epsilon}, \boldsymbol{x})\}|}{|T|} \tag{2.3}$$

In our experiments, we have considered the Noise-Cat similarity relation as defined in [44], where two inputs $\boldsymbol{x}, \boldsymbol{y} \in X$ are similar when: **(i)** given a subset $N \subseteq \mathbb{N}$ of indexes of numerical features and a threshold $\epsilon \in \mathbb{R}_{\geq 0}$, for all $i \in N$, $|\boldsymbol{x}_i - \boldsymbol{y}_i| \leq \epsilon$; **(ii)** given a subset $C \subseteq \mathbb{N}$ of indexes of categorical features, both $\boldsymbol{x}$ and $\boldsymbol{y}$ are allowed to have any category for features with indexes in C; **(iii)** every other feature of $\boldsymbol{x}$ and $\boldsymbol{y}$, i.e. with index not in N nor C, must be the same, namely, for any index $i \notin (N \cup C)$, $\boldsymbol{x}_i = \boldsymbol{y}_i$

holds. For example, if we wanted to make a classification for statistical purposes, two individuals having two numerical features *age* and *height* and two categorical features *gender* and *country*, could be considered similar if their ages are both within the same reference range (e.g. in the age group 25-30) regardless of their country, while having the same gender and height.

3

Related Works

Adversarial robustness regained significant research attention after it was demonstrated that non-linear classifiers, such as support vector machines with the kernel trick or neural networks, are not robust to adversarial examples [6, 5, 55]. Consequently, robustness certification of ML models (i.e., the theoretical assurance that a model remains unaffected by perturbations within a specified bound) emerged as a prominent research topic. Several techniques, such as interval bound propagation (IBP) [25, 64] and abstract interpretation [39, 9, 23, 34, 35, 40, 41, 42, 48, 49, 50, 57], have been proposed for certify the robustness. Although the majority of research has focused on neural network-based models, other major ML models particularly non-parametric ones like k-NN, have been only recently studied. Specifically for the k-nearest neighbor method, in [59] the authors, for the first time, studied the robustness property from theoretical point of view and showed that k-NN classifier can be as robust as the optimal Bayes classifier given a sufficiently large number of samples and k. Others have proposed various methodologies based on some minimization problems to find the smallest perturbations of a sample that changes its classification [58, 62, 52, 54, 51] and more recently in [20] abstract interpretation was used to verify the stability of k-NN given a perturbation of a test sample like we did in our research work.

This chapter will provide a brief overview of the aforementioned stream of works finding adversarial examples on k-NNs.

3.1 Theoretical analysis

Wang et al. [59] conducted a theoretical study on the robustness of k-NN to adversarial attacks. For simplicity, they focused on binary classification and Euclidean distance. Their analysis demonstrated that, given an input sample $\boldsymbol{x}$, if certain continuity assumptions hold within a neighborhood of $\boldsymbol{x}$, then when k is constant (i.e., independent of the training set size n and the input space dimensionality), k-NN is inherently non-robust as $n \to \infty$. This non-robustness is particularly evident in regions where $p(y = 1|x) \in (0, 1)$, indicating areas where samples with different labels are not clearly separated. On the other hand, if $k = \Omega(\sqrt{dn \log n})$ where d is the data dimension and n is the training set size, they demonstrated that, as $n \to \infty$, the robustness region of k-nearest neighbors approaches that of the Bayes Optimal classifier. Since $k = \Omega(\sqrt{dn \log n})$ is not a usable value in practice, they propose a novel method to make 1-nearest neighbor more robust to adversarial attacks. Following the observation that 1-nearest neighbor is robust when oppositely labeled points are far apart, their idea is to enforce this property by removing training samples x such that: (a) there is a lack of confidence about the label of x and its nearby points and (2) the closest points of x are not of the same label. After removing this points they executed the 1-nearest neighbor on the remaining dataset. Their experiments showed that their modified version performs better than or about as well as both standard 1-nearest neighbors and nearest neighbors with adversarial training.

3.2 QP-based analysis

In 2019, Wang et al. [58] studied the problem of evaluating the robustness of k-nearest neighbor classifiers, focusing more on 1NNs. They showed that finding the minimum adversarial perturbation can be formulated as a set of convex *quadratic programming* (*QP*) problems, with an exact solution in polynomial time for 1NNs. When applied to general k-NN models, however, the number of constraints in the QP formulation grows exponentially with k, becoming quickly infeasible, hence NP-hard. As a result, for $k > 1$, their method finds valid lower and upper bounds of the minimum adversarial perturbation. Given a classifier $C\colon X \to L$ and an input sample $(x, l) \in X \times L$, an adversarial perturbation is defined as $\boldsymbol{d} \in \mathbb{R}^n$ **s.t.** $C(\boldsymbol{x} + \boldsymbol{d}) \neq l$, and is the minimum if $\forall \boldsymbol{d'} \in \mathbb{R}^n.\ \|\boldsymbol{d'}\|_2 < \|\boldsymbol{d}\|_2 \Rightarrow C(\boldsymbol{x} + \boldsymbol{d'}) = l$. For instance, let $L = \{A, B\}$, $(x, A) \in X \times L$ and $k = 1$, the problem of finding the minimum perturbation such that $\boldsymbol{x} + \boldsymbol{d}$ is closer

to some $\boldsymbol{x_j}$ labeled with $l_j = B$ than to all class-A samples, can be formulated as the quadratic primal problem:

$$\epsilon^{(j)} \triangleq \arg\min_{\boldsymbol{d}} \frac{1}{2}\boldsymbol{d}^T\boldsymbol{d} \text{ s.t. } \|\boldsymbol{x}+\boldsymbol{d}-\boldsymbol{x_j}\|_2^2 \leq \|\boldsymbol{x}+\boldsymbol{d}-\boldsymbol{x_i}\|_2^2, \quad \forall i, l_i = A$$

Although it is simple to solve, by scaling to $k = 3$ it become necessary to list all the possible combinations of $\{(j_1, j_2, j_3) \mid l_{j_1} = l_{j_2} = B, l_{j_3} = A\}$ and then solve the QP primal problem to force $\boldsymbol{x} + \boldsymbol{d}$ to be closer to $\boldsymbol{x}_{j_1}$, $\boldsymbol{x}_{j_2}$ than to all class-A samples except $\boldsymbol{x}_{j_3}$, hence:

$$\epsilon^{(j_1,j_2,j_3)} \triangleq \arg\min_{\boldsymbol{d}} \frac{1}{2}\boldsymbol{d}^T\boldsymbol{d} \text{ s.t. } \|\boldsymbol{x}+\boldsymbol{d}-\boldsymbol{x_j}\|_2^2 \leq \|\boldsymbol{x}+\boldsymbol{d}-\boldsymbol{x_i}\|_2^2, \quad \forall i, i \neq j_3, l_i = A, j \in \{j_1, j_2\}$$

Thus, it is more expensive to solve. For general $k > 1$, the QP formulation will have $O(nk)$ constraints, but because of the sparsity of solution, greedy coordinate ascent can still solve a subproblem efficiently, computing an upper and lower bound, corresponding to attack and verification.

3.3 Region-based analysis

Back in 2020, Yang et al. [62] investigated adversarial examples for the most popular non-parametric classifiers, including k-nearest neighbors, decision trees and random forests. They devised a general attack technique, known as *region-based attack*, designed to work well for multiple non-parametrics. Since the difficulty in finding adversarial examples stems from the fact that these classifiers have complicated decision regions, the main idea behind this attack is to decompose the decision regions of many classifiers, such as k-NN or random forests, into convex sets. Precisely, they leverage the concept of (s, m)-decomposition, that is a partition of $\mathbb{R}^n$ into s convex polyhedra $P_1, \ldots, P_s$ such that each P_i can be described by up to m linear constraints. They formulated that a classifier is $C\colon X \to L$ (s, m)-decomposable when there is a (s, m)-decomposition such that C is constant on P_i for each $i \in [1, \ldots, s]$. In this context, given a classifier with a decomposition $P_1, \ldots, P_s$ such that $C(\boldsymbol{y}) = l_i$ when $\boldsymbol{y} \in P_i$ for labels $l_i \in L$, finding an adversarial example for an input sample $\boldsymbol{x} \in X$ requires outputting $\bar{\boldsymbol{x}}$ that minimizes::

$$\min_{i\colon C(\boldsymbol{x})\neq l_i} \min_{\boldsymbol{z}\in P_i} \|\boldsymbol{x}-\boldsymbol{z}\|_{p\in\{1,2,\infty\}}$$

Thus, solving the inner minimization problem results in candidates $\boldsymbol{z}^i \in P_i$. Taking then the outer minimum over i with $C(\boldsymbol{x}) \neq l_i$ leads to the optimal adversarial example:

$$\bar{\boldsymbol{x}} \triangleq \arg\min_{\boldsymbol{z}^i} \|\boldsymbol{x} - \boldsymbol{z}^i\|_{p\in\{1,2,\infty\}}$$

The exact attack algorithm's performance is determined by two factors: (i) the number of regions, which is determined by the complexity of the classifier, and (ii) the number of constraints and dimensionality of the polyhedra. For kNN classifiers the number of convex polyhedra scales with $O(n^k)$: when $k = 1$, this is efficiently solvable, because polyhedra have at most n constraints and the adversarial examples can be found quickly using a linear program for ℓ_∞-perturbations. Unfortunately, for $k > 1$, region-based attacks do not scale well, and an approximation algorithm for larger values of k is also discussed.

3.4 Gradient-based analysis

Inspired by their previous work [53] on the robustness of deep k-NN, in 2020 Sitawarin and Wagner [54] proposed a new state-of-the-art attack, called *gradient-based attack*, to evaluate the robustness of k-NN classifiers. Their method also outperforms Yang et al. [62] in that when $k > 1$, an adversarial example with a smaller perturbation is found in less than 1% of the running time. Moreover, increasing the value of k only slightly increases the algorithm's running time. In order to find the minimum $\boldsymbol{d}^* \in \mathbb{R}^n$ **s.t.** $\bar{\boldsymbol{x}} \triangleq \boldsymbol{x} + \boldsymbol{d}^*$ is an adversarial example classified differently than $\boldsymbol{x} \in X$, the following minimization problem is solved:

$$\boldsymbol{d}^* \triangleq \arg\min_{\boldsymbol{d}} \sum_{i=1}^{m} \max\left\{w_i \left(\|\bar{\boldsymbol{x}}_{\boldsymbol{i}} - (\boldsymbol{x} + \boldsymbol{d})\|_2^2 - \eta^2\right) + \Delta, 0\right\}$$

where m denote the mean of examples with different class closest to $\boldsymbol{x}$ in ℓ_2-norm, $w_i = 1$ if $\bar{\boldsymbol{x}}_{\boldsymbol{i}}$ is adversarial, otherwise $w_i = -1$, and η is the distance to the k-th nearest neighbor. The changes compared to the original version are in using the rectifier $\max(z \in \mathbb{R}, 0)$ instead of sigmoid, which avoids the need to deal with overflow and underflow issues caused by the exponential in distance computation, and the introduction of a small gap Δ to ensures that $\bar{\boldsymbol{x}}_{\boldsymbol{i}}$ is a bit closer to the guide samples from the incorrect class than

the ones from the correct class. Unfortunately gradient-based approaches generally do not work well for finding minimum ℓ_∞-norm adversarial examples, therefore only ℓ_2-norm is considered. Furthermore, this method does not guarantee the optimality of the solution, not even for $k = 1$.

3.5 Geometric-based analysis

GeoAdEx [51], which stands for *geometric adversarial example*, is the product of the most recent work finding adversarial examples on k-NNs. This novel tool was proposed by Sitawarin et al. back in 2021, and is the first using geometric approach to perform a search that expands outwards from the given input sample. The main idea of GeoAdEx is to perform a principled geometric exploration around the test sample by processing order-k Voronoi cells à la breadth-first search until it discovers an adversarial cell. Formally, the goal of this algorithm is to find the smallest perturbation $\boldsymbol{d}^* \in \mathbb{R}^n$ that moves a test point $(\boldsymbol{x}, l_{\boldsymbol{x}}) \in X \times L$, to an adversarial cell, that is an order-k Voronoi cell that has different majority than label $l_{\boldsymbol{x}}$. Such objective can be expressed as the following optimization problem:

$$\boldsymbol{d}^* \triangleq \arg\min_{\boldsymbol{d}} \|\delta\|_2^2 \quad \text{s.t. } \boldsymbol{x} + \boldsymbol{d} \in A(x)$$

where $A(x)$ is the set of all adversarial cells with respect to $(\boldsymbol{x}, l_{\boldsymbol{x}})$ and ℓ_2-norm. The above constraint implies that $\boldsymbol{x}+\boldsymbol{d}$ must be a member of an adversarial cell from $A(x)$. A simple approach to solve this minimization is to build a series of optimization problems, one for each of the cells in $A(x)$, and pick the solution with the minimum adversarial distance. Unfortunately, this would require solving $O(\binom{n}{k})$ QP problems, each of which has $k(n-k)$ constraints. While this complexity may be manageable when $k = 1$ and n is small, as it was in the other works, it does not scale well with k becoming quickly unusable. To deal with $k > 1$, the authors of this tool have added an approximated version of the algorithm which consists in bounding the number of neighboring cells taken into account according to a fast heuristic. However, the main drawback of this approach is that the approximation may affect the optimality, which is therefore no longer guaranteed. Overall this geometric-based attack finds an adversarial distance that is closer to the optimal than the baselines, but its main limitation is the excessively long runtime of the non-approximated version.

3.6 Abstract interpretation-based analysis

Fassina et al. [20] proposed novel formal and automatic verification method, implemented in a tool called kNAVe (*k*NN **A**bstract **Ve**rifier), to automatically verify when k-NN is provably stable for an input sample with respect to a given perturbation. Their approach is based upon the well-established framework of abstract interpretation [14], a formal verification method used to statically analyze the dynamic behavior of a system. They first developed a theoretical sound approximation of the k-NN algorithm and the similarity metric δ used by the classifier that are agnostic with respect to a symbolic numerical abstraction A capable of representing input space properties, namely sets of vectors in $\wp(\mathbb{R}^n)$, and basic numerical operations such as addition, product, and modulus. Given an abstract value $a \in A$ which provides a symbolic over-approximation of an adversarial perturbation $P(x)$ of an input sample x modeled as the ℓ_∞-ball of radius ϵ centered in x, this approximate classifier $C^A_{k,\delta}$ returns an over-approximation O of the set of labels computed by k-NN for all the samples within $P(x)$. Therefore, if O is a singleton set consisting of the label predict by k-NN for x then it can be inferred that k-NN is provably stable on x for its perturbation $P(x)$. Subsequently, they performed empirical evaluation of the abstract classifier by instantiating it with two numerical abstract domains to approximate the range of numerical features: (1) the interval domain [15] in which abstract values are closed real intervals (e.g. $x_i \in [l, u]$) and (2) the zonotope domain [13] where the abstract values are affine forms, a first degree polynomial over symbolic noise variable assumed to lie in the interval $[-1, 1]$ (e.g. $x_i = a_0 + \sum_{j=1}^{k} a_j \epsilon_j$ given $a_j \in \mathbb{R}$ and $\epsilon_j \in [-1, 2]$). In their experiment they showed that k-NN is a robust classification algorithm since for adversarial perturbations of $\leq \pm 2\%$, kNAVe was able to infer for several datasets more than 90% of robustness for $k \in 1, 3, 5, 7$.

One limitation of kNAVe is that the abstract classifier is not complete, i.e., the over-approximation O does not always equal the set of labels computed by k-NN for all samples within $P(x)$. Consequently, there are test samples where the abstract classifier cannot prove stability, even though the input samples are genuinely stable under the perturbation. In contrast, the method proposed in this research is exact—it can prove stability for all inputs that are genuinely stable. As shown in Chapter 6, our method outperforms kNAVe across all datasets for the same values of k.

4
Methodology

As discussed in Chapter 3, one limitation of the method proposed in [20] is that it is sound but not complete. This incompleteness arises in part from the computation of distances between training samples and the perturbation region in the abstract domain, which is not always exact. This in turn hinders precise inference of the possible nearest neighbors of each sample within the perturbation region.

To address this distance-related issue, our methodology modifies the standard k-NN algorithm. Instead of sorting samples by their distance to the input sample and then classify the input with the most frequent label among the first k samples, we first construct a graph where nodes represent training samples and edges model the *closer-to-input* relation (i.e., an edge from s_1 to s_2 indicates that s_1 is closer to the input than s_2 with respect to Euclidean distance). We then classify the input with the most frequent labels within samples that constitute specific paths in the graph having length $k-1$, and satisfying relative proximity constraints. To determine the *closer-to-input* relationship between two samples s_1 and s_2, we can simply check on which side of the perpendicular bisector between them, the input sample is located. In this way we can select the k nearest neighbors without explicitly computing the distances between the input and training samples. We termed this new algorithm G-kNN that stands for Graph-k-NN emphasizing the fact that it utilizes a graph structure. A similar approach is used to certify the robustness of k-NN to an adversarial region of an input sample. We construct a graph whose nodes again are the samples of the training set but the edges model the

closer-to-region relation (i.e., an edge from s_1 to s_2 indicates that there exists a point in the adversarial region such that s_1 is closer to it than s_2) and then, for each label l, search for a path in the graph: (1) having length $k-1$, (2) l being the most frequent label among the samples comprising the path and (3) satisfying some validity constraints. If such path is found then the corresponding label is added to the output set otherwise is ignored. To the best of our knowledge this is a novel approach which has not been explored previously.

This chapter illustrates the details of this novel approach by first explaining the new k-NN algorithm and then how the robustness to an adversarial region is inferred.

4.1 G-kNN Algorithm

The main difference with the standard k-NN algorithm is the way in which the k nearest neighbors are selected. The standard k-NN algorithm select the nearest neighbors by using the distance to the input sample. In contrast, G-kNN utilizes the relation of being closer to the input between pair of samples w.r.t. the Euclidean distance. This relation is defined as follows

Definition 4.1 (closer-to-input relation $\preceq_{\boldsymbol{x}}$). Given the points $\boldsymbol{x}, \boldsymbol{s}_1, \boldsymbol{s}_2 \in \mathbb{R}^n$

$$\boldsymbol{s}_1 \preccurlyeq_{\boldsymbol{x}} \boldsymbol{s}_2 \iff \|\boldsymbol{x} - \boldsymbol{s}_1\| \leq \|\boldsymbol{x} - \boldsymbol{s}_2\|$$

To determine if $\boldsymbol{s}_1 \preccurlyeq_{\boldsymbol{x}} \boldsymbol{s}_2$ holds we can check if the input sample $\boldsymbol{x}$ and $\boldsymbol{s}_1$ reside in the same half space induces by the perpendicular bisector of $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ as shown in Figure 4.1.

The G-kNN algorithm first builds a graph $\mathbb{G}$ in which nodes represents the samples in the training set and the edges model the relation $\preccurlyeq_{\boldsymbol{x}}$ (i.e., the tail is closer to the input than the head) and then classify the input sample with the most frequent labels within samples that constitute specific paths in the graph having length $k-1$ and satisfying some validity constraints that will be detailed in Subsection 4.1.2. We termed the graph $\mathbb{G}$ *proximity precedence* graph (PP-G) of the sample x emphasizing the fact that the precedence of the vertices in any path comes from their proximity to x.

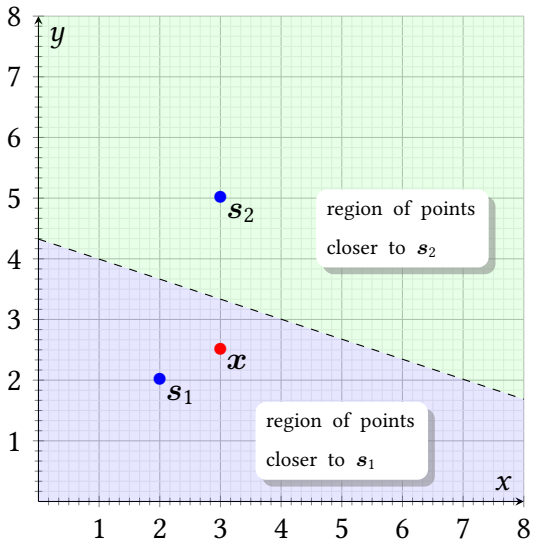


Figure 4.1: Example showing how to determine if $s_1 \preccurlyeq_x s_2$. Since the input sample is in the same half space where s_1 is located it means that the relation $s_1 \preccurlyeq_x s_2$ holds.

4.1.1 Proximity Precedence Graph

Definition 4.2. Given the training set $\mathcal{S} \subset \mathcal{Z}$ and an input sample $x \in \mathbb{R}^n$, the proximity precedence graph $\mathbb{G}_x^S = (V, E)$ of x is the graphical representation of the totally ordered set $(\mathcal{S}_X, \preccurlyeq_x)$ which means that:

- $V = \mathcal{S}_X$
- $E = \{(x_i, x_j) \mid x_i \preccurlyeq_x x_j \text{ holds}\}$

For ease of notation, in the following of this document the superscript $\mathcal{S}$ will be dropped as a training set $\mathcal{S}$ is always assumed.

The graph is represented using adjacent lists. Given a vertex v of the graph:

- $\texttt{adj}(v) = \{v_i \in V \mid (v, v_i) \in E\}$ denotes the set of adjacent vertices of v;
- $\texttt{pred}(v) = \{v_i \in V \mid (v_i, v) \in E \wedge (v, v_i) \notin E\}$ denotes the set of vertices v_j such there is an edge from v_i to v but not the other way around which means

$$v_i \preccurlyeq_x v \ \wedge v \not\preccurlyeq_x v_i$$

The set $\texttt{pred}(v)$ will be called the *predecessors* of v.

- $\texttt{same\_dist}(v) = \{v_i \in V \mid (v_i, v) \in E \wedge (v, v_i) \in E\}$ denotes the set of vertices v_i such that there is a bidirectional edge between v_i and v which means

$$v_i \preccurlyeq_{\boldsymbol{x}} v \ \wedge v \preccurlyeq_{\boldsymbol{x}} v_i$$

Given this definition, $\texttt{pred}(\boldsymbol{s}_i)$ include all the samples strictly closer to the input $\boldsymbol{x}$ than $\boldsymbol{s}_i$ and $\texttt{same\_dist}(\boldsymbol{s}_i)$ contains those that are exactly the same distance to the input as $\boldsymbol{s}_i$.

Algorithm 4.1 shows how the proximity precedence graph is created. Given the dataset S and the input sample $\boldsymbol{x}$, it first creates the proximity precedence graph $\mathbb{G}$ with only the vertices having no adjacent vertices (line 1-2). Then for each order pair of vertices $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ (line 3-7) check if $\boldsymbol{x}_i \preccurlyeq_{\boldsymbol{x}} \boldsymbol{x}_j$ holds (line 4) and if so add the $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ to the set of edges (line 5-6) and finally returns the constructed graph (line 8).

Algorithm 4.1 CreatePPGraph method

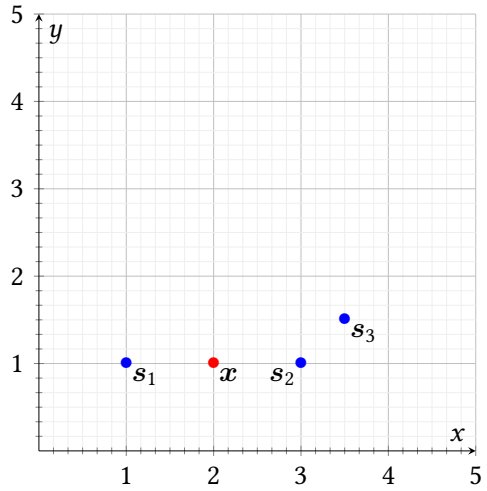
Input: S: A training set, $\boldsymbol{x}$: The input sample
Output: The proximity precedence graph of $\boldsymbol{x}$

1: $V \leftarrow S_X$
2: $E \leftarrow \emptyset$
3: **for all** $(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \{(\boldsymbol{x}_i, \boldsymbol{x}_j) \mid \boldsymbol{x}_i, \boldsymbol{x}_j \in V, \boldsymbol{x}_i \neq \boldsymbol{x}_j\}$ **do**
4: **if** $\boldsymbol{x}_i \preccurlyeq_{\boldsymbol{x}} \boldsymbol{x}_j$
5: $E \leftarrow E \cup \{(\boldsymbol{x}_i, \boldsymbol{x}_j)\}$
6: **return** (V, E)

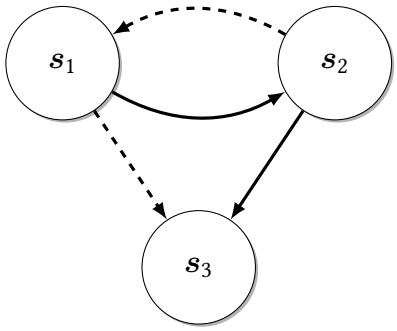
Example 4.1. Consider a dataset $\mathcal{S} \subset \mathbb{R}^2$ and the input sample $\boldsymbol{x} \in \mathbb{R}^2$ as shown in Figure 4.2a. In this scenario the samples $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ are equidistant from $\boldsymbol{x}$ while $\boldsymbol{s}_3$ is the furthest one. So in the PP-G of $\boldsymbol{x}$, as shown in Figure 4.2b, $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ are adjacent of each other, and $\boldsymbol{s}_3$ has no adjacent vertices since it is the furthest.

4.1.2 Nearest Neighbors Selection

After constructing the PP-G $\mathbb{G}_{\boldsymbol{x}}$ of the input, the G-kNN identifies the k nearest neighboring samples to the input sample to predict its label. These neighbors will form a path of length $k-1$ within $\mathbb{G}_{\boldsymbol{x}}$ where, by definition of a proximity precedence graph, the first sample in this path is the closest one to the input, the second sample is the sec-



(a) Graph showing the dataset $\mathcal{S}$ and input sample of Example 4.1



(b) PP-G of input sample of Example 4.1

Figure 4.2: Example showing the PP-G of the input sample of Example 4.1

ond closest one and so on. This means not all paths of length $k-1$ contains the k nearest samples but only those that are coherent with the proximity relations defined among the samples within the dataset $\mathcal{S}$ that is if $\boldsymbol{x}_1 \preccurlyeq_{\boldsymbol{x}} \boldsymbol{x}_2$ but $\boldsymbol{x}_2 \not\preccurlyeq_{\boldsymbol{x}} \boldsymbol{x}_1$ then every path in which $\boldsymbol{x}_2$ is present must also contain $\boldsymbol{x}_1$ preceding $\boldsymbol{x}_2$. For instance in the PP-G of Example 4.1 the path $[\boldsymbol{s}_2, \boldsymbol{s}_3]$ is not a valid path because $\boldsymbol{s}_1 \preccurlyeq_{\boldsymbol{x}} \boldsymbol{s}_3$ but $\boldsymbol{s}_3 \not\preccurlyeq_{\boldsymbol{x}} \boldsymbol{s}_1$, but the path does not contain the sample $\boldsymbol{s}_1$. This observation leads to the following definition for a valid path:

Definition 4.3 (Valid path). A path $\mathcal{P}$ within a PP-G is defined to be *valid* if and only if

$$\forall \boldsymbol{x}_i \in \mathcal{P}.\ \forall \boldsymbol{x}_j \in \texttt{pred}(\boldsymbol{x}_i).\ \ \boldsymbol{x}_j \text{ is a predecessor of } \boldsymbol{x}_i$$

So, by this definition, in the PP-G of Example 4.1 only the edges with the same stroke form valid paths which are:$[\boldsymbol{s}_1]$, $[\boldsymbol{s}_2]$, $[\boldsymbol{s}_1, \boldsymbol{s}_2]$, $[\boldsymbol{s}_2, \boldsymbol{s}_1]$, $[\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3]$ and $[\boldsymbol{s}_2, \boldsymbol{s}_1, \boldsymbol{s}_3]$

Proposition 4.1. Every valid path in a PP-G of $\boldsymbol{x}$ starts with samples $\boldsymbol{x}_0$ closest to the input $\boldsymbol{x}$.

Proof. By Definition 4.3 every predecessor of samples in valid path must precede them so the first sample $\boldsymbol{x}_0$ is such that $\texttt{pred}(\boldsymbol{x}_0) = \emptyset$ which happens only for the closest samples to the input sample $\boldsymbol{x}$. □

Proposition 4.2. Let $s_i \in \mathcal{S}$ be a sample of the training set such that: (1) it does not occur in a valid path $\mathcal{P} = [s_0, s_1, \ldots, s_{i-1}]$, (2) every sample in $\texttt{pred}(x_i)$ is present in $\mathcal{P}$ and (3) is an adjacent of s_{i-1}. Then the path $\mathcal{P}' = \mathcal{P} + [s_i] = [s_0, s_1, \ldots, s_{i-1}, s_i]$ is a valid path.

Proof. By hypothesis, every predecessor of s_i is present in $\mathcal{P}$ which is already a valid path and so the path $\mathcal{P}'$ satisfy the condition to be a valid path. □

Definition 4.4 (Safe sample). Given a valid path $\mathcal{P} = [s_0, s_1, \ldots, s_{i-1}]$, the sample s_i is defined to be *safe* for $\mathcal{P}$ if the path $\mathcal{P} + [s_i] = [s_0, s_1, \ldots, s_{i-1}, s_i]$ is a valid path.

By Proposition 4.2, a sample is safe for a valid path if all its predecessors are present in the path.

Proposition 4.3. Given $k \in \mathbb{N}, k \geq 1$, an input sample $x \in \mathbb{R}^n$ and the PP-G $\mathbb{G}_x$ of x, a valid path $\mathcal{P}$ of length $k-1$ contains the closest k samples to the input x.

Proof. We prove the proposition by induction on the length $k-1$ of the path:

- **Base Case** ($k = 1$): In this case $\mathcal{P}$ is made of only one sample which, by Proposition 4.1, is the closest sample to the input x.
- **Inductive Case** ($k = h + 1, h \geq 1$): The path $\mathcal{P}$ can be expressed as the concatenation of a valid sub-path $\mathcal{P}'$ and the single sample s_h (i.e., $\mathcal{P} = \mathcal{P}' + [s_h]$). By the induction hypothesis, the $\mathcal{P}'$ contains the h samples closest to the input x. Because $\mathcal{P}$ is valid, s_h is a safe sample for $\mathcal{P}'$ (by Definition Definition 4.4). This implies that all samples closer to x than s_h are already included in $\mathcal{P}'$. Furthermore, according to the definition of the proximity precedence graph, any sample s_j in $\mathcal{P}'$ that is not a predecessor of s_h must be equidistant from the input x as s_h (i.e., $s_j \in \texttt{same\_dist}(s_h)$). Therefore, $\mathcal{P}'$ contains all the samples that are either closer to or equidistant from x compared to s_h, and hence the combined path $\mathcal{P}' + [s_h]$ contains the $h + 1$ samples closest to the input x.

□

The generation of valid paths of length $k-1$ is done by traversing the graph using a Breadth-First Search (BFS) approach, while adhering to path validity constraints. Algorithm 4.2 details the extraction of valid paths composed of n samples from a given PP-G $\mathbb{G}_x$ of the input sample. The algorithm employs a FIFO queue (the variable `queue`) to maintain the list of all valid paths containing $k \leq n$ samples. It starts by initializing

the queue with the samples s_i such that $\texttt{pred}(s_i) = \emptyset$ (lines 1) effectively starting the traversal from samples closest to the input x. A counter `k` which denotes the number of samples within the paths in the queue is initialized to 1 (line 2). The algorithm then iterates as long as the queue is not empty (line 3-17). In each iteration, it dequeues all existing paths and checks if they contain the desired number of samples (i.e. the input n). If so, these paths are returned (lines 5-7). Otherwise, each dequeued path is extended by appending every safe adjacent vertex of the path's last sample. These extended paths are then enqueued for the next iteration (lines 8-15). Before the next iteration, the counter `k` is incremented to reflect the increased number of samples in the paths within the queue (line 16)

Algorithm 4.2 ExtractValidPath method

Input: $\mathbb{G}_x$: PP-G of the input sample
n: number of samples in the valid path
Output: Set of all valid paths comprised of n samples

```
1:  queue ← {[s_i] | s_i ∈ G_x ∧ pred(s_i) = ∅}
2:  k ← 1
3:  while queue not empty do
4:      current_paths ← queue.PopAll()
5:      if k = n
6:          return current_paths
7:      for all path in current_paths do
8:          last_sample ← path.last()
9:          for all sample in adj(last_sample) do
10:             if sample is safe for path
11:                 queue.append(path + [sample])
12:     k ← k + 1
```

Proposition 4.4. Given as input the PP-G $\mathbb{G}_x$ and $n \in \mathbb{N}$, Algorithm 4.2 returns all the valid paths within $\mathbb{G}_x$ containing n samples.

Proof. The proposition can be proved by showing, that at the k-th iteration of the while loop, at line 5 (i.e., before the check on the number of sample in the paths extracted from the queue), the queue contains all the valid path of comprised of k samples. We proceed by induction on k:

- **Base Case** ($k = 1$): In the first iteration, the queue contains all paths consisting of a single sample s_i such that$\texttt{pred}(s_i) = \emptyset$. These paths are valid by definition and have length 1.

- **Inductive Case** ($k = h + 1, h \geq 1$): At iteration $h + 1$, the paths inside the queue at line 5 are obtained by extending all the paths in the queue in the h-th iteration with every safe vertex among the adjacent of the path's last sample. By induction hypothesis the queue in the h-th iteration contains all the valid paths containing h samples and so in the $h + 1$ iteration, by definition of safe vertex, the queue contains all the valid paths with $h + 1$ samples.

□

4.1.3 The Full Algorithm

Algorithm 4.3 illustrates the full algorithm of G-kNN. Given a set of samples $\mathcal{S}$ and the input sample $\boldsymbol{x}$ the algorithm first build the PP-G $\mathbb{G}_{\boldsymbol{x}}$ of the input $\boldsymbol{x}$ (line 1). Then the extract all the valid path within $\mathbb{G}_{\boldsymbol{x}}$ with k samples using the `extract_valid_path` method (line 2). Afterward, the set containing the most frequent labels within the samples composing each extracted path is constructed and returned as the possible classifications of the input sample (line 3-8).

Since there could be multiple valid paths in the graph due to samples in $\mathcal{S}$ equidistant to the input $\boldsymbol{x}$, the latter could be classified with different labels hence the need to return a set of labels rather than a single one.

Algorithm 4.3 G-kNN full algorithm

Input: $\mathcal{S}$: the training dataset, $\boldsymbol{x}$: the input sample
k: number of neighbors

Output: Set of all the possible classification of $\boldsymbol{x}$

1: $\mathbb{G}_{\boldsymbol{x}} \leftarrow$ CreatePPGraph($\mathcal{S}, \boldsymbol{x}$)
2: `valid_paths` $\leftarrow$ ExtractValidPath($\mathbb{G}_{\boldsymbol{x}}$,k)
3: `classification` $\leftarrow \emptyset$
4: **for all** `path` **in** `valid_paths` **do**
5: `labels` $\leftarrow$ most frequent labels in `path`
6: `classification` $\leftarrow$ `classification` $\cup$ `labels`
7: **return** `classification`

Theorem 4.1. *Given the training dataset $\mathcal{S}$ and the input sample $\boldsymbol{x}$, Algorithm 4.3 returns all the possible classifications of the input $\boldsymbol{x}$.*

Proof. G-kNN classifies an input $\boldsymbol{x}$ by selecting the most frequent labels among the valid paths returned by ExtractValidPath method. Because this method provably returns all k nearest neighbors of $\boldsymbol{x}$ (as shown in Proposition 4.3 and Proposition 4.4), G-kNN is guaranteed to consider all possible classifications for $\boldsymbol{x}$. □

Example 4.2. Consider the scenario of Example 4.1. Running the Algorithm 4.3 with $k = 3$ the input is classified with the label 1 since in all valid paths having 3 samples (i.e., $[\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3]$ and $[\boldsymbol{s}_2, \boldsymbol{s}_1, \boldsymbol{s}_3]$) the most frequent label is exactly 1. In contrast, if $k = 2$ then Algorithm 4.3 will return the set of labels $\{1, -1\}$ because the valid paths having 2 samples (i.e., $[\boldsymbol{s}_1, \boldsymbol{s}_2]$ and $[\boldsymbol{s}_2, \boldsymbol{s}_1]$) contains both labels and so there is a tie.

4.2 Exact Robustness Certification

The previous section described our modifications the standard kNN in order to select the k nearest neighbors without relying on the Euclidean distance between the training samples and the input samples. As we will see in this section these modifications will be crucial for certifying the robustness (or vulnerability) of G-kNN algorithm, and consequently of the standard k-NN, to an adversarial region of the input sample. An adversarial region is defined as follows

Definition 4.5. Given $\boldsymbol{x} \in \mathbb{R}^n$ and $\epsilon \in \mathbb{N}, \epsilon \geq 0$, the adversarial region of $\boldsymbol{x}$, denoted with $P^{\epsilon}(\boldsymbol{x})$, is defined as the ℓ_∞-ball centered in $\boldsymbol{x}$ and radius ϵ:

$$P^{\epsilon}(\boldsymbol{x}) = \{\boldsymbol{s}_i \in \mathbb{R}^n \mid\mid\mid \boldsymbol{x} - \boldsymbol{s}_i \mid\mid_\infty \leq \epsilon\}$$

where $\| \cdot \|_\infty$ is ℓ_∞ (or maximum) norm.

Following Definition 2.2, given a training dataset $\mathcal{S}$ and an adversarial region $P^{\epsilon}(\boldsymbol{x})$ of a test sample $\boldsymbol{x}$, to provably certify that the G-kNN is robust to the region $P^{\epsilon}(\boldsymbol{x})$ we need to verify whether the prediction of G-kNN remains the same for all the samples within $P^{\epsilon}(\boldsymbol{x})$ and coincides with the ground truth of $\boldsymbol{x}$. To verify this, the robustness certifier, given $\mathcal{S}$ and $P^{\epsilon}(\boldsymbol{x})$, computes a set of labels $O \subseteq \mathcal{L}$ such that

$$O = \bigcup_{\boldsymbol{s}_i \in P^{\epsilon}(\boldsymbol{x})} \text{G-}k\text{NN}(\mathcal{S}, \boldsymbol{s}_i, k)$$

that is O contains all and only the labels assigned by G-kNN to samples within $P^\epsilon(\boldsymbol{x})$. Then if O consists of a single label, then the G-kNN is guaranteed to be stable. Moreover, if this label matches the ground truth of $\boldsymbol{x}$, the G-kNN is considered robust to the adversarial region $P^\epsilon(\boldsymbol{x})$. Computing the set O by applying G-kNN on each sample within the adversarial region is obviously unfeasible since it contains an infinite number of samples. However, notice that the output of G-kNN depends manly on the valid paths within the PP-G of the input. So following this observation the set O can be computed as follows

1. Create a graph $\mathbb{G}^A_{\boldsymbol{x}} = (V, E)$ where $V = \mathcal{S}_{\boldsymbol{x}}$ and the set E is such that $\mathbb{G}^A_{\boldsymbol{x}}$ contains all the valid paths of any PP-G $\mathbb{G}_{\boldsymbol{s}_i}$ of a sample $\boldsymbol{s}_i \in P^\epsilon(\boldsymbol{x})$. We call the graph $\mathbb{G}^A_{\boldsymbol{x}}$ *adversarial proximity precedence* graph (APP-G);
2. Select every path in $\mathbb{G}^A_{\boldsymbol{x}}$ which is a valid path in some PP-G of a sample $\boldsymbol{s}_i \in P^\epsilon(\boldsymbol{x})$;
3. Classify the perturbation with most frequent labels in each path selected in the previous step.

With this procedure the set O can be computed since the number of paths within $\mathbb{G}^A_{\boldsymbol{x}}$ is finite. The next subsections will describe each step of this procedure.

4.2.1 Adversarial Proximity Precedence Graph

To understand how to construct the adversarial proximity precedence graph $\mathbb{G}^A_{\boldsymbol{x}}$ suppose for example there are two samples $\boldsymbol{s}_1, \boldsymbol{s}_2 \in \mathcal{S}$ and points $\boldsymbol{x}_1, \boldsymbol{x}_2 \in P^\epsilon(\boldsymbol{x})$ such that

$$\begin{cases} \boldsymbol{s}_1 \preccurlyeq_{\boldsymbol{x}_1} \boldsymbol{s}_2 \wedge \boldsymbol{s}_2 \not\preccurlyeq_{\boldsymbol{x}_1} \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \preccurlyeq_{\boldsymbol{x}_2} \boldsymbol{s}_1 \wedge \boldsymbol{s}_1 \not\preccurlyeq_{\boldsymbol{x}_2} \boldsymbol{s}_2 \end{cases}$$

that is $\boldsymbol{s}_1$ is strictly closer to $\boldsymbol{x}_1$ than $\boldsymbol{s}_2$ and $\boldsymbol{s}_2$ is strictly closer to $\boldsymbol{x}_2$ than $\boldsymbol{s}_1$. In this case paths in which $\boldsymbol{s}_1$ is a predecessor of $\boldsymbol{s}_2$ and those in which $\boldsymbol{s}_2$ is a predecessor of $\boldsymbol{s}_1$ are valid paths in the PP-G of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ respectively. Therefore, those paths must also be both valid paths in $\mathbb{G}^A_{\boldsymbol{x}}$. This leads to the definition of the following relation between samples

Definition 4.6 (closer-to-region relation $\preccurlyeq^A_{(\boldsymbol{x},\epsilon)}$). Given $\boldsymbol{x} \in \mathbb{R}^n, \epsilon \in \mathbb{N}$ and $\boldsymbol{s}_1, \boldsymbol{s}_2 \in \mathcal{S}$

$$\boldsymbol{s}_1 \preccurlyeq^A_{(\boldsymbol{x},\epsilon)} \boldsymbol{s}_2 \iff \exists \boldsymbol{s}_i \in P^\epsilon(\boldsymbol{x}).\ \boldsymbol{s}_1 \preccurlyeq_{\boldsymbol{s}_i} \boldsymbol{s}_2$$

In the following, for ease of the notation, the subscript ϵ will be dropped since it is constant.

According to Definition 4.6, $\boldsymbol{s}_1 \preccurlyeq_{\boldsymbol{x}}^{A} \boldsymbol{s}_2$ and $\boldsymbol{s}_2 \preccurlyeq_{\boldsymbol{x}}^{A} \boldsymbol{s}_1$ therefore in $\mathbb{G}_{\boldsymbol{x}}^{A}$ there should be an edge between $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ in both direction. If instead $\boldsymbol{s}_1 \preccurlyeq_{\boldsymbol{x}}^{A} \boldsymbol{s}_2$ but $\boldsymbol{s}_2 \not\preccurlyeq_{\boldsymbol{x}}^{A} \boldsymbol{s}_1$ then it means the $\boldsymbol{s}_1$ is closer to every sample within $P^{\epsilon}(\boldsymbol{x})$ than $\boldsymbol{s}_2$. As consequence the sample $\boldsymbol{s}_1$ is a predecessor of $\boldsymbol{s}_2$ in every PP-G of a sample in $P^{\epsilon}(\boldsymbol{x})$. Therefore, in $\mathbb{G}_{\boldsymbol{x}}^{A}$ there should be only and edge from $\boldsymbol{s}_1$ to $\boldsymbol{s}_2$.

To determine if the relation $\preccurlyeq_{\boldsymbol{x}}^{A}$ exists between two samples $\boldsymbol{s}_1, \boldsymbol{s}_2 \in \mathcal{S}$, one would need to verify for each point $\boldsymbol{x}_i \in P^{\epsilon}(\boldsymbol{x})$ whether $\boldsymbol{s}_1 \preccurlyeq_{\boldsymbol{x}_i} \boldsymbol{s}_2$ holds or not. However, this approach is impractical due to the infinite number of points in $P^{\epsilon}(\boldsymbol{x})$. A more feasible method is to check whether the perpendicular bisector of $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ intersect with $P^{\epsilon}(\boldsymbol{x})$ as show in Figure 4.3 where the gray region is the adversarial region of the input sample.

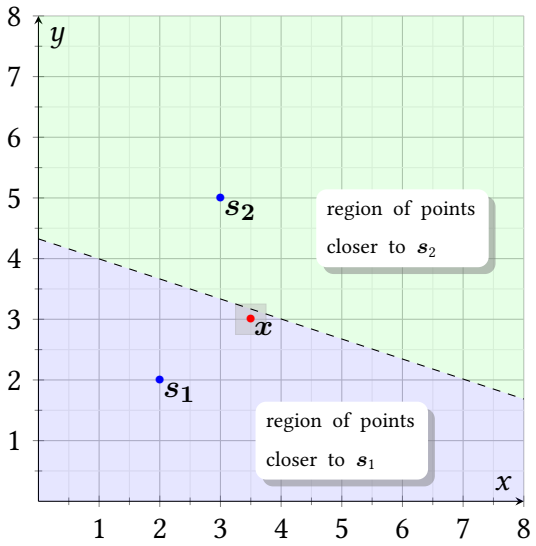


Figure 4.3: Example showing how to determine if $a\boldsymbol{s}_1 \preccurlyeq_{\boldsymbol{x}} \boldsymbol{s}_2$. Since the adversarial region intersect with the perpendicular bisector of $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ the relation $\boldsymbol{s}_1 \preccurlyeq_{\boldsymbol{x}}^{A} \boldsymbol{s}_2$ holds.

To see how this can be done let first define some quantities that will be used later:

Definition 4.7 (pos_neg function). Given $\boldsymbol{x}_i \in \mathbb{R}^n$ let $pos\_neg : \mathbb{R}^n \rightarrow \{-1, 1\}$ be a function defined as

$$pos\_neg(\boldsymbol{x}_i) = \begin{cases} 1, \text{ if } \boldsymbol{x}_i \geq 0 \\ -1, \text{ otherwise} \end{cases}$$

Proposition 4.5. Given $\boldsymbol{n} \in \mathbb{R}^n, b \in \mathbb{N}$ let $\pi_{\boldsymbol{n},b}$ be a hyperplane with normal vector $\boldsymbol{n}$ and $P^\epsilon(\boldsymbol{x})$ an adversarial region of a point $\boldsymbol{x} \in \mathbb{R}^n$ as defined in Definition 4.5 then

$$\pi_{\boldsymbol{n},b} \text{ intersect } P^\epsilon(\boldsymbol{x}) \iff \boldsymbol{x} \in \pi_{\boldsymbol{n},b} \vee sign(\pi_{\boldsymbol{n},b}(\boldsymbol{x})) \neq sign(\pi_{\boldsymbol{n},b}(\boldsymbol{x}'))$$

where $\boldsymbol{x}' = \boldsymbol{x} - \epsilon \cdot sign(\pi_{\boldsymbol{n},b}(\boldsymbol{x})) \cdot pos\_neg^\star(\boldsymbol{n})$ and $pos\_neg^\star$ is the component-wise pos_neg operation over vectors in $\mathbb{R}^n$. Essentially the hyperplane $\pi_{\boldsymbol{n},b}$ intersects the adversarial region of $\boldsymbol{x}$ if and only if $\boldsymbol{x}$ and the point $\boldsymbol{x}'$, which is the vertex of the hypercube $P^\epsilon(\boldsymbol{x})$ in the direction of $\pi_{\boldsymbol{n},b}$ from $\boldsymbol{x}$, are on the opposite side of $\pi_{\boldsymbol{n},b}$.

Proof. The proposition can be proved by demonstrating each direction of the implication separately:

- ($\Longrightarrow$): Suppose $\pi_{\boldsymbol{n},b}$ intersect $P^\epsilon(\boldsymbol{x})$ and $\boldsymbol{x}_i \in \pi_{\boldsymbol{n},b}$. Since $\pi_{\boldsymbol{n},b}$ intersect the adversarial region there surely exist a point $\boldsymbol{x}'' \in P^\epsilon(\boldsymbol{x})$ such that $\boldsymbol{x}'' \in \pi_{\boldsymbol{n},b}$. Because $\boldsymbol{x}'' \in P^\epsilon(\boldsymbol{x})$ it can also be defined as

  $$\boldsymbol{x}'' = \boldsymbol{x} - sign(\pi_{\boldsymbol{n},b}(\boldsymbol{x})) \cdot \boldsymbol{\varepsilon}'' \odot pos\_neg^\star(\boldsymbol{n})$$

  where $\odot$ denotes the *Hadamard* product and $\boldsymbol{\varepsilon}'' \in \mathbb{R}^n$ is such that $||\boldsymbol{\varepsilon}''||_\infty \leq \epsilon$. The point $\boldsymbol{x}'$ on the other side of the hyperplane with respect to where $\boldsymbol{x}$ is located can be defined in terms of $\boldsymbol{x}''$ as follows

  $$\begin{aligned}\boldsymbol{x}' &= \boldsymbol{x}'' - sign(\pi_{\boldsymbol{n},b}(\boldsymbol{x})) \cdot \boldsymbol{\varepsilon}' \odot pos\_neg^\star(\boldsymbol{n}) \\ &= \boldsymbol{x} - sign(\pi_{\boldsymbol{n},b}(\boldsymbol{x})) \cdot (\boldsymbol{\varepsilon}'' + \boldsymbol{\varepsilon}') \odot pos\_neg^\star(\boldsymbol{n})\end{aligned}$$

  for some $\boldsymbol{\varepsilon}' \in \mathbb{R}^n$ is such that $\forall i \in \{1, \ldots, n\} \;\; |\varepsilon_i'' + \varepsilon_i'| = \epsilon$. With such value $sign(\pi_{\boldsymbol{n},b}(\boldsymbol{x}'))$ is as follows

  $$\begin{aligned}sign(\pi_{\boldsymbol{n},b}(\boldsymbol{x}')) &= sign\left(\boldsymbol{n} \cdot \left[\boldsymbol{x} - \overbrace{sign(\pi_{\boldsymbol{n},b}(\boldsymbol{x}))}^{s} \cdot (\boldsymbol{\varepsilon}'' + \boldsymbol{\varepsilon}') \odot \overbrace{pos\_neg^\star(\boldsymbol{n})}^{\boldsymbol{dir}}\right] + b\right) \\ &= sign\left(\boldsymbol{n} \cdot [\boldsymbol{x} - s \cdot (\boldsymbol{\varepsilon}'' \odot \boldsymbol{dir} + \boldsymbol{\varepsilon}' \odot \boldsymbol{dir})] + b\right) \\ &= sign\left(\boldsymbol{n} \cdot [(\boldsymbol{x} - s \cdot \boldsymbol{\varepsilon}'' \odot \boldsymbol{dir}) + s \cdot \boldsymbol{\varepsilon}' \odot \boldsymbol{dir}] + b\right) \\ &= sign\left(\boldsymbol{n} \cdot [\boldsymbol{x} - s \cdot \boldsymbol{\varepsilon}'' \odot \boldsymbol{dir}] + b - s \cdot \boldsymbol{n} \cdot \boldsymbol{\varepsilon}' \odot \boldsymbol{dir}\right)\end{aligned}$$

$$
\begin{aligned}
&= sign\left(\overbrace{\boldsymbol{n}\cdot\boldsymbol{x}''+b}^{=0} - s\cdot\boldsymbol{n}\cdot\boldsymbol{\varepsilon}'\odot\boldsymbol{dir}\right)\\
&= sign\left(-s\cdot\boldsymbol{n}\cdot\boldsymbol{\varepsilon}'\odot\boldsymbol{dir}\right)\\
&= sign\left(-\,sign(\pi_{\boldsymbol{n},b}(\boldsymbol{x}))\cdot\overbrace{\boldsymbol{n}\cdot\boldsymbol{\varepsilon}'\odot pos\_neg^{\star}(\boldsymbol{n})}^{\text{positive}}\right) \qquad (4.1)\\
&= -\,sign(\pi_{\boldsymbol{n},b}(\boldsymbol{x}))\neq sign(\pi_{\boldsymbol{n},b}(\boldsymbol{x}))
\end{aligned}
$$

In Equation 4.1 $\forall i \in 1,.,n$ the sign of $\boldsymbol{\varepsilon}'_i \cdot pos\_neg(\boldsymbol{n}_i)$ is the same as $\boldsymbol{n}_i$ and so the sign of $\boldsymbol{n}_i \cdot \boldsymbol{\varepsilon}'_i \cdot pos\_neg(\boldsymbol{n}_i)$ is always positive hence $\boldsymbol{n}\cdot\boldsymbol{\varepsilon}'\cdot pos\_neg\star(\boldsymbol{n})$ is positive value.

In the case $\boldsymbol{x}\in\pi_{\boldsymbol{n},b}$ then the implication holds vacuously.

- ($\Longleftarrow$): if $\boldsymbol{x}\in\pi_{\boldsymbol{n},b}\vee sign(\pi_{\boldsymbol{n},b}(\boldsymbol{x}))\neq sign(\pi_{\boldsymbol{n},b}(\boldsymbol{x}'))$ then there exist two points in the adversarial region on opposite sides of the hyperplane. This implies that $\pi_{\boldsymbol{n},b}$ definitely intersect $P^{\epsilon}(\boldsymbol{x})$.

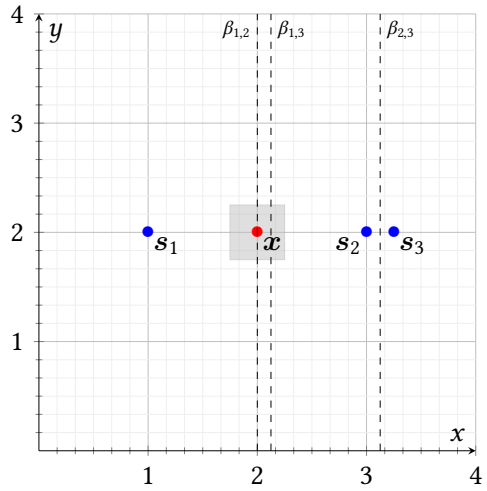
□

Example 4.3. Consider a dataset $\mathcal{S}\subset\mathbb{R}^2$ and the adversarial region $P^{\epsilon}(\boldsymbol{x})$ of an input sample $\boldsymbol{x}\in\mathbb{R}^2$ with magnitude $\epsilon = 0.25$ as shown in Figure 4.4a. In this scenario the perpendicular bisector of samples $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ (i.e. the dashed line label with $\beta_{1,2}$) intersects with adversarial region $P^{\epsilon}(\boldsymbol{x})$, and same holds for samples $\boldsymbol{s}_1$ and $\boldsymbol{s}_3$ as evidenced by perpendicular bisector labeled with $\beta_{1,3}$. Meanwhile, the sample $\boldsymbol{s}_2$ is closer to the every point within the adversarial region than $\boldsymbol{s}_3$ as show by the fact that their perpendicular bisector (i.e., the dashed line label with $\beta_{2,3}$) does not intersect with $P^{\epsilon}(\boldsymbol{x})$. As a consequence in the APP-G of $\boldsymbol{x}$, illustrated in Figure 4.4b, the samples $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ are adjacent of each other as well as $\boldsymbol{s}_1$ and $\boldsymbol{s}_3$. Meanwhile, between $\boldsymbol{s}_2$ and $\boldsymbol{s}_3$ there is only one edge going from $\boldsymbol{s}_2$ to $\boldsymbol{s}_3$.

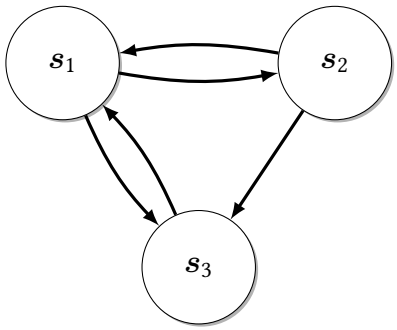
Proposition 4.6. Given a sample $\boldsymbol{s}\in\mathcal{S}$ and a APP-G $\mathbb{G}^A_{\boldsymbol{x}}$

$$
\texttt{pred}^A(\boldsymbol{s}) = \bigcap_{\boldsymbol{x}'\in P^{\epsilon}(\boldsymbol{x})} \texttt{pred}_{\boldsymbol{x}'}(\boldsymbol{s})
$$

where $\texttt{pred}^A(\boldsymbol{s})$ and $\texttt{pred}_{\boldsymbol{x}'}(\boldsymbol{s})$ are the predecessor of $\boldsymbol{s}$ within the APP-G of $\boldsymbol{x}$ and PP-G of a sample $\boldsymbol{x}'$ within $P^{\epsilon}(\boldsymbol{x})$ respectively.



(a) Graph showing the dataset $\mathcal{S}$ and input sample of Example 4.3



(b) APP-G of input sample of Example 4.3

Figure 4.4: Example showing the APP-G of the input sample of Example 4.3

Proof. We can prove the preposition by demonstrating both direction of inclusion of the equality:

- ($\subseteq$): Suppose $\boldsymbol{s}_i \in \texttt{pred}^A(\boldsymbol{s})$. By definition of adversarial proximity precedence graph and Definition 4.6, we have that $\boldsymbol{s}_i \preccurlyeq^A_{\boldsymbol{x}} \boldsymbol{s} \wedge \boldsymbol{s} \not\preccurlyeq^A_{\boldsymbol{x}} \boldsymbol{s}_i$ which imply that every point in $P^\epsilon(\boldsymbol{x})$ is strictly closer to $\boldsymbol{s}_i$ than $\boldsymbol{s}$. Consequently, by definition of proximity precedence graph and Definition 4.1, $\boldsymbol{s}_i \in \texttt{pred}(\boldsymbol{s})$ in the PP-G of any point within $P^\epsilon(\boldsymbol{x})$. This means that

$$\boldsymbol{s}_i \subseteq \bigcap_{\boldsymbol{x}' \in P^\epsilon(\boldsymbol{x})} \texttt{pred}_{\boldsymbol{x}'}(\boldsymbol{s})$$

- ($\supseteq$): Suppose $\boldsymbol{s}_i \in \texttt{pred}(\boldsymbol{s})$ in the PP-G of any point within $P^\epsilon(\boldsymbol{x})$. Then by definition of proximity precedence graph and Definition 4.1, it means the every point in $P^\epsilon(\boldsymbol{x})$ is strictly closer to $\boldsymbol{s}_i$ than $\boldsymbol{s}$. As consequence, By Definition 4.6, $\boldsymbol{s}_i \preccurlyeq^A_{\boldsymbol{x}} \boldsymbol{s} \wedge \boldsymbol{s} \not\preccurlyeq^A_{\boldsymbol{x}} \boldsymbol{s}_i$ which means that

$$\boldsymbol{s}_i \in \texttt{pred}^A(\boldsymbol{s})$$

□

Algorithm 4.4 shows how the APP-G $\mathbb{G}^A_{\boldsymbol{x}}$ of an input sample $\boldsymbol{x} \in \mathbb{R}^n$ can be created given the dataset $\mathcal{S}$ and the perturbation magnitude ϵ. The procedure is the same as

Algorithm 4.1 with the only difference being the order relation used which in this case is $\preccurlyeq^{A}_{(\boldsymbol{x},\epsilon)}$.

Algorithm 4.4 CreateAPPGraph method

Input: S: A training set, $\boldsymbol{x}$: The input sample, ϵ: The perturbation magnitude
Output: The adversarial proximity precedence graph $\mathbb{G}^{A}_{\boldsymbol{x}}$ of $\boldsymbol{x}$

1: $V \leftarrow S_X$
2: $E \leftarrow \varnothing$
3: **for all** $(\boldsymbol{s}_i, \boldsymbol{s}_j) \in \{(\boldsymbol{s}_i, \boldsymbol{s}_j) \mid \boldsymbol{s}_i, \boldsymbol{s}_j \in V, \boldsymbol{s}_i \neq \boldsymbol{s}_j\}$ **do**
4: **if** $\boldsymbol{s}_i \preccurlyeq^{A}_{(\boldsymbol{x},\epsilon)} \boldsymbol{s}_j$
5: $E \leftarrow E \cup \{(\boldsymbol{s}_i, \boldsymbol{s}_j)\}$
6: **return** (V, E)

4.2.2 Nearest Neighbors Selection

Like the G-kNN algorithm, once the graph is constructed, the certifier identifies the possible k nearest neighbors of each point inside the adversarial region. To achieve this it finds all the paths comprised of k samples within the APP-G $\mathbb{G}^{A}_{\boldsymbol{x}}$ of the input sample that are valid in some PP-G of a point $\boldsymbol{x}' \in P^{\epsilon}(\boldsymbol{x})$. Such paths will be called *adversarially valid*. Formally:

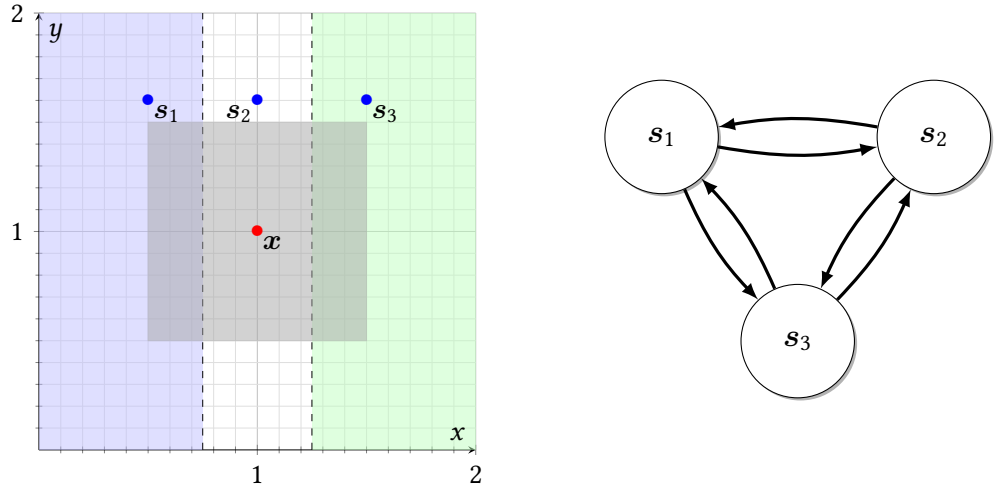
Definition 4.8 (Adversarially valid path). Given an adversarial proximity precedence graph $\mathbb{G}^{A}_{\boldsymbol{x}}$, let $\mathcal{P} = [\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_m]$ be a path in $\mathbb{G}^{A}_{\boldsymbol{x}}$. Then $\mathcal{P}$ is called *adversarially valid* if

1. $\forall \boldsymbol{s}_i \in \mathcal{P}.\ \forall \boldsymbol{s}_j \in \texttt{pred}(\boldsymbol{s}_i).\ \boldsymbol{s}_j$ is a predecessor of $\boldsymbol{s}_i$ in $\mathcal{P}$;
2. $\exists \boldsymbol{x}' \in P^{\epsilon}(\boldsymbol{x})$ such that $\mathcal{P}$ is valid in the PP-G of $\boldsymbol{x}'$.

Proposition 4.7. Not all paths of a APP-G are adversarially valid.

Proof. Consider the dataset $\mathcal{S} \subset \mathbb{R}^2$ and the adversarial region of the input sample $\boldsymbol{x} \in \mathbb{R}^2$ shown in Figure 4.5a. In this case all samples in $\mathcal{S}$ are equidistant from the adversarial region and consequently, the APP-G $\mathbb{G}^{A}_{\boldsymbol{x}}$ of the input forms a clique as illustrated in Figure 4.5b,

Consider, for instance, the path $\mathcal{P} = [s_1, s_3, s_2]$. $\mathcal{P}$ satisfy the first condition of Definition 4.8 but not the second. This is because for $\mathcal{P}$ to be adversarially valid there must exist a point $x' \in P^\epsilon(x)$ such that s_1 is the nearest sample to x', s_3 is the second closest and s_2 the third closest, however such x' does not exist. The region of space containing points closer to s_1 than any other samples (i.e., the region highlighted with blue) does not intersect with the region containing points closer to sample s_3 than s_2 (i.e., the region colored with green).



(a) Graph showing the dataset $\mathcal{S}$ and input sample of Proposition 4.7

(b) APP-G of input sample of Proposition 4.7

Figure 4.5: Example of non adversarially valid path in a APP-G

□

Determining if a path $\mathcal{P}$ is adversarial valid by exhaustively searching for a point in the adversarial region whose PP-G includes $\mathcal{P}$ as valid path is computationally intractable due the infinite cardinality of $P^\epsilon(x)$. A more efficient strategy is to define a region R of points such that the sequence of samples defined by $\mathcal{P}$ is ordered according to the distance to those points. The path's adversarial validity can then be assessed by checking for an intersection between R and the adversarial region. To see how this can be achieved consider the dataset of Proposition 4.7. The path $[s_2]$ is adversarially valid if there exist a region of points $R_1 \in P^\epsilon(x)$ intersecting with $P^\epsilon(x)$ such that the sample $[s_2]$ is the closest to those points. This means that points in R_1 must satisfy the following linear system of inequalities:

$$\begin{cases} x_1 - 1.25 \leq 0 \\ -x_1 + 0.75 \leq 0 \\ 0.50 \leq x_1 \leq 1.50 \\ 0.50 \leq x_2 \leq 1.50 \end{cases}$$

where $x_1 - 1.25 = 0$ and $x_1 + 0.75 = 0$ are the perpendicular bisector between the samples $\boldsymbol{s}_2$ and $\boldsymbol{s}_3$ and samples $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ respectively. The region R_1 does exist as it shown Figure 4.6a as the red highlighted area of the perturbation region. Consider now the path $[\boldsymbol{s}_2, \boldsymbol{s}_1]$. This path is valid if it exists a region of points $R_2 \in P^{\epsilon}(\boldsymbol{x})$, intersecting the adversarial region, such that the closest sample to them $\boldsymbol{s}_2$ and the second closest is $\boldsymbol{s}_1$. So points in R_2 must satisfy the following linear system of inequalities:

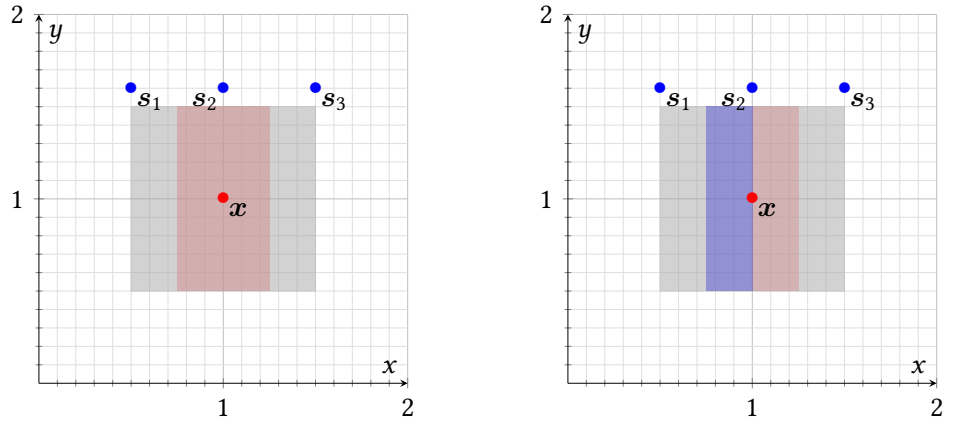
$$\begin{cases} \left.\begin{array}{l} x_1 - 1.25 \leq 0 \\ -x_1 + 0.75 \leq 0 \end{array}\right\} \dashrightarrow \text{region of points closer to } \boldsymbol{s}_2 \text{ than any other samples} \\ x_1 - 1 \leq 0 \dashrightarrow \text{region of points closer to } \boldsymbol{s}_1 \text{ than } \boldsymbol{s}_3 \\ \left.\begin{array}{l} 0.50 \leq x_1 \leq 1.50 \\ 0.50 \leq x_2 \leq 1.50 \end{array}\right\} \dashrightarrow \text{ adversarial region} \end{cases} \tag{4.2}$$

where the first two inequalities define the R_1 region while the third inequality is the region of space where points are closer to $\boldsymbol{s}_1$ than $\boldsymbol{s}_3$. This region is shown with the blue area in Figure 4.6b. Finally, the same logic applies to path $[\boldsymbol{s}_2, \boldsymbol{s}_1, \boldsymbol{s}_3]$ which defined the same the linear system as Equation 4.2. On the other hand the linear system of inequalities associated with path $[\boldsymbol{s}_1, \boldsymbol{s}_3]$ is the following

$$\begin{cases} x_1 - 0.75 \leq 0 \dashrightarrow \text{region of points closer to } \boldsymbol{s}_1 \text{ than any other samples} \\ -x_1 + 1.25 \leq 0 \dashrightarrow \text{region of points closer to } \boldsymbol{s}_3 \text{ than } \boldsymbol{s}_2 \\ \left.\begin{array}{l} 0.50 \leq x_1 \leq 1.50 \\ 0.50 \leq x_2 \leq 1.50 \end{array}\right\} \dashrightarrow \text{adversarial region} \end{cases} \tag{4.3}$$

The regions defined by the first two equations are illustrated in Figure 4.5a. As can be seen from the plot the two regions do not intersect hence the linear system has no

solutions. So the path $[s_1, s_3]$ and consequently$[s_1, s_3, s_2]$ are not adversarially valid paths.



(a) Graph showing region R_1 **(b)** Graph showing regions R_1 and R_2

Figure 4.6: Example showing how to assess the adversarial validity of a path in an APP-G

In summary, given a path $\mathcal{P}$, the idea is to find a polytope within the perturbation region such that for any point x' within this polytope, the sequence of samples defined by $\mathcal{P}$ is ordered by their distance to x'. This polytope is defined by a system of linear inequalities. If this system has a solution (i.e., the polytope exists), it implies the existence of at least one PP-G $\mathbb{G}_{x'}$ for some $x' \in P^{\epsilon}(x)$ such that $\mathcal{P}$ is valid in $\mathbb{G}_{x'}$. Conversely, if the linear system has no solutions, it means that no such PP-G exists and so $\mathcal{P}$ is not adversarially valid. Given its role in determining the validity of $\mathcal{P}$, we refer to this polytope as the *validity polytope* of the path.

Given the path $\mathcal{P} = [s_1, s_2, \ldots, s_m]$ and the adversarial region $P^{\epsilon}(x)$ of the input sample x the system of linear inequalities defining the validity polytope of $\mathcal{P}$ is the following

$$
\begin{cases}
\textsc{Closer}(s_1, \texttt{same\_dist}(s_1)) \\
\textsc{Closer}(s_2, \texttt{same\_dist}(s_2) \setminus \{s_1\}) \\
\qquad\qquad \vdots \\
\textsc{Closer}(s_i, \texttt{same\_dist}(s_i) \setminus \{s_1, \dots s_{i-1}\}) \\
\qquad\qquad \vdots \\
\textsc{Closer}(s_m, \texttt{same\_dist}(s_m) \setminus \{s_1, \dots s_{m-1}\}) \\
x_1 - \epsilon \leq x_1 \leq x_1 + \epsilon \\
\qquad\quad \vdots \\
x_n - \epsilon \leq x_n \leq x_n + \epsilon
\end{cases}
\tag{4.4}
$$

where Closer is a method, shown in Algorithm 4.5, that, given a sample $s \in \mathcal{S}$ and a set of samples $S \subseteq \mathcal{S}$, returns the set of inequalities defining the region of points closer to s than any sample in S. To do this it computes, for each $s_i \in S$,the perpendicular bisector $\beta : a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$ between s and s_i such that $\beta(s) \leq 0$ (line 3) and add the inequality $a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b$ to the returned set of inequality (line 4).

Algorithm 4.5 Closer method

Input: s: A sample, S: A set of sample
Output: Set of linear of inequalities defining a region with point closer to s than any sample in S

1: `ineq` $\leftarrow \emptyset$
2: **for all** $s_i \in S$ **do**
3: $\quad a_1x_1 + a_2x_2 + \cdots + a_nx_n = b \leftarrow \textsc{Bisector}(s, s_i)$
4: $\quad$ `ineq` $\leftarrow$ `ineq` $\cup \{a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b\}$
5: **return** `ineq`

Definition 4.9 (Adversarially safe sample). Let $\mathcal{P} = [s_1, s_2, \dots, s_m]$ be an adversarially valid path within an APP-G $\mathbb{G}^A_{\boldsymbol{x}}$. Then a sample $s_{m+1} \in \mathcal{S}, s_{m+1} \notin \mathcal{P}$ is called an *adversarially safe* sample for $\mathcal{P}$ if the path $\mathcal{P}' = \mathcal{P}' + [s_{m+1}] = [s_1, s_2, \dots, s_m, s_{m+1}]$ is adversarially valid.

To determine if a sample $s \in \mathcal{S}$ is adversarially safe for path $\mathcal{P}$, we first check if all

its predecessors (i.e., $\texttt{pred}(s)$) are present in $\mathcal{P}$. Then, we verify the existence of the validity polytope for the extended path $\mathcal{P} + [s]$ by checking the consistency of the linear system obtained by adding the inequalities in $\textsc{Closer}(s, \texttt{same\_dist}(s) \setminus \mathcal{P})$ to the linear system defining the validity polytope of $\mathcal{P}$. If the resulting linear system is consistent (i.e., has a solution), then s is adversarially safe for $\mathcal{P}$; otherwise, it is not.

Algorithm 4.6 shows how all the adversarially valid paths with n samples are extracted from a given APP-G $\mathbb{G}_x^A$ of the input sample. Essentially the procedure is the same as Algorithm 4.2 with only the difference being that a sample is added to a path if it is adversarially safe for that path.

Algorithm 4.6 SELECTADVVALIDPATHS method

Input: $\mathbb{G}_x^A$: An APP-G,
n: number of samples in the path
Output: Set of all adversarially valid paths comprised n samples

```
1:  queue ← {[s_i] | s_i ∈ G_x ∧ pred(s_i) = ∅}
2:  k ← 1
3:  while queue not empty do
4:      current_paths ← queue.PopAll()
5:      if k = n
6:          return current_paths
7:      for all path in current_paths do
8:          last_sample ← path.last()
9:          for all sample in adj(last_sample) do
10:             if sample is adversarially safe for path
11:                 queue.append(path + [sample])
12:     k ← k + 1
```

Proposition 4.8. Given an APP-G $\mathbb{G}_x^A$ and the length $n \in \mathbb{N}$, Algorithm 4.6 returns all the adversarially valid paths comprised of n samples within $\mathbb{G}_x^A$.

Proof. The proposition can be proved by showing that in the k-th iteration of the while loop, at line 4 the queue contains all the valid paths with k samples. We proceed by induction on k:

- **Base Case**($k = 1$): In the first iteration of the loop the queue contains all the path made of a single sample $s \in \mathcal{S}$ such that $\texttt{pred}(s_i) = \emptyset$. By definition of APP-G

and Definition 4.6, there exists a region $R \subseteq P^{\epsilon}(\boldsymbol{x})$ within whose points are closer to $\boldsymbol{s}$ than any other sample in $\mathcal{S}$. Consequently, by Proposition 4.1 the path $[\boldsymbol{s}]$ is a valid in the PP-G of points in R. So this means that all the paths in the queue are adversarial valid. Additionally, the queue is surely non-empty due to the voronoi tesselation [38] of $\mathbb{R}^n$ induced by the samples in $\mathcal{S}$.

- **Inductive Case**($k = h + 1, h \geq 1$): At iteration h+1, the paths inside the queue at line 4 are obtained by extending all the paths in the queue in the h-th iteration with every adversarially safe sample among the adjacent of the path's last sample. By induction hypothesis the queue in the h-th iteration contains all the adversarially valid paths with h samples and since they are extended with every adversarially safe samples it means that in the h+1 iteration the queue contains only and all the adversarially valid paths comprised of $h + 1$.

□

4.2.3 Full Algorithm

Algorithm 4.7 shows the full algorithm of the exact certifier. Given the sample dataset $\mathcal{S}$, the input sample $\boldsymbol{x}$, and the perturbation magnitude ϵ, the certifier starts by constructing the APP-G $\mathbb{G}^A_{\boldsymbol{x}}$ of the input sample using the CreateAPPGraph method (line 1). Next, it extracts all adversarially valid paths consisting of k samples using the SelectAdvValidPaths method (line 2), and then collects the most frequent labels within each path (lines 3-6). Finally, these collected labels are returned as the possible classification of points in the adversarial region (line 8).

Algorithm 4.7 Certifier algorithm

Input: $\mathcal{S}$: A dataset, $\boldsymbol{x}$: The input sample
ϵ: The perturbation magnitude, k: The number neighbors to look for
Output: Set of possible labels that points in $P^{\epsilon}(\boldsymbol{x})$ can be classified with

1: $\mathbb{G}^A_{\boldsymbol{x}} \leftarrow$ CreateAPPGraph($\mathcal{S}, \boldsymbol{x}, \epsilon$)
2: `paths` $\leftarrow$ SelectAdvValidPaths($\mathbb{G}^A_{\boldsymbol{x}}$, k)
3: `classification` $\leftarrow \emptyset$
4: **for all** `path` **in** `paths` **do**
5: `labels` $\leftarrow$ most frequent labels in `path`
6: `classification` $\leftarrow$ `classification` $\cup$ `labels`
7: **return** `classification`

Theorem 4.2 (Certifier Exactness). *Given a dataset* $\mathcal{S} = \{(\boldsymbol{s}_i, l_{\boldsymbol{s}_i}) \mid \boldsymbol{s}_i \in \mathbb{R}^n, l_{\boldsymbol{s}_i} \in \mathcal{L}\}_{i=1}^{i=n}$ *and the input sample's adversarial region* $P^\epsilon(\boldsymbol{x})$ *of magnitude* ϵ*, the certifier returns exactly the labels assigned by G-kNN to the points within* $P^\epsilon(\boldsymbol{x})$ *that is*

$$\forall k, \epsilon \in \mathbb{N}\ \forall \boldsymbol{x} \in \mathbb{R}^n \quad \textsc{Certifier}(\mathcal{S}, \boldsymbol{x}, \epsilon, k) = \bigcup_{\boldsymbol{x}' \in P^\epsilon(\boldsymbol{x})} \textit{G-kNN}(\mathcal{S}, \boldsymbol{x}, k)$$

Proof. The set of labels returned by Certifier and G-kNN consists of the most frequent labels of the samples forming the paths of length $k-1$ extracted from the APP-G $\mathbb{G}_{\boldsymbol{x}}^A$ and PP-G $\mathbb{G}_{\boldsymbol{x}'}$ of the input sample respectively. Consequently, the theorem can be demonstrated by proving the following equality

$$\forall k, \epsilon \in \mathbb{N}\ \forall \boldsymbol{x} \in \mathbb{R}^n \quad \textsc{PathExtracted}^A(\mathbb{G}_{\boldsymbol{x}}^A) = \bigcup_{\boldsymbol{x}' \in P^\epsilon(\boldsymbol{x})} \textsc{PathExtracted}(\mathbb{G}_{\boldsymbol{x}'}) \tag{4.5}$$

where $\textsc{PathExtracted}^A(G)$ denotes the set of paths extracted from APP-G by Certifier and $\textsc{PathExtracted}(G)$ represents the set of paths extracted from the PP-G by G-kNN. We proceed by demonstrating both direction of inclusion of Equation 4.5:

- ($\subseteq$): Suppose the path $\mathcal{P}$ within the APP-G $\mathbb{G}_{\boldsymbol{x}}^A$ of the input $\boldsymbol{x}$ is an adversarially valid path. Then, by definition Definition 4.8, there exists a point $\boldsymbol{x}' \in P^\epsilon(\boldsymbol{x})$ such that $\mathcal{P}$ is valid path in the PP-G of $\boldsymbol{x}'$. Consequently, by Proposition 4.4

  $$\mathcal{P} \in \bigcup_{\boldsymbol{x}' \in P^\epsilon(\boldsymbol{x})} \textsc{PathExtracted}(\mathbb{G}_{\boldsymbol{x}'})$$

- ($\supseteq$): Without loss of generality consider a valid path $\mathcal{P} = [\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_k]$ in the PP-G $\mathbb{G}_{\boldsymbol{x}'}$ of some point $\boldsymbol{x}' \in P^\epsilon(\boldsymbol{x})$. By definition of PP-G

  $$\boldsymbol{s}_1 \preccurlyeq_{\boldsymbol{x}'} \boldsymbol{s}_2 \preccurlyeq_{\boldsymbol{x}'} \cdots \preccurlyeq_{\boldsymbol{x}'} \boldsymbol{s}_k$$

  and so, by definition of $\preccurlyeq_{\boldsymbol{x}}^A$

  $$\boldsymbol{s}_1 \preccurlyeq_{\boldsymbol{x}}^A \boldsymbol{s}_2 \preccurlyeq_{\boldsymbol{x}}^A \cdots \preccurlyeq_{\boldsymbol{x}}^A \boldsymbol{s}_k$$

  This indicates that $\mathcal{P}$ is also a path in the APP-G $\mathbb{G}_{\boldsymbol{x}}^A$ of $\boldsymbol{x}$. Moreover, since $\mathcal{P}$ is a valid path in $\mathbb{G}_{\boldsymbol{x}'}$, by Definition 4.3 and Proposition 4.6, every sample in $\mathcal{P}$ is preceded by its predecessor in $\mathbb{G}_{\boldsymbol{x}}^A$. This implies that $\mathcal{P}$ satisfies the requirements of Definition 4.8 and therefore, by Proposition 4.8, $\mathcal{P} \in \textsc{PathExtracted}^A(G)$. □

5

Optimizations

One issue with certifier is that it is not really efficient. One reason for this is the construction of the adversarial proximity precedence graph which is done by iterating over all the pairs of samples in $\mathcal{S}$. Building the graph thus costs $O(n^2)$ where n is the number of samples in $\mathcal{S}$, but it hides an important constant which is the dimension of the feature space. Another reason is the algorithm itself, which is quite inefficient because it explicitly enumerates all the possible paths of length k within the APP-G before computing the set of labels. For instance consider a dataset with n samples equidistant from the adversarial region $P^{\epsilon}(\boldsymbol{x})$ then in this case the graph of could contain at least

$$\frac{n!}{(n-k)!}$$

adversarially valid paths with k samples. This means that just for small values such as $n = 10$ and $k = 7$ the certifier could potentially iterate over more than 10^6 paths before starting to compute the set of labels. This chapter explores some optimizations to mitigate these inefficiencies.

5.1 APP-G Construction Optimizations

One way to optimize the construction of the APP-G is to use only the samples in the neighborhood of the input sample, as not all are needed for the classification. The following result provides a sufficient condition to ensure that the selected samples include the k nearest neighbors of every point in the input's adversarial region.

Proposition 5.1. Given a dataset $\mathcal{S}$ and the adversarial region $P^{\epsilon}(\boldsymbol{x})$ of input sample $\boldsymbol{x}$, let d be the distance between $\boldsymbol{x}$ and its k-th closest sample. Then, the hypersphere centered at $\boldsymbol{x}$ with radius $2\epsilon\sqrt{N} + d$, where N is the dimension of the feature space, includes the k nearest neighbors of every point within $P^{\epsilon}(\boldsymbol{x})$.

Proof. Given the set $A = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_k\}$ of the k nearest neighbor to the input sample, where $\boldsymbol{s}_k$ is the k-th nearest, let $\boldsymbol{x}' \in P^{\epsilon}(\boldsymbol{x})$ be an arbitrary point of the adversarial region. Consider the hypersphere $H_{\boldsymbol{x}'}$ centered in $\boldsymbol{x}'$ and radius $\left\|\boldsymbol{x}' - \boldsymbol{s}_{\boldsymbol{x}'}\right\|$ where $\boldsymbol{s}_{\boldsymbol{x}'} = \arg\max_{\boldsymbol{s}_i \in A} \left\|\boldsymbol{x}' - \boldsymbol{s}_i\right\|$ (i.e, $\boldsymbol{s}_{\boldsymbol{x}'}$ is the farthest of the k nearest neighbors of $\boldsymbol{x}$ from $\boldsymbol{x}'$). By definition, $H_{\boldsymbol{x}'}$ contains all the sample in A. Since the radius of $H_{\boldsymbol{x}'}$ is the distance from $\boldsymbol{x}'$ to the farthest point in A, any sample closer to $\boldsymbol{x}'$ than any sample in A will also be contained within $H_{\boldsymbol{x}'}$. Therefore, $H_{\boldsymbol{x}'}$ contains at least the k nearest neighbors to $\boldsymbol{x}'$. Let H be hypersphere centered ar $\boldsymbol{x}$ that encloses all hyperspheres $H_{\boldsymbol{x}'}$ for every $\boldsymbol{x}' \in P^{\epsilon}(\boldsymbol{x})$. Consequently, H surely contain all the k nearest samples to each point within the adversarial region. To determine the radius of H observe that given $\boldsymbol{y} \in \mathbb{R}^n$ and $\boldsymbol{x}' \in P^{\epsilon}(\boldsymbol{x})$

$$\left\|\boldsymbol{x}' - \boldsymbol{y}\right\| \leq \left\|\boldsymbol{x} - \boldsymbol{x}'\right\| + \|\boldsymbol{x} - \boldsymbol{y}\|$$

due to the triangular inequality property of norms. Therefore, for every $\boldsymbol{x}' \in P^{\epsilon}(\boldsymbol{x})$

$$\left\|\boldsymbol{x}' - \boldsymbol{s}_{\boldsymbol{x}'}\right\| \leq \left\|\boldsymbol{x} - \boldsymbol{x}'\right\| + \left\|\boldsymbol{x} - \boldsymbol{s}_{\boldsymbol{x}'}\right\| \leq \left\|\boldsymbol{x} - \boldsymbol{x}'\right\| + \|\boldsymbol{x} - \boldsymbol{s}_k\| \leq \epsilon\sqrt{N} + d$$

Since the maximum radius of $H_{\boldsymbol{x}'}$ for any $\boldsymbol{x}' \in P^{\epsilon}(\boldsymbol{x})$ is $\epsilon\sqrt{N}\epsilon + d$ and distance between the centers of H and any $H_{\boldsymbol{x}'}$ is at most $\epsilon\sqrt{N}$, the radius of hypersphere H must be $\epsilon\sqrt{N} + d + \epsilon\sqrt{N} = 2\epsilon\sqrt{N} + d$. □

An efficient approach to selecting samples in the neighborhood of the input is the following:

1. Partition the dataset $\mathcal{S}$ such that each partition contain at most m of samples where $k \leq m \ll |\mathcal{S}|$;

2. Find the partition P containing the input $\boldsymbol{x}$;

3. Compute the distance d between the $\boldsymbol{x}$ and its k-th closest sample in P;

4. Find the partitions Ps intersecting the hypersphere centered at $\boldsymbol{x}$ with radius

$$2\epsilon\sqrt{N} + d$$

where ϵ and N are the perturbation magnitude and the dimension of the feature space respectively.

This procedure, as demonstrated Proposition 5.1, allows us to select a subset of samples that contains the k nearest neighbors of every point in the input's adversarial region.

5.1.1 Dataset partitioning

The dataset is partitioned using a binary space partition (BSP) scheme [4]. This involves recursively partitioning the hyperspace into two halves along hyperplanes until a specified criterion is met. This subdivision process creates a hierarchy of regions modeled by a geometric data structure known as a BSP tree, in which objects in space are organized in the form of a binary tree. The leaves of this tree represent the resulting spatial partitions, and the internal nodes represent the splits according to the hyperplanes used during the partitioning process. For our purposes we used random projections [12] (i.e., random hyperplanes) to split the space, and stop partitioning when the number of samples in the partition is at most m, where $k \leq m \ll |\mathcal{S}|$.

Algorithm 5.1 shows how the BSP tree is constructed. Given a dataset $\mathcal{S}$ and a maximum partition size m, the algorithm first checks whether the size of the dataset is less than m. If so, it returns a tree consisting of a single leaf initialized with the dataset $\mathcal{S}$ (line 1-2). Otherwise, it creates a random hyperplane π using the SelectRandomProj method and splits the dataset into two subsets according to π (line 3-5). The algorithm then recursively invokes itself on the two resulting subsets to construct their respective BSP trees (line 6-7). Finally, it returns a node initialized with the splitting hyperplane and the previously constructed subtrees (line 8).

Algorithm 5.1 BuildBSPTree algorithm

Input: $\mathcal{S}$: A dataset, m: The maximum size of a partition

Output: A BSP tree of the dataset

1: **if** $|\mathcal{S}| \leq m$
2: **return** Leaf(S)
3: $\pi \leftarrow$ CreateRandomProj($\mathcal{S}$)
4: `left_dataset` $\leftarrow \{\boldsymbol{s} \in \mathcal{S} \mid \pi(\boldsymbol{s}) \leq 0\}$
5: `right_dataset` $\leftarrow \{\boldsymbol{s} \in \mathcal{S} \mid \pi(\boldsymbol{s}) > 0\}$
6: `left_tree` $\leftarrow$ BuildBSPTree(`left_dataset`, m)
7: `right_tree` $\leftarrow$ BuildBSPTree()`right_dataset`, m)
8: **return** Node(π, `left_tree`, `right_tree`)

Random Projection Selection

One way to split dataset is for example to simply pick two random samples and split the dataset using the perpendicular bisector of the two samples. Another method is to use k-Means with $k = 2$ which will split the dataset in two and the splitting hyperplane would be the perpendicular bisector of the two cluster centers. However, neither of these strategies guarantees a balanced tree or partitions containing at least k samples.

To satisfy both requirements, one approach is to first partition the dataset using the perpendicular bisector of two randomly chosen points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$. The splitting hyperplane is then translated along the line segment joining $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ moving towards the point with the larger number of closer samples, until the two resulting partitions differ in size by at most one sample. With this approach BuildBSPTree subdivide the dataset into at most

$$2^{\left(\left\lceil \log_2^{|S|/m} \right\rceil\right)}$$

partition each of which contains at least $m - 1$ samples.

5.1.2 Searching in the BSP Tree

Once constructed, the BSP tree can be used to efficiently search for samples belonging to a partition or samples within a distance r of a sample of interest $\boldsymbol{x}$—that is, points inside the hypersphere centered at $\boldsymbol{x}$ with radius r.

Samples within a Hypersphere

Samples within a hypersphere can be found by traversing the BSP tree from top to bottom, selecting only the partitions that intersect the hypersphere until leaf nodes are reached. This procedure is described by the recursive method in Algorithm 5.2. Given a BSP tree T, a center $\boldsymbol{x}$, and a radius r of the hypersphere, the algorithm first checks whether T is a leaf node. If so, it returns the set of points within the dataset associated with T that are inside the hypersphere (lines 1-2). Otherwise, if the splitting hyperplane associated with T, denoted $T.hyperplane$, intersects the hypersphere (i.e., the distance between $\boldsymbol{x}$ and $T.hyperplane$ is less than r), the algorithm recursively calls itself on the two half-spaces induced by $T.hyperplane$ to compute the sets of points inside the hypersphere on both sides of $T.hyperplane$, and returns their union (lines 3-4). However, if $T.hyperplane$ does not intersect the hypersphere, the algorithm calls itself on the half-space containing the hypersphere entirely to compute and return the set of points inside the hypersphere (lines 5-8).

Algorithm 5.2 SearchClosePoints algorithm

Input: T: A BSP tree, $\boldsymbol{x}$: Center of the hypersphere, r: Radius of the hypersphre
Output: Set of point inside the hypersphere

1: **if** T is a Leaf
2: ⌊ **return** $\{\boldsymbol{s} \in T.dataset \mid \|\boldsymbol{x} - \boldsymbol{s}\| \leq r\}$
3: **if** Distance$(\boldsymbol{x}, T.hyperplane) \leq r$
4: ⌊ **return** SearchClosePoints$(T.left, \boldsymbol{x}, r) \cup$ SearchClosePoints$(T.right, \boldsymbol{x}, r)$
5: **else if** $T.hyperplane(\boldsymbol{x}) > 0$
6: ⌊ **return** SearchClosePoints$(T.right, \boldsymbol{x}, r)$
7: **else**
8: ⌊ **return** SearchClosePoints$(T.left, \boldsymbol{x}, r)$

Partition of a Point

We can determine the partition a point $\boldsymbol{x}$ belongs to by slightly modifying the SearchClosePoints method, as shown in Algorithm 5.3. The differences with Algorithm 5.2 are as follows: (1) when a leaf node is reached, the entire dataset associated with that leaf is returned; and (2) when the point lies on the splitting hyperplane, the returned samples are those belonging to the partitions sharing that hyperplane as a boundary. Otherwise, Algorithm 5.3 traverses the BSP tree in the same manner as Algorithm 5.2.

Algorithm 5.3 SearchPartition algorithm

Input: T: A BSP tree, $\boldsymbol{x}$: Center of the hypersphere, r: Radius of the hypersphre
Output: Set of point inside the hypersphere

```
1: if T is a Leaf
2:     return T.dataset
3: if T.hyperplane(x) = 0
4:     return SearchPartition(T.left, x, r) ∪ SearchPartition(T.right, x, r)
5: else if T.hyperplane(x) > 0
6:     return SearchPartition(T.right, x, r)
7: else
8:     return SearchPartition(T.left, x, r)
```

5.2 Certifier Optimizations

As highlighted at the beginning of the chapter, Algorithm 4.7, exhibits computational inefficiency due to its exhaustive enumeration of all adversarially valid paths within the input's APP-G before determining the set of labels assigned to points within the adversarial region. This two-stage process leads to significant redundancy, as the algorithm spends considerable effort exploring paths that ultimately yield the same set of labels already encountered along other paths. Recall that a label $l \in \mathcal{L}$ is in included in the output of Algorithm 4.7 if at least one adversarially valid path with k samples and having l among its most common labels is found. Therefore, a more efficient approach would be to find such a path, for each possible label $l \in \mathcal{L}$, and terminate further exploration for paths with the same most frequent label l once such a path is found. This targeted approach avoids redundant computation and significantly reduces the certifier's execution time. Moreover, if a label l is dominant in some adversarially valid path $\mathcal{P} = [\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_m]$ then it will remain so regardless of the order of samples in $\mathcal{P}$. Additionally, observe that the validity polytope of each permutation of samples in $\mathcal{P}$ is entirely contained in the high order voronoi cell [38, 31] whose sites are exactly the samples $\{\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_m\}$ (i.e., the region of space containing points closer to the m samples in $\mathcal{P}$ than any other sample in the training set). Therefore, given a set S of samples we can check the existence of an adversarially valid path composed of samples in S by verifying whether the high order voronoi cell, whose site are samples in S, intersects with the perturbation region. If there is an intersection than it means that there surely exist an adversarially valid path composed of sample in S. Conversely, if they do

not intersect, then no such path exists. So using voronoi cell we can explore just one single path instead of all the permutations of S further reducing the certification time. Additionally, since k-NN is stable for an input $\boldsymbol{x}$ only when every point in $P^{\epsilon}(\boldsymbol{x})$ is classified with the same label, we can terminate the certification process once we found two different labels assigned by k-NN to samples in $P^{\epsilon}(\boldsymbol{x})$. Other optimizations that can accelerate the certification process are:

Opt. 1) If the samples in the neighborhood of the adversarial region share same label l then there is no need to explore the APP-G of the input, and we can safely return a singleton with label l;

Opt. 2) Suppose that in an adversarially path $\mathcal{P}$ having $m < k$ samples, the two most common labels l_1 and l_2 occur t_1 and t_2 times respectively with $t_1 \geq t_2$. Then if $t_1 - t_2 \geq k - m$ it means that l_1 will be among the dominant labels regardless of the last $k - m$ samples of the path. So in this case there is no need to further extend the path and label l_1 can be safely added to the output labels;

Opt. 3) One special case of **Opt. 1)** is when the first $\lceil \frac{k}{2} \rceil$ samples of the path have the same label l. Again extending the path further will not change the dominant label and so l can be safely added to the output labels;

Following these observations, the certifier algorithm, detailed in Algorithm 5.4, optimizes the robustness certification process. Given the BSP tree `Tree` of the dataset, the input sample $\boldsymbol{x}$, the perturbation magnitude ϵ, and the number k of neighbors to search, the algorithm begins by identifying the training samples $\mathcal{S}$ within the perturbed region's neighborhood (lines 1-2). If all samples in $\mathcal{S}$ share the same label l, the algorithm immediately returns the set l (lines 3-4). Otherwise, it constructs the APP-G for $\boldsymbol{x}$ (line 5). The algorithm then examines each unique label l found within $\mathcal{S}$. For each label l, it checks if multiple labels have already been identified; if so, those labels are returned, and certification process terminates. Otherwise, the samples in $\mathcal{S}$ are separated into two distinct sets: $\mathcal{S}_L$, containing samples labeled with l, and $\mathcal{S}_O$, containing samples with alternative labels (line 10-11). To evaluate the dominance of label l, the algorithm checks the existence of a potential path with k samples within $\mathbb{G}_{\boldsymbol{x}}^{A}$ where l remains the predominant label. If such path does not exist, the algorithm concludes that label l cannot be assigned to points within the perturbation region and therefore excludes it from consideration and continue with next label (line 12-14). Conversely, the algorithm iteratively constructs sample sets of cardinality n between $\lceil k/2 \rceil$ and k, ensuring l is among

the most frequent labels within each set (lines 15-28). These sets represent potential paths within $\mathbb{G}^A_{\boldsymbol{x}}$. For a given set size n, the algorithm computes the minimum and maximum possible occurrences of samples labeled l within any path of length n (lines 16-18). Subsequently, for each possible number of occurrences `n_occur`, it identifies the set $\texttt{PS}_L \subseteq \mathcal{S}_L$ containing samples that can be included in a path of length n with the required `n_occur` occurrences of label l (line 20). Then, for each subset $\texttt{VC}_L \subseteq \texttt{PS}_L$ of size `n_occur`, the algorithm attempts to expand it with samples labeled differently to reach the required length n. To do this it constructs the set $\texttt{PS}_O \subseteq \mathcal{S}_O$ containing samples that can extend the set `vertices`, which contains samples in $\texttt{VC}_L$ and their predecessors, without exceeding the required length n (lines 22-23). Afterward, the algorithm iteratively adds to `vertices` each possible subset $\texttt{VC}_O \subseteq \texttt{PS}_O$ of cardinality $n - |\texttt{vertices}|$ (lines 24-25) and checks whether the Voronoi cell whose sites are the samples in the resulting set intersects with the perturbation (line 26). If there is an intersection, it indicates an adversarial path composed of samples within that extended set. The algorithm then checks whether **Opt. 2)** or **Opt. 3)** applies to the samples in the extended set (line 26). If so, the label l is added to the classification, and the algorithm proceeds to the next label (lines 27-28). Finally, after wall the labels are processed returns the identified labels (line 29).

In essence, for each unique label l in the APP-G of the input sample, Algorithm 5.4 iteratively constructs sets of samples of cardinality n ($\lceil k/2 \rceil \leq n \leq k$). To achieve this it first builds sets up to size n comprised of samples labeled with l and their predecessors, and then extends these sets with all possible samples labeled differently, while ensuring label l remains the dominant label. Then, it verifies whether an adversarial valid path exists in $\mathbb{G}^A_{\boldsymbol{x}}$ of the input by checking for an intersection between the adversarial region and Voronoi cells whose sites are the samples in the constructed sets, and if either optimization **Opt. 2)** or **Opt. 3)** can be applied. If both conditions are met, label l is added to the classification.

ExistsPath and ApproxMaxOccurences Methods

One optimization used in Algorithm 5.4 that allows us to skip the exploration of the graph for a specific label l is to check whether a path of length k can exist in the APP-G of the input. One way to efficiently compute this is to find the set A of samples with size n such that l is a dominant label regardless of the existence of an adversarially valid

Algorithm 5.4 OptCertifier algorithm

Input: `Tree`: BSP tree of the dataset, $\boldsymbol{x}$: The input sample
ϵ: The perturbation magnitude, k: The number neighbors
Output: Set of possible labels that point in $P^{\epsilon}(\boldsymbol{x})$ can be classified with

$\mathcal{S} \leftarrow$ `Tree`.GetSamples($\boldsymbol{x}$)
`labels` $\leftarrow$ distinct labels among samples in $\mathcal{S}$
if |`labels`| = 1
  return `labels`
$\mathbb{G}^A_{\boldsymbol{x}} \leftarrow$ CreateAPPGraph($\mathcal{S}, \boldsymbol{x}, \epsilon$)
`classification` $\leftarrow \emptyset$
for all `label` **in** `labels` **do**
  if |`classification`| > 1
    return `classification`
  $\mathcal{S}_L \leftarrow \{\boldsymbol{s}_i \in \mathcal{S} | \boldsymbol{s}_i$.Label = `label`$\}$
  $\mathcal{S}_O \leftarrow \{\boldsymbol{s}_i \in \mathcal{S} | \boldsymbol{s}_i$.Label $\neq$ `label`$\}$
  if not ExistsPath(`label`, $\mathcal{S}_L \cup \mathcal{S}_O, k$)
    go to next label
  for all n **in** $[\lceil k/2 \rceil, \ldots, k]$ **do**
    `max_label_occur` $\leftarrow$ ApproxMaxOccurences(`label`, $\mathcal{S}_L, n$)
    `max_label_occur` $\leftarrow min(n,$ `max_label_occur`$)$
    `min_label_occur` $\leftarrow \left\lceil \frac{n}{|\texttt{labels}|} \right\rceil$
    for all `n_occur` **in** [`max_label_occur`, ..., `min_label_occur`] **do**
      $\text{PS}_L \leftarrow$ FindValidSamples($\emptyset, \mathcal{S}_L, n\_occur, n$)
      for all VC_L **in** Combinations(PS_L, `n_occur`) **do**
        `vertices` $\leftarrow$ samples in VC_L and their predecessors
        $\text{PS}_O \leftarrow$ FindValidSamples(`vertices`, $\mathcal{S}_O, n\_occur, n$)
        for all VC_O **in** Combinations($\text{PS}_O, n - n\_occur$) **do**
          `all_vertices` $\leftarrow$ samples in $\text{VC}_L \cup \text{VC}_O$ and their predecessors
          if ExistsValidPath(`all_vertices`, $\mathbb{G}^A_{\boldsymbol{x}}$) and
          Opt. 2) or **Opt. 3)** applicable
            `classification` $\leftarrow$ `classification` $\cup$ {`label`}
            go to next label
return `classification`

path comprising samples in A. This is a combinatorial optimization problem that can be solved using Binary Integer Programming (BIP) [45]. As an optimization problem it can be formulated as follows: if there are n samples in $\mathbb{G}_{\boldsymbol{x}}^{A}$ and x_i is a boolean variable that indicates whether sample $\boldsymbol{s}_i$ is included in A then the problem is to maximize

$$\sum_{i=1}^{i=n} v_i \cdot x_i$$

where v_i is 2 if sample $\boldsymbol{s}_i$ is labeled with l otherwise is 1, subject to the following constraints

1. If sample $\boldsymbol{s}_i$ is included A then its predecessors must be included as well. This constraint can be formulated with

$$\forall \boldsymbol{s}_j \in \texttt{pred}(\boldsymbol{s}_i) \quad x_i - x_j \leq 0$$

2. the number of samples nn A with a label different from l must not exceed the number of samples labeled with l. This can be formulated as

$$\forall l_j \in \mathit{Labels},\ l_j \neq l \quad \sum_{j \in I_j} x_j - \sum_{i \in I} x_i \leq 0$$

   where Labels is the set of unique labels in $\mathbb{G}_{\boldsymbol{x}}^{A}$, I_j and I are the set indices of the samples with label l_j and l respectively.

3. The number of selected samples is exactly k. This constraint can be expressed as

$$\sum_{i=1}^{i=n} x_i = k$$

If none of the variable corresponding to samples labeled with l is set to 1 then it means that no path with length equal to k exists where l is a dominant label hence the maximum length is 0. Otherwise, such a path exists. This is exactly the problem solved by the ExistsPath to verify the existence of path with k samples in which the label l is dominant. Furthermore, this linear programming formulation also yields the maximum occurrences of label l under a specific path length constraint within the APP-G. Consequently, the ApproxMaxOccurences method employs the same linear problem to approximate the maximum occurrences of label l in any path of a given length. However, because we do not explicitly check for the existence of an adversarially valid path,

ExistsPath may incorrectly assert existence when, in actuality, it does not hold. This constitutes the primary source of inefficiency for the algorithm. Specifically, when ExistsPath returns true, but the desired path does not exist, the certifier wastes significant time searching for a path that will never be found, leading to significant increased runtime, especially when the desired length is large. Some solution to this problem are given in Chapter 7.

FindValidSamples Methods

This method is used by the optimized certifier to identify the samples to use in order to construct the sets of size n in line 20 and 23. It takes as input a set A to be extended, the set S containing the samples that we choose from, the maximum number o of occurrences for every label in the extended set and the size n of the resulting set. For any $s \in S$ let $B(s) = A \cup \{s\} \cup \texttt{pred}(s)$ then FindValidSamples returns the samples $s \in S$ such that

- $|B(s)| \leq n$
- The number of occurences of any label in $B(s)$ is not greater than o

6 Experimental Evaluation

We implemented the certifier entirely in Python adopting the optimizations described in Section 5.2 and the source code is publicly available in author GitHub repository [47]. Our experimental evaluation was run on Arch Linux in WSL2 on Windows 11 machine with Intel Core i7-1260P 2.10 GHz CPU with 12 physical core and 32 GB of RAM.

This chapter will present the findings of our experimental evaluations of *robustness*, *stability*, and *individual fairness* of k-NN on different dataset. We compared the results with the tool kNAVe [20].

6.1 Datasets

For our experiments, we considered 7 standard datasets used in the most recent stream of works studying the robustness of k-NNs [59, 58, 62, 52, 54, 51], on which we also carried out stability tests, and 4 datasets used for fairness verification of deep neural networks [33].
The datasets used to certify robustness and stability properties of k-NNs are the following:

- **Australian**: dataset concerning credit card applications, consisting of categorical and numerical features. All attribute names and values have been changed to meaningless symbols to protect confidentiality of the data.

- **BreastCancer**: classic and very easy binary classification dataset for diagnostic purpose, where the classification is whether the cancer is benign or malignant.
- **Diabetes**: dataset taken from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict, based on diagnostic measurements, whether a patient has diabetes.
- **Fourclass**: non-linear separable dataset consisting of 862 two-dimension data points.
- **Letter**: letter recognition dataset, where the objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet.
- **Pendigits**: digit database created by collecting 250 samples from 44 writers. The samples written by 30 writers are used for training and the digits written by the other 14 are used for testing.
- **Satimage**: database consisting of the multi-spectral values of pixels in 3×3 neighbourhoods in a satellite image, and the classification associated with the central pixel in each neighbourhood. The aim is to predict this classification, given the multi-spectral values.

The following dataset were for individual fairness verification:

- **Adult**: dataset extracted from the 1994 US Census database. Every sample assigns a yearly income (below or above 50K US$) to an individual based on personal attributes such as gender, race, and occupation.
- **Compas**: dataset containing data collected on the use of the COMPAS risk assessment tool in Broward County, FL. Each sample predicts the risk of recidivism for individuals based on personal attributes and criminal history.
- **Crime**: dataset containing socio-economic, law enforcement, and crime data for communities within the US. Each sample indicates whether a community is above or below the median number of violent crimes per population.
- **German**: dataset containing individuals with either a good or bad credit score.

The main characteristics of these datasets, as well as their accuracy, are summarized in Table 6.1 and Table 6.2, respectively.

Name	#training	#test	#features	#features (one-hot)	#classes
Australian	483	207	14	39	2
BreastCancer	479	204	10	10	2
Diabetes	556	230	8	8	2
Fourclass	604	258	2	2	2
Letter	15000	5000	16	16	26
Pendigits	7494	3498	16	16	10
Satimage	4435	2000	36	36	6
Adult	30162	15060	14	103	2
Compas	4222	1056	10	370	2
Crime	1595	399	102	147	2
German	800	200	20	56	2

Table 6.1: Datasets used in experimental evaluations.

	Accuracy (%)			
Name	$k = 1$	$k = 3$	$k = 5$	$k = 7$
Australian	77.8	80.2	82.6	82.6
BreastCancer	92.6	94.6	93.6	93.6
Diabetes	70.9	72.2	70.0	71.3
Fourclass	100	100	100	100
Letter	95.7	94.6	94.2	94.3
Pendigits	97.7	97.8	97.5	97.5
Satimage	88.8	90.3	89.5	90.1
Adult	78.4	81.1	81.8	82.3
Compas	58.4	59.1	60.2	61.1
Crime	77.7	79.7	80.7	81.2
German	73.0	71.5	74.5	77.0

Table 6.2: Accuracy of each dataset for $k \in \{1, 3, 5, 7\}$.

6.2 Perturbations

For our robustness and stability experiments we considered the ℓ_2-norm perturbation discussed in Subsection 2.2.3 with a magnitude ϵ ranging in $[0.001, 0.05]$, i.e., numerical features can be altered from $\pm 0.1\%$ to $\pm 5\%$. Individual fairness for Adult, Compas, Crime and German datasets has been evaluated for the Noise-Cat similarity relation defined in Subsection 2.2.5, where we applied a ℓ_2-norm Noise perturbation to numerical features, also in this case with ϵ ranging in $[0, 0.05]$, and a Cat perturbation for the following categorical sensible features:

- For Adult and German: *gender*, having 2 categories;
- for Compas: *race*, having 2 categories;
- for Crime: *state*, having 46 categories.

6.3 Preprocessing

We preprocessed each dataset, in accordance with Ruoss et al. [44], as follows: (i) rows and columns with missing values are dropped; (ii)when needed, datasets are split into training ($\approx$ 70-80%) and test ($\approx$ 20-30%) sets. (Letter, Pendigits and Satimage are the only datasets having an independent test set) (iii) categorical features are one-hot encoded; (iv) numerical features are scaled to $[0, 1]$ range.
Thus, if the space X consists of n-dimensional real vectors scaled to $[0, 1]$, an ℓ_2-norm perturbation having magnitude $\epsilon \geq 0$ on a feature vector $\boldsymbol{x} \in X$ defines the adversarial region:

$$P_{\infty}^{\epsilon}(\boldsymbol{x}) \triangleq \{\boldsymbol{w} \in \mathbb{R}^n \mid \forall i \in [1, \ldots, n].\, \boldsymbol{w}_i \in [\boldsymbol{x}_i - \epsilon, \boldsymbol{x}_i + \epsilon]\}$$

6.4 Results

We performed our experiments $k \in \{1, 3, 5, 7\}$, and, following the standard practice for the k-NN algorithm, we avoided even values of k as they are more likely to introduce tie votes in the classification. To make the certification process more efficient:

- We generated the BSP tree of test dataset with partition size 100 only once and used it for the test samples;

- We compute the APP-G of the input only once since it is the same for each value of k;

- We processed the test samples concurrently using 12 concurrent process (i.e., the number of the cpu physical core)

This allowed us to save a considerable amount of time for the certification with multiple values of k.

Table 6.3 shows the average certification time per input sample and perturbation magnitude ϵ. These runtimes are computed taking the average time for executing the optimized certifier for all $k \in \{1, 3, 5, 7\}$ on a given sample x and for a given perturbation magnitude ϵ. For all datasets we run our certifier with magnitude $\epsilon \in \{0.001, 0.002, 0.005, 0.01, 0.02, 0.03, 0.04, 0.05\}$. However, for *Letter, SatImage, Adult* and *Crime*, we only used magnitudes $\epsilon \in 0.001, 0.002, 0.005, 0.01$ and *Diabetes, Pendigits* we used $\epsilon \in 0.001, 0.002, 0.005, 0.01, 0.02, 0.030.04$ due to high runtimes at higher perturbations. This is because the certifier does not scale well for these datasets, as for some labels l it asserts the existence for a path with $k = 7$ sample where l is dominant when in actuality it doesn't because the solution found with liner programming is not actually an adversarially valid path. Consequently, the certifier spends a lot of time to find a path with $k = 7$ samples even when such a path does not exist.

Name	N. of magnitude ϵ	Avg. Time per ϵ
Australian	8	2 s
BreastCancer	8	2.75 s
Diabetes	7	3 m
Fourclass	8	1.2 s
Letter	4	2 m
Pendigits	8	15 m
Satimage	4	2 m
Adult	8	60 m
Compass	8	2.5 m
Crime	4	8 m
German	8	5 s

Table 6.3: Average certification time.

The results for stability, robustness, and individual fairness are shown in the following two subsections.

6.4.1 Stability and Robustness Certification

The Tables below report the percentages of samples in the test dataset T certified to be stable and robust, namely, $\textsc{Stability}(C, \mathcal{T})$ and $\textsc{Robustness}(C, \mathcal{T})$ metrics defined in (2.2). These results show that for majority of dataset, the certifier was able to certify the stability of more than 90% of test samples for $\epsilon \leq 0.02$ which means the k-NN is a fairly robust classifier against adversarial attacks. As expected, higher perturbation magnitude ϵ values leads to a decrease in the stability and robustness percentages since stronger perturbations result in adversarial region that most likely intersect with the decision boundaries. In particular, we observe that *Diabetes* exhibits the worst stability scores: while it is not possible to give a sure interpretation, the low accuracy ($\simeq$ 70%) hints that diagnosis of diabetes is a hard task for which kNN does not perform well. Consistent with theoretical predictions, for a given value of ϵ, higher values of k leads to more stable classifications. This trend is observed for all dataset except for *Letter* probably due to the fact the decision boundaries are not well separated due to the data distribution making the k-NN particularly susceptible to adversarial attacks.

The tables also present a comparison with the tool kNAVe [20]. Our certifier outperforms kNAVe on all datasets, as expected, since kNAVe is not exact, especially at higher ϵ values, where ties in abstract distance become more prevalent.

Australian Certified Stability (%)								
ϵ	$k = 1 \uparrow$		$k = 3 \uparrow$		$k = 5 \uparrow$		$k = 7 \uparrow$	
	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert
0.001	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.002	99.0	99.0	99.5	99.5	100.0	100.0	100.0	100.0
0.005	97.1	99.0	98.6	99.5	97.6	100.0	99.0	100.0
0.01	95.2	97.5	96.6	98.0	96.1	98.5	97.1	99.5
0.02	92.8	96.1	91.3	97.1	91.8	96.6	97.1	98.5
0.03	91.3	94.2	88.9	94.2	89.4	93.7	94.7	96.1
0.04	88.4	93.2	85.0	92.2	87.9	93.2	88.9	95.1
0.05	86.0	93.2	84.1	90.3	85.5	91.7	86.0	94.6

Australian Certified robustness (%)								
ϵ	$k = 1 \uparrow$		$k = 3 \uparrow$		$k = 5 \uparrow$		$k = 7 \uparrow$	
	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert
0.001	77.8	77.8	80.2	80.2	82.6	82.6	82.6	82.6
0.002	77.8	77.8	79.7	79.7	82.6	82.6	82.6	82.6
0.005	77.8	77.8	79.2	79.7	81.2	82.6	81.6	82.6
0.01	75.4	77.2	77.8	78.7	79.7	82.1	80.7	82.1
0.02	73.9	76.3	75.8	78.2	77.8	80.1	79.2	81.1
0.03	72.9	74.3	74.4	77.8	76.8	78.7	77.8	79.7
0.04	70.5	73.4	72.5	76.3	76.3	78.2	75.4	79.7
0.05	68.6	73.4	71.5	75.3	74.4	77.8	73.9	79.7

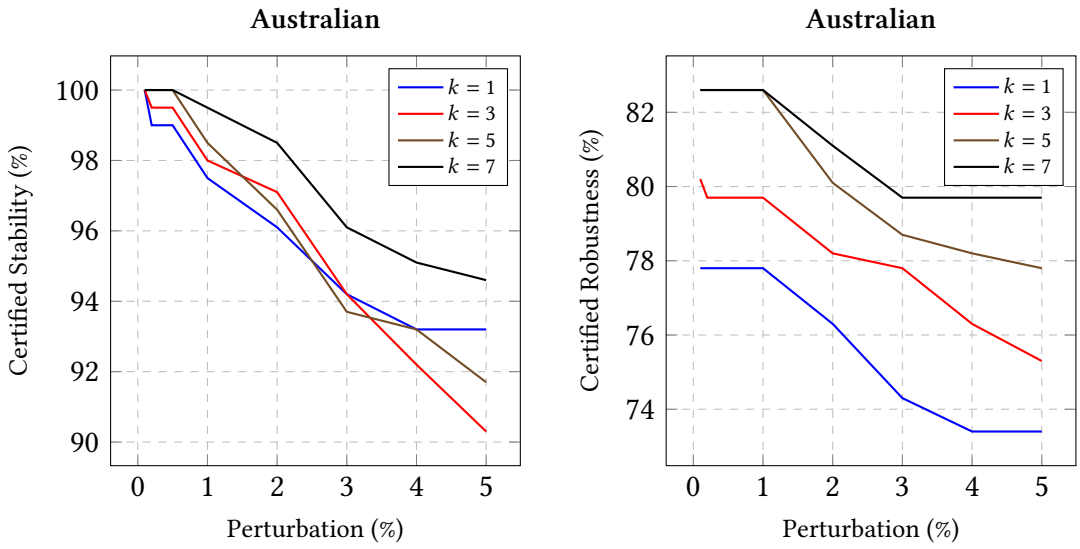


Figure 6.1: Certified stability and robustness on the whole Australian test set.

BreastCancer Certified Stability (%)								
ϵ	$k = 1 \uparrow$		$k = 3 \uparrow$		$k = 5 \uparrow$		$k = 7 \uparrow$	
	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert
0.001	99.5	99.5	100.0	100.0	99.0	99.0	99.5	99.5
0.002	98.5	99.5	100.0	100.0	99.0	99.0	99.0	99.0
0.005	96.6	98.0	99.5	100.0	98.0	99.0	98.5	99.0
0.01	96.6	96.5	99.0	99.5	96.6	98.0	97.1	98.0
0.02	92.2	96.5	93.1	97.0	91.7	98.0	93.1	97.5
0.03	86.8	94.1	88.7	94.6	87.7	94.1	82.2	96.0
0.04	80.4	91.6	83.3	92.1	83.3	93.1	85.3	94.1
0.05	75.0	88.7	77.9	90.1	81.4	91.6	83.8	90.1

BreastCancer Certified robustness (%)								
ϵ	$k = 1 \uparrow$		$k = 3 \uparrow$		$k = 5 \uparrow$		$k = 7 \uparrow$	
	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert
0.001	92.6	92.6	94.6	94.6	93.6	93.6	93.1	93.1
0.002	92.2	92.6	94.6	94.6	93.6	93.6	92.6	92.6
0.005	91.2	92.1	94.1	94.6	93.1	93.6	92.6	92.6
0.01	91.2	91.2	93.6	94.1	91.7	93.1	92.2	92.6
0.02	88.2	91.1	89.7	92.1	88.7	93.1	89.7	92.1
0.03	84.3	89.7	86.8	90.6	86.8	90.1	87.3	91.1
0.04	78.9	88.7	82.4	89.7	82.4	89.7	84.8	90.1
0.05	74.5	85.7	77.5	88.2	80.9	89.2	83.3	90.1

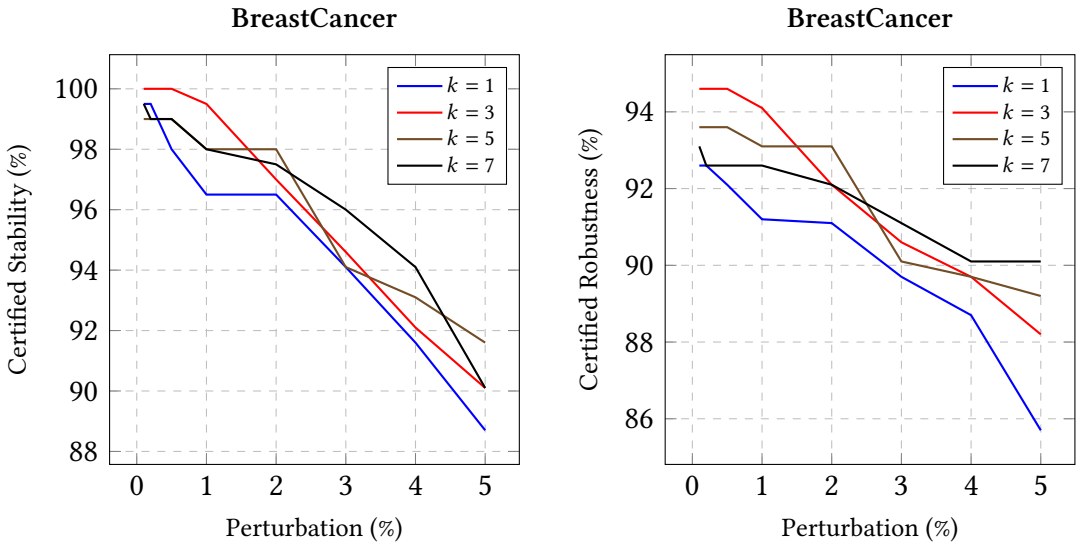


Figure 6.2: Certified stability and robustness on the whole BreastCancer test set.

Diabetes Certified Stability (%)								
ϵ	$k = 1 \uparrow$		$k = 3 \uparrow$		$k = 5 \uparrow$		$k = 7 \uparrow$	
	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert
0.001	95.2	97.8	96.1	97.3	98.7	98.6	98.7	99.1
0.002	89.1	94.7	91.3	93.4	94.8	97.3	94.3	96.9
0.005	78.7	85.6	80.4	89.1	80.0	91.3	77.4	91.7
0.01	62.6	77.8	59.6	80.4	56.5	81.3	54.3	80.0
0.02	31.3	59.1	25.7	59.1	23.9	61.3	23.0	66.0
0.03	13.0	40.4	9.1	44.3	7.8	47.8	6.1	51.7
0.04	7.8	30.4	4.3	33.4	4.8	38.2	3.0	43.4

Diabetes Certified robustness (%)								
ϵ	$k = 1 \uparrow$		$k = 3 \uparrow$		$k = 5 \uparrow$		$k = 7 \uparrow$	
	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert
0.001	69.1	70.0	71.3	71.7	69.6	69.6	71.3	71.3
0.002	65.7	68.6	68.7	70.0	67.4	68.2	69.6	71.3
0.005	60.0	63.9	62.2	67.3	61.3	67.3	59.1	67.8
0.01	49.1	58.6	49.6	63.0	47.0	63.0	46.5	61.7
0.02	25.2	45.6	23.0	47.8	22.6	50.0	21.7	54.3
0.03	11.7	33.4	8.7	37.3	7.4	41.3	6.1	44.7
0.04	7.4	27.3	4.3	29.5	4.3	33.4	3.0	38.2

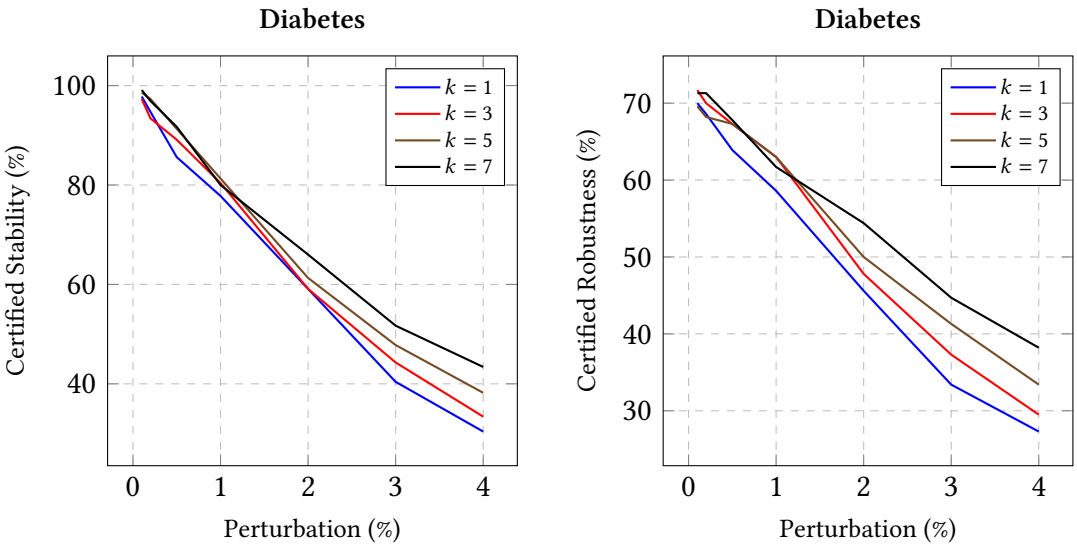


Figure 6.3: Certified stability and robustness on the whole Diabetes test set.

Fourclass Certified Stability (%)								
ϵ	$k = 1 \uparrow$		$k = 3 \uparrow$		$k = 5 \uparrow$		$k = 7 \uparrow$	
	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert
0.001	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.002	99.6	99.6	99.6	99.6	100.0	100.0	100.0	100.0
0.005	99.6	99.6	99.6	99.6	99.2	99.2	98.8	99.6
0.01	99.2	99.6	99.6	99.6	98.4	98.8	97.7	98.4
0.02	93.4	96.5	91.5	97.6	86.8	97.2	85.3	96.8
0.03	78.3	89.1	75.6	86.8	75.6	89.1	72.9	88.3
0.04	62.0	79.0	60.5	79.4	59.3	79.8	55.8	80.2
0.05	45.0	71.3	42.6	70.9	40.3	70.5	37.6	72.0

Fourclass Certified robustness (%)								
ϵ	$k = 1 \uparrow$		$k = 3 \uparrow$		$k = 5 \uparrow$		$k = 7 \uparrow$	
	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert
0.001	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.002	99.6	99.6	99.6	99.6	100.0	100.0	100.0	100.0
0.005	99.6	99.6	99.6	99.6	99.2	99.2	98.8	99.6
0.01	99.2	99.6	99.6	99.6	98.4	98.8	97.7	98.4
0.02	93.4	96.5	91.5	97.6	86.8	97.2	85.3	96.8
0.03	78.3	89.1	75.6	86.8	75.6	89.1	72.9	88.3
0.04	62.0	79.0	60.5	79.4	59.3	79.8	55.8	80.2
0.05	45.0	71.3	42.6	70.9	40.3	70.5	37.6	72.0

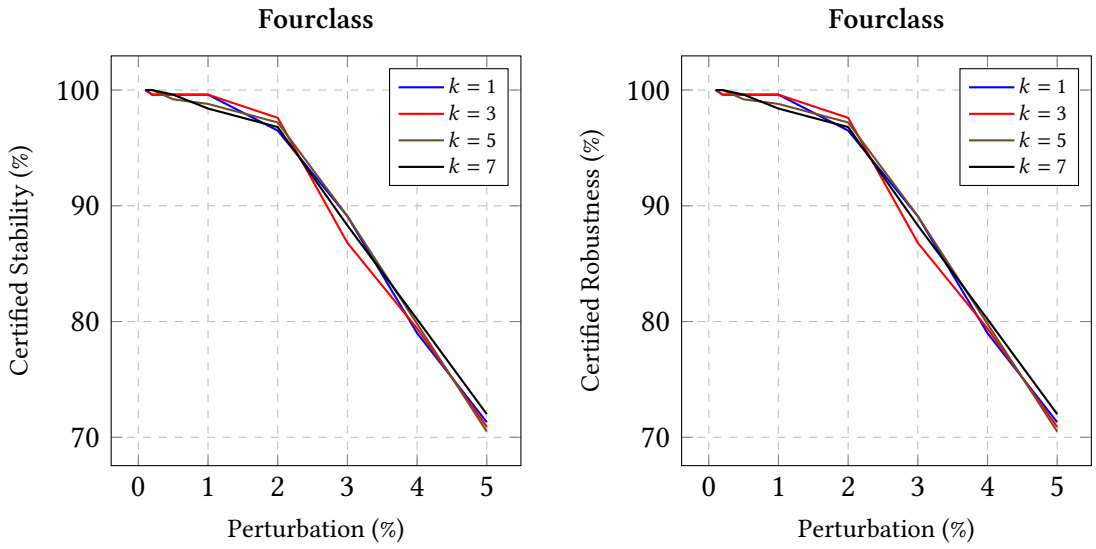


Figure 6.4: Certified stability and robustness on the whole Fourclass test set.

Letter Certified Stability (%)								
ϵ	$k = 1 \uparrow$		$k = 3 \uparrow$		$k = 5 \uparrow$		$k = 7 \uparrow$	
	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert
0.001	98.0	98.0	96.3	96.3	95.2	95.2	95.6	95.7
0.002	96.7	97.8	94.4	96.0	92.7	94.7	92.1	94.7
0.005	91.7	94.0	88.6	92.3	85.4	90.4	83.1	90.4
0.01	82.7	88.7	75.1	86.4	69.5	84.4	65.2	83.4

Letter Certified robustness (%)								
ϵ	$k = 1 \uparrow$		$k = 3 \uparrow$		$k = 5 \uparrow$		$k = 7 \uparrow$	
	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert
0.001	94.7	94.7	93.6	93.6	92.9	92.9	93.0	93.0
0.002	93.9	94.6	92.3	93.4	91.1	92.5	90.5	92.4
0.005	90.4	92.2	87.8	90.9	84.8	89.3	82.7	89.3
0.01	82.2	87.9	75.0	85.9	69.4	83.8	65.1	82.9

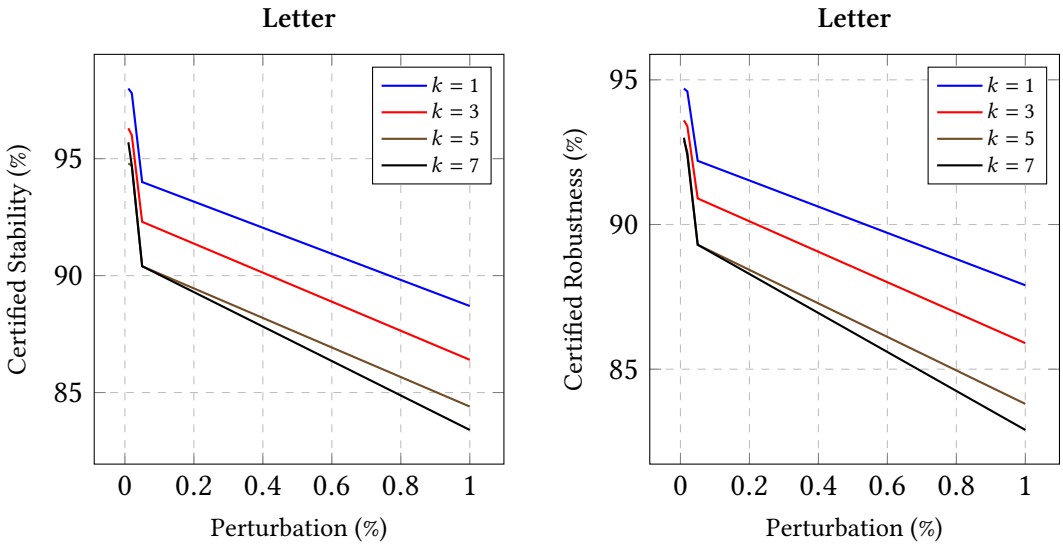


Figure 6.5: Certified stability and robustness on the whole Letter test set.

Pendigits Certified Stability (%)								
ϵ	$k = 1 \uparrow$		$k = 3 \uparrow$		$k = 5 \uparrow$		$k = 7 \uparrow$	
	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert
0.001	99.6	99.6	99.5	99.6	99.4	99.4	99.6	99.7
0.002	99.3	99.5	99.1	99.3	99.1	99.2	99.2	99.4
0.005	98.3	98.9	98.1	98.7	98.1	98.7	97.8	98.9
0.01	96.7	97.7	96.1	97.8	95.7	97.8	95.0	97.8
0.02	91.4	95.2	90.5	95.2	89.5	95.2	89.1	95.1
0.03	82.0	91.7	82.0	92.1	81.0	92.2	80.0	92.4
0.04	71.3	86.5	71.5	88.9	71.0	89.2	70.2	89.3

Pendigits Certified robustness (%)								
ϵ	$k = 1 \uparrow$		$k = 3 \uparrow$		$k = 5 \uparrow$		$k = 7 \uparrow$	
	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert
0.001	97.6	97.6	97.5	97.6	97.4	97.4	97.4	97.4
0.002	97.4	97.5	97.5	97.5	97.3	97.3	97.2	97.3
0.005	96.8	97.2	96.8	97.2	96.7	97.0	96.3	97.0
0.01	95.6	96.4	95.3	96.6	94.9	96.4	94.1	96.3
0.02	90.9	94.4	90.2	94.5	89.2	94.5	88.7	94.2
0.03	81.7	91.2	81.7	91.6	80.8	91.7	79.8	91.8
0.04	71.2	86.2	71.4	88.5	71.0	88.9	70.2	89.0

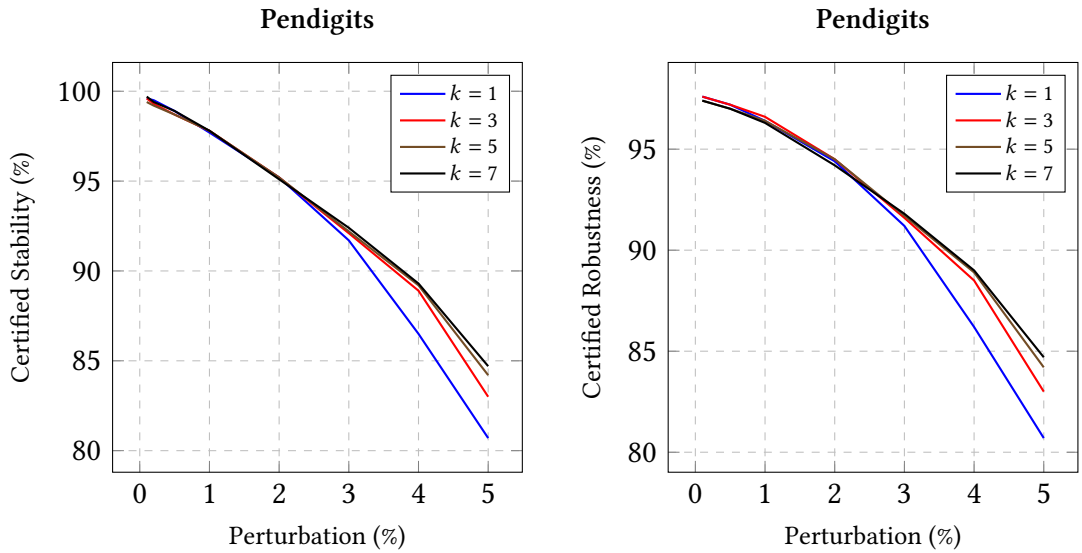


Figure 6.6: Certified stability and robustness on the whole Pendigits test set.

SatImage Certified Stability (%)								
ϵ	$k = 1 \uparrow$		$k = 3 \uparrow$		$k = 5 \uparrow$		$k = 7 \uparrow$	
	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert
0.001	93.8	96.0	93.8	95.7	93.7	95.9	94.3	96.3
0.002	88.7	92.9	88.9	93.4	88.9	93.9	88.1	94.1
0.005	73.7	84.4	72.9	86.0	72.4	88.0	72.4	88.7
0.01	56.0	71.3	54.9	74.9	54.0	77.7	54.2	79.0

SatImage Certified robustness (%)								
ϵ	$k = 1 \uparrow$		$k = 3 \uparrow$		$k = 5 \uparrow$		$k = 7 \uparrow$	
	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert
0.001	86.1	87.0	87.4	88.5	87.0	88.2	87.5	88.6
0.002	83.0	85.9	84.3	87.3	84.1	87.1	83.4	87.5
0.005	71.7	80.1	71.7	82.4	71.0	83.8	70.8	83.9
0.01	55.4	69.7	54.6	73.4	53.8	75.6	53.8	76.8

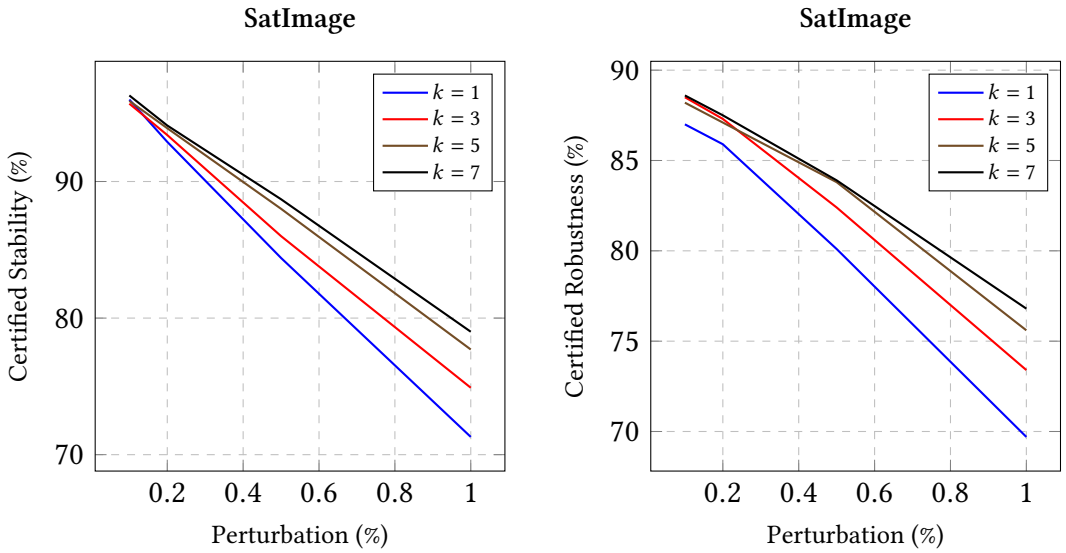


Figure 6.7: Certified stability and robustness on the whole SatImage test set.

6.4.2 Individual Fairness Certification

Tables below reports the percentages of provable individual fairness for all the test samples ranging in each ground truth test dataset T, namely, $\textsc{Fairness}(C, T)$ metric defined in (2.3). As for stability and robustness, we performed our experiments with $k \in \{1, 3, 5, 7\}$, thus avoiding even values of k as they are more likely to introduce tie votes in the classification. Our experiments show that k-NN predications are fair for the *German* and *Adult* on the sensible *gender* category since when $\epsilon = 0$, that is no perturbation on the numerical feature but only on the chosen categorical feature, the percentage of stable samples is already relatively high. On the other hand *Compas* and especially *Crime* are rather unfair on the sensible *race* and *state* category respectively.

Adult Certified Individual Fairness (%)								
ϵ	$k = 1 \uparrow$		$k = 3 \uparrow$		$k = 5 \uparrow$		$k = 7 \uparrow$	
	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert
0	95.4	95.4	97.3	97.3	97.7	97.7	97.8	97.8
0.001	92.3	93.8	94.6	96.1	95.3	96.6	95.3	96.7
0.002	89.9	92.4	92.0	94.7	93.0	95.5	93.2	95.7
0.005	82.4	88.5	84.7	91.5	85.9	92.7	86.5	93.2
0.01	72.7	82.7	75.5	86.5	76.5	88.4	77.1	89.4

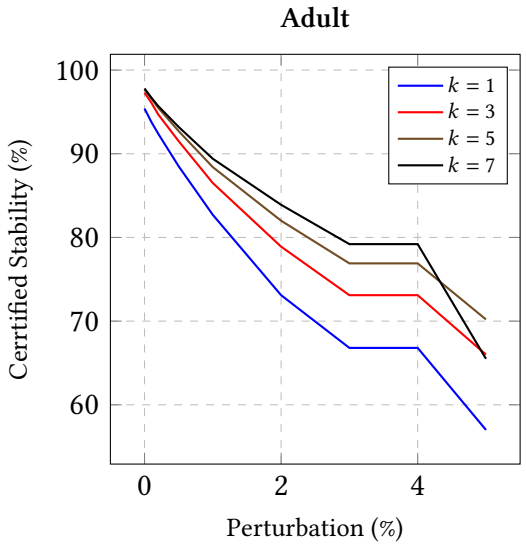


Figure 6.8: Certified fairness on the whole Adult test set.

Compas Certified Individual Fairness (%)								
ϵ	$k = 1 \uparrow$		$k = 3 \uparrow$		$k = 5 \uparrow$		$k = 7 \uparrow$	
	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert
0	58.3	58.3	62.7	62.6	69.1	69.1	71.1	71.1
0.001	48.2	51.0	54.5	57.7	61.1	64.8	62.5	66.1
0.002	45.6	49.3	50.9	55.7	56.8	62.2	57.6	64.0
0.005	35.3	45.4	41.4	51.4	45.4	55.7	46.1	58.4
0.01	25.7	35.4	30.7	42.5	32.8	47.9	36.4	51.4
0.02	17.8	27.2	20.9	4.39	23.9	41.0	27.5	43.7

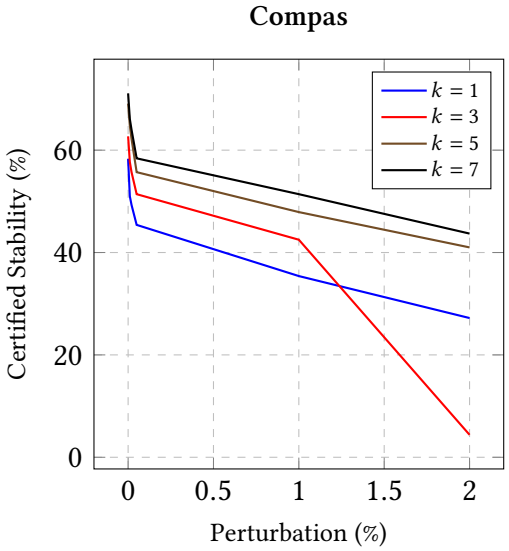


Figure 6.9: Certified fairness on the whole Compas test set.

Crime Certified Individual Fairness (%)								
ϵ	$k = 1 \uparrow$		$k = 3 \uparrow$		$k = 5 \uparrow$		$k = 7 \uparrow$	
	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert
0	7.5	7.5	13.0	13.0	15.8	15.8	18.5	18.5
0.001	6.5	7.01	12.3	12.5	14.5	15.0	17.8	18.0
0.002	6.0	6.5	11.5	11.7	13.3	14.2	16.0	17.2
0.005	4.8	6.0	9.8	11.2	10.8	13.2	13.0	15.7
0.01	4.0	5.2	5.0	10.0	7.0	11.5	7.8	13.7

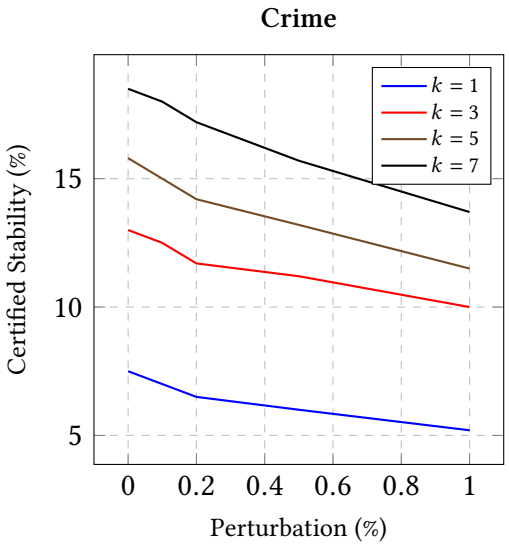


Figure 6.10: Certified fairness on the whole Crime test set.

German Certified Individual Fairness (%)								
ϵ	$k = 1 \uparrow$		$k = 3 \uparrow$		$k = 5 \uparrow$		$k = 7 \uparrow$	
	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert	kNAVe	OptCert
0	85.0	85.0	83.0	83.0	83.5	83.5	82.5	82.5
0.001	85.0	85.0	83.0	83.0	83.0	83.0	82.0	82.5
0.002	85.0	85.0	82.5	83.0	82.5	82.5	82.0	82.0
0.005	85.0	85.0	81.5	83.0	82.5	82.5	81.5	81.5
0.01	83.0	84.0	78.0	80.5	81.0	81.5	81.5	81.5
0.02	80.0	81.5	75.5	79.0	78.0	80.5	78.5	81.5
0.03	76.0	80.0	72.5	77.0	75.0	79.0	73.5	79.5
0.04	73.0	78.5	71.5	76.0	72.5	76.5	71.0	78.0
0.05	72.0	77.0	68.5	76.0	69.0	75.5	69.5	77.0

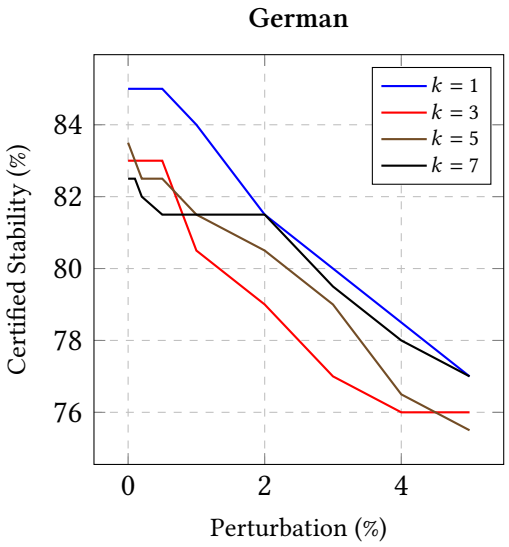


Figure 6.11: Certified fairness on the whole German test set.

7

Conclusion and Future Works

The purpose of this work was to certify the stability and robustness of the k-NN algorithm against adversarial attacks, specifically to determine whether the prediction of the k-NN classifier remains consistent when the features of the input sample are slightly perturbed. To this end we first modified the way the standard k-NN algorithm finds the k closest samples leveraging the relation of *relative proximity* of training samples to the input $\boldsymbol{x}$ without explicitly computing Euclidean distances. We then developed a novel algorithm that, given an adversarial attack modeled as small region of space around the input $\boldsymbol{x}$, computes the set of labels that points within the adversarial region can be classified with by k-NN. If this set contains only a single label then we can guarantee with absolute certainty the stability of k-NN for the input $\boldsymbol{x}$ with respect to the adversarial region. Additionally, if this single label coincides with the ground truth of $\boldsymbol{x}$, we can also conclude that the k-NN classifier is also robust for $\boldsymbol{x}$ within the defined adversarial region. We experimented with our algorithm on different standard datasets for k-NN evaluation, showing that our approach generally scales well and is effective, and that k-NN is in general robust for numerical perturbations of up to ±3%. However, further improvements to this algorithm are possible, particularly aiming to reduce the certification runtime. For example, leveraging high-order Voronoi diagram properties to better approximate the upper bound on path length and terminate path exploration earlier is worth exploring. As future challenge would be interesting to adapt the algorithm to distance metrics other than Euclidean distance, like the Manhattan distance.

References

[1] A. Alanazi, "Using machine learning for healthcare challenges and opportunities," *Informatics in Medicine Unlocked*, vol. 30, p. 100924, Jan. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352914822000739

[2] E. Alpaydin, *Introduction to Machine Learning, fourth edition*, ser. Adaptive Computation and Machine Learning series. MIT Press, 2020. [Online]. Available: https://books.google.it/books?id=tZnSDwAAQBAJ

[3] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," 2016. [Online]. Available: https://arxiv.org/abs/1606.06565

[4] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Santa Clara, CA, USA: Springer-Verlag TELOS, 2008.

[5] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, *Evasion Attacks against Machine Learning at Test Time*. Springer Berlin Heidelberg, 2013, p. 387–402. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40994-3_25

[6] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," 2013. [Online]. Available: https://arxiv.org/abs/1206.6389

[7] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 2154–2156. [Online]. Available: https://doi.org/10.1145/3243734.3264418

[8] G. Bontempi, M. Birattari, and H. Bersini, "Lazy learning for local modelling and control design," *International Journal of Control*, vol. 72, no. 7-8, pp. 643–658, 1999.

[9] S. Calzavara, P. Ferrara, and C. Lucchese, "Certifying decision trees against evasion attacks by program analysis," in *Proc. 25th European Symposium on Research, ESORICS 2020*, ser. LNCS. Springer, 2020.

[10] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 39–57.

[11] N. Carlini and D. A. Wagner, "Towards Evaluating the Robustness of Neural Networks," in *Proc. 38th IEEE Symposium on Security and Privacy, IEEE S&P 2017*, 2017, pp. 39–57.

[12] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. Montreal Quebec Canada: ACM, May 2002, pp. 380–388. [Online]. Available: https://dl.acm.org/doi/10.1145/509907.509965

[13] L. D. Comba and J. Stolfi, "Affine arithmetic and its applications to computer graphics," 1990. [Online]. Available: https://api.semanticscholar.org/CorpusID:8848808

[14] P. Cousot, *Principles of Abstract Interpretation*. MIT Press, 2021.

[15] P. Cousot and R. Cousot, "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *Proc. 4th ACM Symposium on Principles of Programming Languages, POPL 1977*, 1977, pp. 238–252.

[16] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[17] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma, "Adversarial classification," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 99–108. [Online]. Available: https://doi.org/10.1145/1014052.1014066

[18] B. Daniels, *Rote-LCS learning classifier system for classification and prediction*. Missouri University of Science and Technology, 2015.

[19] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. Zemel, "Fairness Through Awareness," in *Proc. 3rd Innovations in Theoretical Computer Science Conference*, 2012, pp. 214–226.

[20] N. Fassina, F. Ranzato, and M. Zanella, "Robustness certification of k-nearest neighbors," in *2023 IEEE International Conference on Data Mining (ICDM)*, 2023, pp. 110–119.

[21] E. Fix and J. L. Hodges, "Discriminatory analysis - nonparametric discrimination: Small sample performance," 1952. [Online]. Available: https://api.semanticscholar.org/CorpusID:115703342

[22] ——, "Discriminatory analysis. nonparametric discrimination: Consistency properties," *International Statistical Review/Revue Internationale de Statistique*, vol. 57, no. 3, pp. 238–247, 1989.

[23] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. T. Vechev, "AI2: Safety and robustness certification of neura networks with abstract interpretation," in *Proc. 2018 IEEE Symposium on Security and Privacy, IEEE S&P 2018*, 2018, pp. 3–18. [Online]. Available: https://doi.org/10.1109/SP.2018.00058

[24] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015. [Online]. Available: https://arxiv.org/abs/1412.6572

[25] S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelovic, T. Mann, and P. Kohli, "On the effectiveness of interval bound propagation for training verifiably robust models," 2019. [Online]. Available: https://arxiv.org/abs/1810.12715

[26] U. Khandelwal, A. Fan, D. Jurafsky, L. Zettlemoyer, and M. Lewis, "Nearest neighbor machine translation," 2021. [Online]. Available: https://arxiv.org/abs/2010.00710

[27] B. R. Kowalski and C. F. Bender, "K-nearest neighbor classification rule (pattern recognition) applied to nuclear magnetic resonance spectral interpretation," *Analytical Chemistry*, vol. 44, no. 8, pp. 1405–1411, 1972. [Online]. Available: https://doi.org/10.1021/ac60316a008

[28] O. Kramer, "K-nearest neighbors," in *Dimensionality Reduction with Unsupervised Nearest Neighbors, Intelligent Systems Reference Library, vol 51.* Berlin: Springer, 2013, pp. 13–23.

[29] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Artificial intelligence safety and security.* Chapman and Hall/CRC, 2018, pp. 99–112.

[30] S. V. Mahadevkar, B. Khemani, S. A. Patil, K. V. Kotecha, D. R. Vora, A. Abraham, and L. A. K. Gabralla, "A review on machine learning styles in computer vision—techniques and future directions," *IEEE Access*, vol. 10, pp. 107 293–107 329, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:252558821

[31] J. E. Martínez-Legaz, V. Roshchina, and M. Todorov, "On the structure of higher order voronoi cells," 2019. [Online]. Available: https://arxiv.org/abs/1811.10257

[32] J. McCarthy, "What is artificial intelligence," http://www-formal.stanford.edu/jmc/whatisai.html, 2004.

[33] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, "A survey on bias and fairness in machine learning," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–35, 2021.

[34] M. Mirman, T. Gehr, and M. T. Vechev, "Differentiable abstract interpretation for provably robust neural networks," in *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, 2018, pp. 3575–3583. [Online]. Available: http://proceedings.mlr.press/v80/mirman18b.html

[35] C. Müller, F. Serre, G. Singh, M. Püschel, and M. T. Vechev, "Scaling polyhedral neural network verification on gpus," in *Proceedings of Machine Learning and Systems 2021, MLSys 2021*, 2021. [Online]. Available: https://proceedings.mlsys.org/paper/2021/hash/ca46c1b9512a7a8315fa3c5a946e8265-Abstract.html

[36] F. S. M. Nadisah Zakaria, Ainin Sulaiman and A. Feizollah, "Machine learning in the financial industry: A bibliometric approach to evidencing applications," *Cogent Social Sciences*, vol. 9, no. 2, p. 2276609, 2023.

[37] M. Nakada, H. Wang, and D. Terzopoulos, "Acfr: Active face recognition using convolutional neural networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 35–40, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:33743430

[38] A. Okabe, B. Boots, and K. Sugihara, *Spatial tessellations: concepts and applications of Voronoi diagrams*. USA: John Wiley & Sons, Inc., 1992.

[39] F. Ranzato, C. Urban, and M. Zanella, "Fairness-aware training of decision trees by abstract interpretation," in *Proceedings of the 30th ACM International Conference on Information and Knowledge Management, CIKM 2021*. ACM, 2021, pp. 1508–1517.

[40] F. Ranzato and M. Zanella, "Robustness Verification of Support Vector Machines," in *Proc. 26th International Static Analysis Symposium, SAS 2019*, ser. LNCS, vol. 11822, 2019, pp. 271–295.

[41] ——, "Abstract interpretation of decision tree ensemble classifiers," in *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, 2020, pp. 5478–5486. [Online]. Available: https://aaai.org/ojs/index.php/AAAI/article/view/5998

[42] ——, "Genetic adversarial training of decision trees," in *Proceedings of the 2021 Genetic and Evolutionary Computation Conference, GECCO 2021*. ACM, 2021, pp. 358–367.

[43] S. Raschka, "Stat 479: Machine learning lecture notes," https://sebastianraschka.com/pdf/lecture-notes/stat479fs18/02_knn_notes.pdf, 2018.

[44] A. Ruoss, M. Balunovic, M. Fischer, and M. Vechev, "Learning certified individually fair representations," in *Proc. 34th Annual Conference on Advances in Neural Information Processing Systems, NeurIPS 2020*, 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/55d491cf951b1b920900684d71419282-Abstract.html

[45] A. Schrijver, *Theory of linear and integer programming*. USA: John Wiley & Sons, Inc., 1986.

[46] R. E. Shaffer, S. L. Rose-Pehrsson, and R. A. McGill, "A comparison study of chemical sensor array pattern recognition algorithms," *Analytica Chimica Acta*, vol. 384, pp. 305–317, 1999.

[47] A. Shakeel, GitHub Page: https://github.com/shakehd/KNN_Certifier.

[48] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. T. Vechev, "Fast and effective robustness certification," in *Proc. Annual Conf. on Neural Information Processing Systems, NeurIPS 2018*, 2018, pp. 10 825–10 836. [Online]. Available: http://papers.nips.cc/paper/8278-fast-and-effective-robustness-certification

[49] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," *Proc. ACM Program. Lang.*, vol. 3, no. POPL 2019, pp. 41:1–41:30, Jan. 2019. [Online]. Available: http://doi.acm.org/10.1145/3290354

[50] G. Singh, T. Gehr, M. Püschel, and M. T. Vechev, "Boosting robustness certification of neural networks," in *Proceedings of the 7th International Conference on Learning Representations, ICLR 2019*, 2019. [Online]. Available: https://openreview.net/forum?id=HJgeEh09KQ

[51] C. Sitawarin, E. M. Kornaropoulos, D. Song, and D. A. Wagner, "Adversarial examples for k-nearest neighbor classifiers based on higher-order Voronoi diagrams," in *Proc. Annual Conference on Neural Information Processing Systems, NeurIPS 2021*, 2021, pp. 15 486–15 497. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/82ca5dd156cc926b2992f73c2896f761-Abstract.html

[52] C. Sitawarin and D. Wagner, "Defending against adversarial examples with k-nearest neighbor," 2019. [Online]. Available: https://arxiv.org/abs/1906.09525

[53] ——, "On the robustness of deep k-nearest neighbors," 2019. [Online]. Available: https://arxiv.org/abs/1903.08333

[54] ——, "Minimum-norm adversarial examples on KNN and KNN based models," in *2020 IEEE Security and Privacy Workshops, SP Workshops, 2020*. IEEE, 2020, pp. 34–40.

[55] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2014. [Online]. Available: https://arxiv.org/abs/1312.6199

[56] S. Tarek, R. Abd Elwahab, and M. Shoman, "Gene expression based cancer classification," *Egyptian Informatics Journal*, vol. 18, no. 3, pp. 151–159, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1110866516300809

[57] C. Urban and A. Miné, "A Review of Formal Methods applied to Machine Learning," *CoRR*, vol. abs/2104.02466, 2021. [Online]. Available: https://arxiv.org/abs/2104.02466

[58] L. Wang, X. Liu, J. Yi, Z.-H. Zhou, and C.-J. Hsieh, "Evaluating the robustness of nearest neighbor classifiers: A primal-dual perspective," 2019. [Online]. Available: https://arxiv.org/abs/1906.03972

[59] Y. Wang, S. Jha, and K. Chaudhuri, "Analyzing the robustness of nearest neighbors to adversarial examples," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 2018, pp. 5120–5129. [Online]. Available: http://proceedings.mlr.press/v80/wang18c.html

[60] E. Wong and Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 5286–5295. [Online]. Available: https://proceedings.mlr.press/v80/wong18a.html

[61] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. F. M. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. S. Steinbach, D. J. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, pp. 1–37, 2007. [Online]. Available: https://api.semanticscholar.org/CorpusID:2367747

[62] Y. Yang, C. Rashtchian, Y. Wang, and K. Chaudhuri, "Robustness for non-parametric classification: A generic attack and defense," in *Proc. 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020*, ser. Proceedings of Machine Learning Research, vol. 108. PMLR, 2020, pp. 941–951. [Online]. Available: http://proceedings.mlr.press/v108/yang20b.html

[63] F. Ye, M. Dong, W. Luo, X. Chen, and W. Min, "A new re-ranking method based on convolutional neural network and two image-to-class distances for remote sensing image retrieval," *IEEE Access*, vol. 7, pp. 141 498–141 507, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:204085020

[64] H. Zhang, H. Chen, C. Xiao, S. Gowal, R. Stanforth, B. Li, D. Boning, and C.-J. Hsieh, "Towards stable and efficient training of verifiably robust neural networks," 2019. [Online]. Available: https://arxiv.org/abs/1906.06316

[65] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/file/d04863f100d59b3eb688a11f95b0ae60-Paper.pdf