EXPLORING MACHINE LEARNING STRATEGIES FOR THE IDENTIFICATION, QUANTIFICATION, AND MITIGATION OF SMALL DISJUNCTS IN IMBALANCED DATA

Hugo Valdrez
Master in Computer Science
Department of Computer Science
Faculty of Sciences of the University of Porto
2025

EXPLORING MACHINE LEARNING STRATEGIES FOR THE IDENTIFICATION, QUANTIFICATION, AND MITIGATION OF SMALL DISJUNCTS IN IMBALANCED DATA

Hugo Valdrez

Dissertation carried out as part of the
Master in Computer Science

Department of Computer Science

2025

Supervisors

Miriam Seoane Santos, Assistant Professor, Department of Computer Science, Faculty of Sciences, University of Porto

Ricardo Cardoso Pereira, Assistant Professor, Department of Informatics Engineering, Faculty of Sciences and Technology, University of Coimbra

“ If I take one more step, it’ll be the farthest away from home I’ve ever been. ”

Samwise Gamgee in *The Fellowship of the Ring*

Acknowledgements

These acknowledgements will, much like the theme of this work, come in small disjuncts. There is no way to thank everyone who has crossed my path during these last years and I can only mention a small fraction of them here. Within this minority, most do not even know each other, as they are scattered across different spaces of my life, yet all are equally important and all deserve my deepest gratitude.

First and foremost, I would like to thank my supervisor, Prof. Miriam Seoane Santos, for her unwavering help, dedication and motivation throughout the development of this work. Even when I was not the perfect student - when mistakes were made, when results were lacking and when my own motivation faltered - she went the extra mile to keep the boat afloat. I could not thank her without also thanking Prof. Ricardo Pereira, who joined us willingly and supported us in every possible way. He was always there, and without their joint guidance, I doubt this work would ever have reached completion.

To my work colleagues, both past and present, thank you for standing by my side and making sure, again and again, that time and workload would never be a barrier. A special thanks to Carolina for her constant support.

To my family, especially my parents and grandparents: to my grandmother, for her unconditional care and encouragement; to my father, for being the greatest example of relentless work and energy; to my mother, the toughest person I know, for her limitless love; and to my sister - I could not be prouder of you.

To my friends: the countless hours spent playing board games and enjoying live music are what truly matter in the end. A special thanks to Duarte - without him, this journey would not have began.

Lastly, to my eternal partner, Inês, the one I look to the most - my infinite gratitude.

Resumo

A aprendizagem automática (*machine learning*) tem sido cada vez mais integrada em contextos do mundo real e o seu sucesso depende fortemente da disponibilidade de dados de elevada qualidade. Entre os desafios que comprometem esse sucesso, um dos mais prejudiciais é o desequilíbro de classes (*class imbalance*). Os algoritmos de aprendizagem automática são, em geral, concebidos sob a premissa de distribuições de classes equilibradas e, quando expostos a um desequilíbrio das mesmas, a sua precisão e fiabilidade degradam-se significativamente.

No âmbito deste problema este trabalho foca-se numa manifestação específica de desequilíbrio intra-classes (*within-class imbalance*) conhecida como o problema de "pequenos disjuntos" (*small disjuncts*), estes surgem quando alguns conceitos são representados por pequenos grupos sub-representados no espaço de dados. Embora este seja um problema bem documentado na literatura há várias décadas, as estratégias existentes para lidar com os disjuncts continuam a ser quase exclusivamente centradas nos modelos (*model-centric*), isto é, dependentes de algoritmos específicos, como árvores de decisão ou sistemas baseados em regras. Como consequência, ainda não existe uma metodologia geral, agnóstica aos modelos (*model-agnostic*), para identificar e mitigar pequenos disjuntos em conjuntos de dados reais.

Neste trabalho, colmatamos essa lacuna propondo o IterDBSCAN, uma abordagem baseada em análise de agrupamentos (*clustering*) concebida para identificar sistematicamente pequenos disjuntos. Esta estratégia é inicialmente validada com diversos conjuntos de dados sintéticos e, posteriormente, validada através de um benchmark a conjuntos de dados imbalanced do mundo real. Foi ainda desenvolvido um estudo empírico com o objetivo de avaliar a robustez de técnicas conhecidas de sobreamostragem (*oversampling*) relativamente ao problema dos pequenos disjuntos, e desenvolvido um novo algoritmo, IterDBSCAN-SMOTE, que demonstrou melhorar o desempenho de classificadores.

Palavras-chave: Aprendizagem Automática, Desequilíbrio de Classes, Dificuldades de Dados, Complexidade de Dados, Pequenos Disjuntos,Qualidade dos Dados

Abstract

As machine learning becomes increasingly integrated into real-world applications, its success depends heavily on the availability of high-quality data. Among the challenges that undermine its success, one of the most detrimental is class imbalance. Standard machine learning algorithms are typically designed under the assumption of balanced class distributions, and when exposed to uneven data, their accuracy and reliability degrade significantly.

Within the broader imbalance problem, this work focuses on a specific manifestation of within-class imbalance known as *small disjuncts*, which arises when concepts are represented by small, underrepresented clusters within the input space. Although the issue has been acknowledged in the literature for decades, existing strategies for handling small disjuncts remain almost exclusively model-centric, tied to specific algorithms such as decision trees or rule-based systems. Beyond these learning paradigms, there is still no general, model-agnostic methodology for identifying and mitigating small disjuncts, especially in real-world datasets.

In this work, we address this gap by proposing IterDBSCAN, a clustering-based approach designed to systematically identify small disjuncts. This strategy is first validated across several established synthetic datasets and then evaluated across a wider benchmark of real-world imbalanced datasets. We further produce a comprehensive empirical framework to assess the robustness of well-known oversampling techniques to the problem of small disjuncts, and develop a new algorithm, IterDBSCAN-SMOTE, which has shown to improve the performance of standard classifiers in the recognition of these rare pockets of data.

Keywords: Machine Learning, Class Imbalance, Data Difficulty Factors, Data Complexity, Small Disjuncts, Data Quality

Table of Contents

List of Figures

1. Introduction

As Machine Learning (ML) becomes increasingly prominent in our daily lives, one of the cornerstones of its continued success is the availability of high-quality data to support these models [1, 2]. Traditionally, research has focused on improving ML systems through a *model-centric* approach [3], emphasizing the development of new model types and architectures, which has led to considerable advancements in their maturity and capabilities.

Yet, model performance is not determined solely by architecture or hyperparameter choices alone. The provision, selection and curation of appropriate data also play a critical role in ensuring effectiveness. Furthermore, as model-centric research has begun to plateau - that is, as models grow in complexity without yielding proportional gains in performance - a paradigm shift has emerged toward a data-centric perspective. Accordingly, Data-Centric Artificial Intelligence (DCAI) emphasizes that the systematic design, engineering and refinement of data are essential for building effective and efficient ML-based systems [3].

This shift towards DCAI is particularly relevant in real-world contexts where data is often imperfect, irregular or misaligned with researcher expectations [4, 5]. Furthermore, within a data-centric framework, the focus extends beyond improving raw performance: it also includes enhancing fairness, robustness and trustworthiness [6].

1.1. Context and Motivation

Among the imperfections frequently encountered in real-world contexts, one of the most detrimental is imbalanced data [7]. This issue is particularly harmful to most standard ML algorithms because, traditionally, such algorithms are designed to learn from balanced distributions [5, 7, 8]. When exposed to uneven data distributions, however, their accuracy and reliability tend to degrade significantly [7]. This challenge, commonly referred to as the *imbalance problem*, remains an active research topic despite ongoing advances in data-centric methodologies [3, 8]. In essence, research in imbalanced learning aims to develop intelligent systems capable of effectively mitigating the biases introduced by skewed data distributions.

Within this broader problem, our work is particularly concerned with a special case of imbalance known as *small disjuncts*. Small disjuncts occur when concepts are represented by small clusters within the data space in a phenomenon known as *within-class imbalance* [9]. While the fragmentation of a class into multiple sub-concepts may be oftentimes surpassed by ML algorithms with non-linear decision boundaries, difficulties arise when these small clusters emerge under conditions of class imbalance. In such cases, it becomes increasingly challenging to discern whether these clusters genuinely represent meaningful substructures of the minority class or are merely noise [10].

This lack of homogeneity poses a specific challenge for most ML algorithms, especially those based on divide-and-conquer strategies [10, 11]. The reason is intuitive: such algorithms partition the feature space into distinct regions - squares in two dimensions, cubes in three, and hypercubes in higher dimensions. Once partitioned, the algorithm treats each region as an independent concept, thereby losing the relational structure that may exist across these smaller clusters.

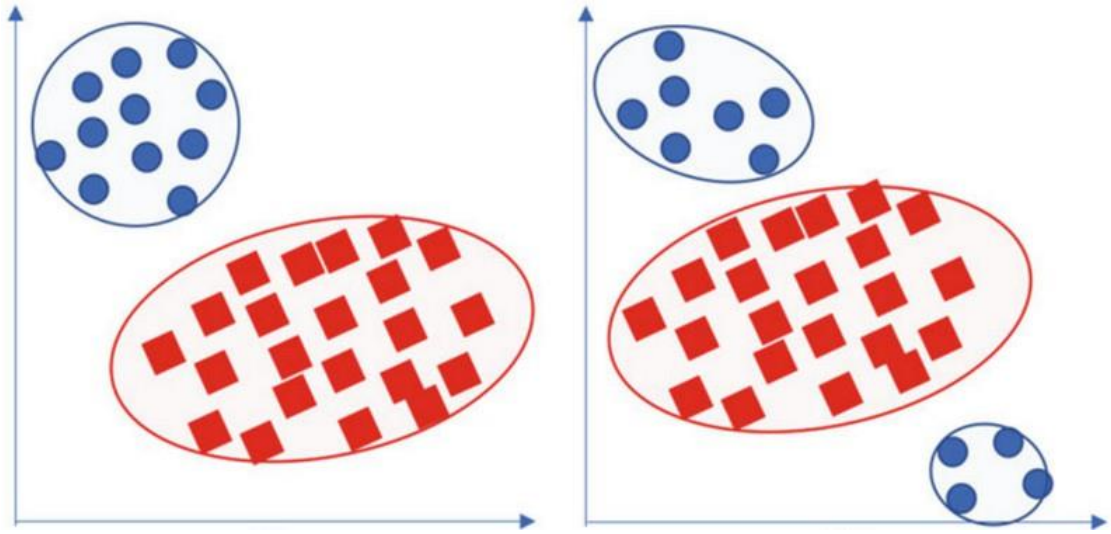


Figure 1.1: Dataset without vs. with small disjuncts. [11].

Although the problem of small disjuncts has long been recognized in the literature, existing strategies for their identification and handling are almost exclusively limited to model-centric approaches, such as decision trees or rule-based systems, where small disjuncts can be observed directly as rules or leaves with very low coverage. Outside of these models, however, there is still no general methodology for identifying or mitigating small disjuncts in real-world datasets. This gap highlights the need for data-centric strategies that can be applied independently of the learning algorithm, enabling the detection and

treatment of small disjuncts across diverse ML paradigms. Developing such strategies - for both identification and mitigation - constitutes the central goal of this work.

1.2. Research Goals

Given the prevalence of imbalanced data and the relatively underexplored nature of the small disjuncts problem, this work pursues two big main objectives:

1. the proposal of a strategy for the identification and quantification of small disjuncts, and
2. its integration with a complementary strategy for their mitigation.

To accomplish these goals we developed a clustering algorithm, intended to find small underrepresented clusters in data (small disjuncts), initially designed and evaluated using synthetic datasets. Then, this approach was incorporated into an oversampling method to mitigate class imbalance. Subsequently, the integrated approach was assessed on real-world datasets and benchmarked against existing state-of-the-art methodologies.

1.3. Research Contributions

With this work, we consider that three main contributions have been achieved:

1. **A novel clustering algorithm for sub-concept identification.** We propose an algorithm capable of identifying sub-concepts by extending a well-known algorithm, DBSCAN, and mitigating its previous problems with varying densities and linkage points (which we define as “bridging points“ or “bridges”). This process not only enables the recognition of sub-concepts within a class but also provides clarity regarding meaningful concepts, with the bridge-related points removed.
2. **An oversampling method informed by sub-concept identification.** Building on the first contribution, we introduce an oversampling strategy that makes use of the detected sub-concepts. By oversampling in a manner that preserves the internal structure and representation of each sub-concept our method provides a more faithful representation of the overall class distribution.
3. **Disjunct-specific performance metrics.** We propose a set of novel evaluation measures designed to capture classifier behavior on small disjuncts, inspired by

Weiss' Error Concentration metric. These metrics quantify performance within structurally rare and error-prone regions of the minority class, ensuring that global improvements do not obscure local deficiencies.

4. **An empirical study benchmarking against state-of-the-art approaches.** Finally, we conduct a comprehensive empirical study to evaluate the proposed framework, comparing its performance with existing state-of-the-art methodologies.

1.4. Work Plan

The research was structured into 4 distinct phases, each building upon the previous one to ensure a systematic progression toward the final objectives. While some adjustments were made during the process, the overall plan followed the trajectory outlined below.

Phase 1: Literature Review and Theoretical Background

The initial stage of the work focused on establishing a solid theoretical foundation. The literature review was conducted in two steps:

1. **General approaches to imbalanced learning** - This included the study of data-level methods (e.g., oversampling and undersampling techniques), algorithm-level modifications and cost-sensitive learning strategies. These approaches provided a broad perspective on how the research community has traditionally addressed imbalance.

2. **Specific approaches to small disjuncts** - After building this general background, the focus shifted to methods explicitly addressing small disjuncts. The literature in this area is relatively scarce but the related strategies identified can be divided into two categories:

    - **Identification methods**, based on complexity metrics and related measures, which provide insights into where standard ML algorithms may fail or succeed.
    - **Resampling methods**, such as some SMOTE-based approaches, which aim to mitigate the issue through data generation or manipulation.

Phase 2: Exploration of Identification and Resampling Strategies

The second phase involved a deeper examination of both identification and resampling approaches.

- On the **identification side**, complexity measures were studied and tested with the goal of assessing their suitability to guide preprocessing or cost-based strategies. A set of relevant metrics was established, along with an analysis of their advantages and weaknesses. For instance, it was observed that certain metrics (e.g., F1) lose reliability when classes are highly separable [12].
- On the **resampling side**, we have reviewed both the top-performing oversampling approaches in imbalanced domains (such as SMOTE and SMOTE Tomek Links [13]) and approaches that incorporated cluster-based methods, ideally suited to handle class decomposition and possibly small disjuncts (such as MWMOTE and CBO [14, 15]).

Phase 3: Algorithm Development

Building on the acquired knowledge, the third phase was dedicated to the design and development of a method capable of identifying small disjuncts more effectively. This stage constituted the core of the research and spanned most of the second semester. The algorithm was developed iteratively through cycles of experimentation, error analysis and refinement. Initial validation was conducted primarily on synthetic datasets as these allowed controlled testing conditions.

Phase 4: Pipeline Design and Implementation

Once the algorithm reached a stable version, the focus shifted to the construction of a comprehensive experimental pipeline to assess the identification algorithm in real-world domains and test its incorporation into an oversampling algorithm capable of handling class imbalance and alleviating small disjuncts. This pipeline defined:

- The classifiers and datasets to be employed;
- The state-of-the-art methods against which comparisons would be made;
- The evaluation metrics to be considered;
- The specific analyses required to ensure a thorough assessment.

This pipeline provided the necessary structure for systematic evaluation and serves as the foundation for the results presented in later chapters.

1.5. Document Structure

This work is organized as follows. Chapter 2 reviews the state of the art, focusing on related work in the areas of imbalanced learning, data complexity metrics and specialized methods for identifying and mitigating small disjuncts. We also highlight open challenges and research gaps that motivate the development of new approaches.

Chapter 3 describes the methodology followed in this work. We provide an overview of the experimental design and present the proposed IterDBSCAN algorithm, including its motivation and design choices.

Chapter 4 discusses the proposed approach in depth, presenting each step of the Iter-DBSCAN algorithm. We illustrate the results obtained on synthetic datasets and analyze the impact of the proposed method.

Chapter 5 presents the benchmark over real-world datasets, comparing IterDBSCAN-SMOTE, our proposed oversampling approach, with other state-of-the-art methods. We discuss the obtained results and highlight the strengths and limitations of the proposed approach.

Finally, Chapter 6 concludes this work by summarizing its main contributions, discussing its limitations and outlining possible directions for future work.

2. State of the Art

In this chapter we will establish the background knowledge and related work on the topic of small disjuncts. Accordingly, we start by introducing the imbalanced data problem and the main approaches to address it (Section 2.1). We then examine existing data complexity metrics, with particular emphasis on their relevance to characterize classification difficulty (Section 2.2). Finally, we focus specifically on the small disjuncts problem itself, surveying methods and approaches developed to mitigate its effects and highlighting current open challenges that remain to be addressed (Section 2.3).

2.1. Imbalanced Data Problem

2.1.1 The Class Imbalance Problem

Real-world datasets frequently display significant disparities in class representation, a phenomenon known as the *class imbalance problem*. This situation arises when one class, typically referred to as the minority class (often the positive class), is heavily under-represented in comparison to the majority class (often the negative class).

Since most traditional ML algorithms are conceptually designed under the assumption of balanced data distributions [7], they tend to favor the majority class when trained on imbalanced datasets [8]. As a result, minority class instances are often overlooked, leading to biased models and reduced predictive performance. Beyond the purely algorithmic challenges, this imbalance carries practical significance: from a data mining perspective, minority class instances frequently encode valuable knowledge [8], and in many cases, detecting rare events can itself be framed as a prediction task [16].

The degree of imbalance is typically quantified through the *Imbalance Ratio (IR)*, defined as:

$$IR = \frac{N_{\text{maj}}}{N_{\text{min}}}, \tag{2.1}$$

where N_{maj} denotes the number of majority-class samples and N_{min} denotes the number of minority-class samples. In imbalanced domains, $IR \geq 1$, with $IR = 1$ representing a perfectly balanced dataset and higher values indicating increasingly severe imbalance.

Importantly, class imbalance is domain-agnostic, occurring across a wide variety of fields [4]. For instance, in healthcare, breast cancer diagnosis provides a salient example: while the disease can affect both women and men, incidence rates among men are considerably

lower. Without appropriate handling, a machine learning model trained on such data may disregard male cases entirely, resulting in biased and potentially harmful predictions. Similarly, in finance, fraudulent transactions typically constitute only a minute fraction of all daily banking operations. Here, the imbalance ratio is often extreme yet accurate detection is critical, as both financial security and institutional trust are at stake.

These examples illustrate that class imbalance is not a peripheral issue but rather a central challenge in many high-stakes domains. Whether the consequences involve human health or financial integrity, the ability to design solutions that address imbalance effectively is of fundamental importance.

2.1.2 Strategies to Handle Imbalanced Data

Class imbalance is not a new problem; it has been recognized for decades and numerous strategies have been developed to mitigate its effects. These strategies are commonly divided into four main groups [16]: data-level methods/preprocessing techniques, cost-sensitive learning, algorithmic-level modifications, and ensemble methods. Among these, data-level methods will be the primary focus of our work since they are model-agnostic and can be applied to virtually any domain and application.

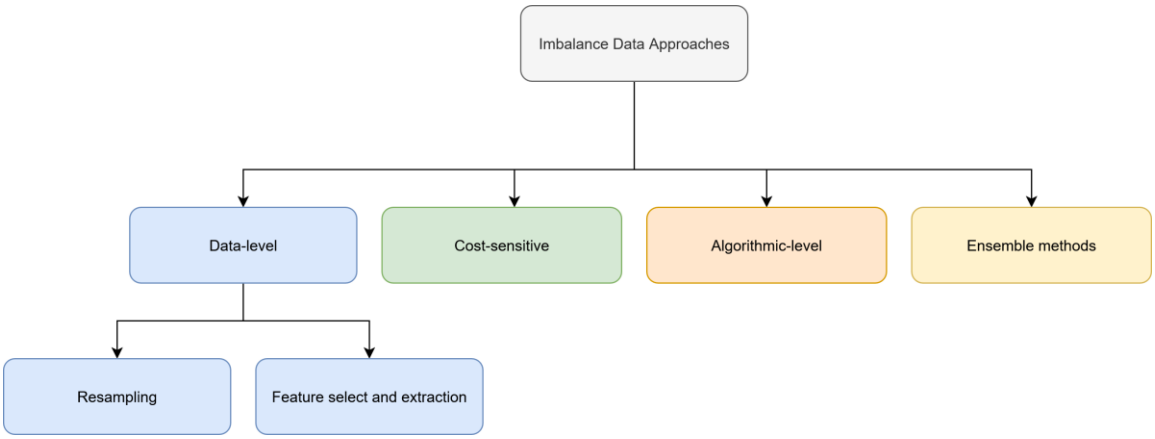


Figure 2.1: Approaches to handle imbalanced data.

Data-level methods. These approaches can be understood as preprocessing techniques applied before model training. Their goal is to rebalance the sample space of an imbalanced dataset in order to alleviate the skewed effect of class distribution on the learning process. An important advantage is that they are *classifier-independent* [10], meaning they can be applied regardless of the chosen model, though benchmarking has primarily been performed with classifiers such as Support Vector Machines (SVMs), k-Nearest

Neighbours (k-NNs), Logistic Regression, Decision Trees, and Naïve Bayes [7]. Data-level approaches are also popular due to their relative simplicity, making them accessible even to practitioners without deep expertise in machine learning [16].

Among data-level approaches, resampling methods represent the most widely adopted strategy and they can be broadly divided into three categories:

- *Oversampling.* This technique balances the dataset by increasing the number of minority class samples, often through synthetic generation. The most common approach is the Synthetic Minority Oversampling Technique (SMOTE), which creates new samples by interpolating between existing minority instances and their nearest neighbors. Although highly effective, SMOTE may generate duplicate samples, ignore local distributions and introduce noisy or false instances [8], which is particularly problematic in high-dimensional spaces or with noise-sensitive algorithms. To overcome these handicaps, other variants have been proposed throughout the years [13, 17], which we will explore later on Section 2.3.2.
- *Undersampling.* This method reduces imbalance by discarding samples from the majority class. The simplest approach is Random Undersampling (RUS), which randomly removes majority instances. While conceptually straightforward, undersampling risks discarding important concepts of the majority class which might be relevant for an appropriate definition of decision boundaries.
- *Hybrid methods.* These approaches combine oversampling and undersampling strategies. Although less frequently used (around 21% of cases [16]), hybrid methods have shown promising performance across some benchmarking studies [7].

Another data-level approach is *feature selection*. In imbalanced datasets, irrelevant features increase the likelihood that minority samples are treated as noise. Feature selection seeks to identify a subset of informative features that reduces dimensionality and enhances classifier performance [16].

Cost-sensitive learning. This family of methods resembles an algorithmic-level approach. The core idea is to assign higher misclassification costs to minority class instances compared to majority class instances [10, 16]. Since in imbalanced contexts it is often more important to correctly identify minority (positive) cases, the penalty for misclassifying them is greater than for majority (negative) cases.

Algorithm-level modifications. Beyond cost-sensitive adjustments, algorithm-level strategies directly modify the learning process to better handle imbalance. These methods adapt the internal mechanics of classifiers so that they become inherently more robust to skewed distributions. Examples include modifying decision tree splitting criteria to favor minority samples or adapting SVM margin constraints to weight minority samples more heavily [18].

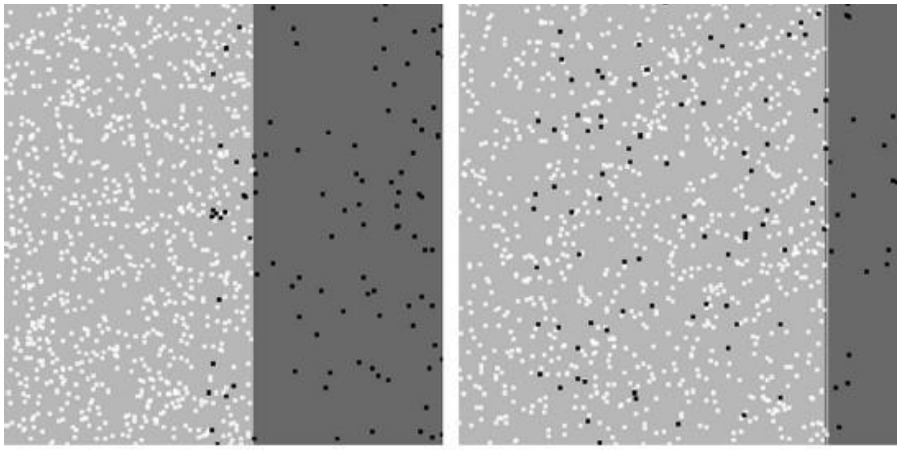
Ensemble methods. Finally, ensemble methods combine multiple classifiers to build a stronger composite model. In the context of class imbalance, ensemble approaches can also be combined with data-level resampling strategies or cost-sensitive techniques.

2.1.3 Data Difficulty Factors

Class imbalance by itself does not necessarily imply poor learning performance; rather, it is the interaction of imbalance with other data intrinsic characteristics that typically degrades the effectiveness of learning algorithms [10, 11]. A high IR can be manageable when the remainder of the problem is "well-behaved" (for example, when classes are separable and the dataset is sufficiently representative) [11]. However, when imbalance co-occurs with other adverse data characteristics, the combined effect can be highly detrimental to model performance. Recognizing and diagnosing difficulty factors such as *class overlap*, *lack of density*, *dataset shift*, and *small disjuncts* is therefore a necessary step before selecting or designing imbalance-handling strategies.

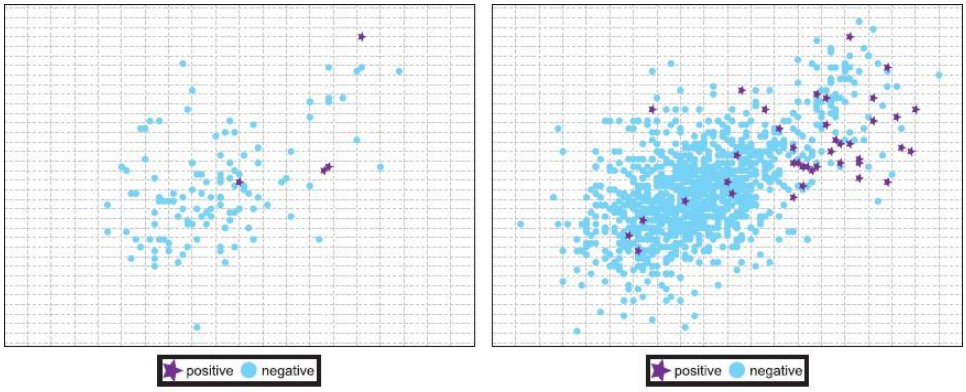
Class overlap. Class overlap occurs when instances of different classes occupy the same regions of the feature space, producing regions of ambiguity where labels are intermixed (Figure 2.2) [19]. Overlap reduces class separability and produces ambiguous or unstable decision boundaries; this increases classification error and is often identified as more detrimental than imbalance alone [10].

Lack of density. When the minority class has very low density, ML algorithms lack sufficient evidence to learn reliable decision boundaries, resulting in overfitting and poor generalization ability. For example, in the *yeast4* benchmark dataset (Figure 2.3) it has been shown that if one retains only 10% of the original instances, the existing concepts become highly deteriorated [10] .



(a) 20% overlap (b) 80% overlap

Figure 2.2: Example of class overlap in imbalanced datasets: boundaries detected by C4.5 [10].



(a) 10 % of training instances (b) 100 % of training instances

Figure 2.3: Example of lack of density or small sample size on the *yeast4* dataset [10].

Dataset shift. Two datasets with identical IRs cannot be assumed to present the same learning complexity: as shown in Figures 2.4 and 2.5, the way the training data represents the structure of minority instances (their spatial distribution, local density, overlap with the majority class, and noise level) is crucial. In particular, dataset shift (covariate and/or prior probability changes between training and testing/deployment) disproportionately affects minority classes because their already-small support can change or disappear in new environments, producing sharp performance degradation.

Small disjuncts and sub-concepts. Finally, a lack of homogeneity in the input space can produce *sub-concepts* that cover only a few instances and these *sub-concepts* frequently give rise to *small disjuncts* (Figure 2.6). As traditional classifiers are biased towards larger, well-represented disjuncts, smaller disjuncts can be neglected and not effectively learned. Thus, they can concentrate most of a classifier's errors and constitute

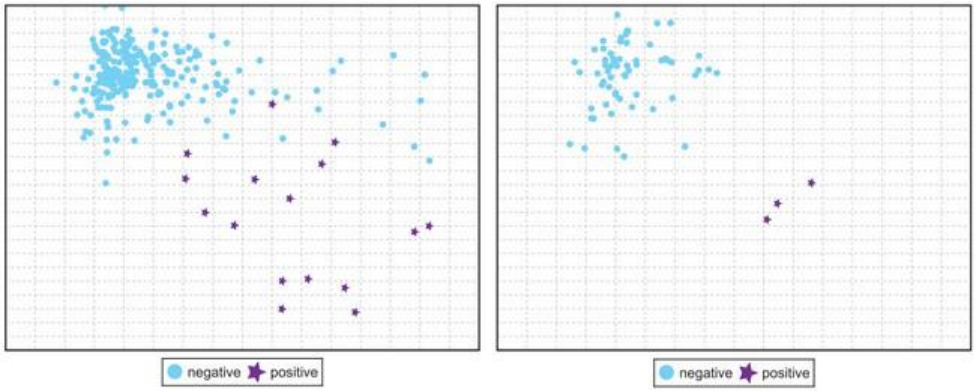


Figure 2.4: Example of representative training and test sets [10].

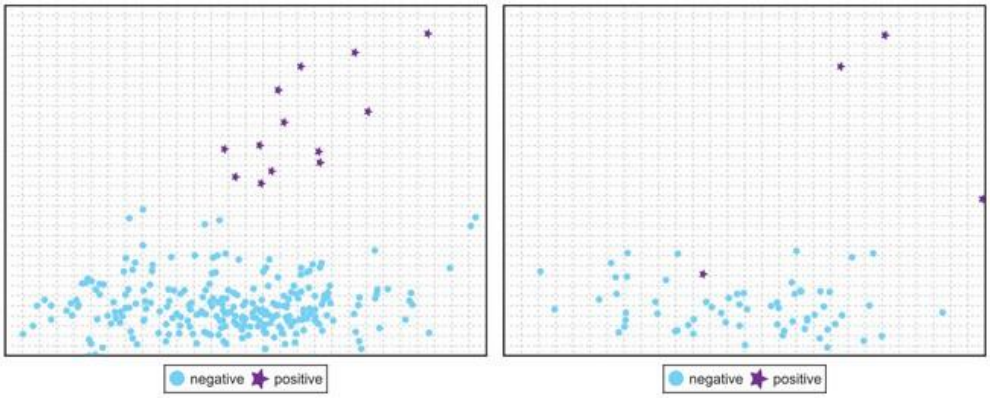
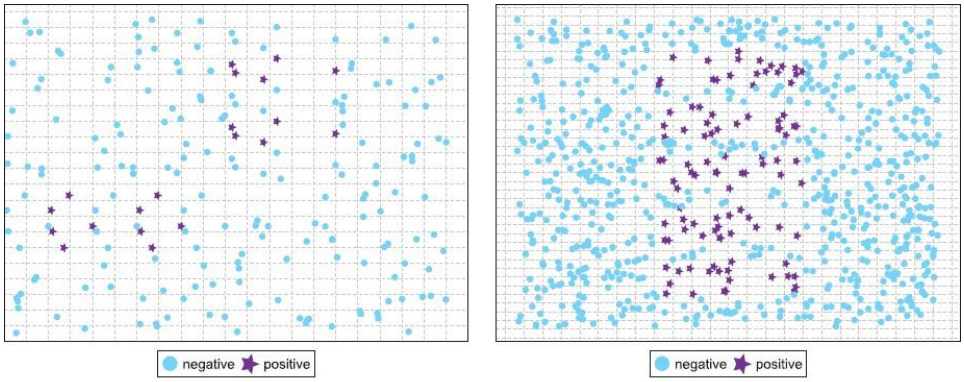


Figure 2.5: Example of non-representative training and test sets [10].

a major source of poor generalization in imbalanced domains. Most importantly, when small disjuncts reflect important, although rare, sub-concepts in data, neglecting them can prove harmful for high-stakes domains (e.g., rare event detection across verticals such as healthcare and fraud).

Another possible origin of small disjuncts is the presence of clustered noisy instances: if noisy examples occur in the same region of the input space, they may be mistakenly interpreted as a genuine sub-concept, producing an apparent small disjunct that is in fact artefactual. Because the representative support of such sub-concepts is diminutive, this identification difficulty is a core reason why distinguishing true sub-concepts from noise is challenging in practice. We will further discuss the problem of small disjuncts in Section 2.3.

As previously noted, many learning algorithms implicitly rely on assumptions (e.g., sufficient representative samples, homogeneity within class regions), and when any of these assumptions is violated, the normal learning process is hindered and performance can drop substantially.



(a) Artificial dataset: small disjuncts for the minority class

(b) Subclus dataset: small disjuncts for both classes

Figure 2.6: Example of small disjuncts in imbalanced data [10].

Moreover, because these difficulty factors rarely appear in isolation, their interactions (for example, overlap combined with scarcity, or noise combined with small disjuncts) frequently explains why methods that perform well on moderately imbalanced but otherwise "easy" benchmarks fail on complex real-world problems. Understanding data difficulty factors and diagnosing which assumption(s) fail(s) in a given dataset helps explain empirical failures and guides the selection of suitable solutions (resampling, robust learning, feature selection, etc.).

To properly address this problem, current research relies on data complexity metrics, which we explain in what follows.

2.2. Data Complexity Metrics

Characteristics extracted from the data used in classification problems have proven to be effective performance predictors in a number of *meta-analyses*. In this context, a meta-analysis refers to a systematic study that aggregates performance results from multiple experiments (usually classification tasks), and links them to the characteristics of the used datasets, in order to identify consistent patterns and general principles. Among the characteristics studied, measures of *classification complexity* are particularly useful since they attempt to quantify the difficulty of separating data points into their expected classes.

According to Ho and Basu (2002) [20], the complexity of a classification problem can be attributed to three main factors:

1. **Class ambiguity:** scenarios in which classes cannot be distinguished using the available features, regardless of the classification algorithm employed;

2. **Sparsity and dimensionality:** incomplete or sparse datasets hinder proper learning and limit generalization;
3. **Boundary complexity:** the difficulty of the class boundary, which reflects the minimal description needed to represent the classes. The complexity of classification boundaries can be formally related to Kolmogorov complexity. However, since Kolmogorov complexity is incomputable in practice, approximations are made through the computation of *indicators* and *geometric descriptors*, drawn directly from the data.

In an attempt to formally characterize these factors, several families of complexity measures have been proposed. Ho and Basu (2002) originally divided them into three groups: (i) overlap of individual feature values, (ii) separability of classes, and (iii) geometry, topology, and density of manifolds. Sotoca et al. (2005) [21] proposed a similar division. More recently, Lorena et al. (2020) extended this work to consider six main families: *feature-based measures*, *linearity measures*, *neighborhood measures*, *network measures*, *dimensionality measures*, and *class imbalance measures*.

1. **Feature-based measures**: quantify how informative the available features are in separating classes.
2. **Linearity measures**: determine the degree to which the classes may be divided by a hyperplane, that is, if they are linearly separable.
3. **Neighborhood measures**: characterize the presence and density of classes in local neighborhoods, providing insight into class overlap and boundary complexity.
4. **Network measures**: create a graph model of the dataset and derive metrics for network statistical characterization.
5. **Dimensionality measures**: capture the effect of high-dimensional spaces on classification difficulty.
6. **Class imbalance measures**: capture the effect of uneven class distributions on classification.

Table 2.1 provides a summary of the characteristics of the complexity measures [12] across these families. Additionally, we will examine instance-level data typology as a complementary approach to understanding minority class complexity.

Table 2.1: Measures families, names, acronyms and ranges. Based on the work of Lorena et al. (2019)[12].

Category	Name	Acronym	Range
Feature-based	Maximum Fisher's discriminant ratio	F1	(0, 1]
	Directional vector maximum Fisher's discriminant ratio	F1v	(0, 1]
	Volume of overlapping region	F2	[0, 1]
	Maximum individual feature efficiency	F3	[0, 1]
	Collective feature efficiency	F4	[0, 1]
Linearity	Sum of the error distance by linear programming	L1	[0, 1)
	Error rate of linear classifier	L2	[0, 1]
	Non linearity of linear classifier	L3	[0, 1]
Neighborhood	Fraction of borderline points	N1	[0, 1]
	Ratio of intra/extra class NN distance	N2	[0, 1)
	Error rate of NN classifier	N3	[0, 1]
	Non linearity of NN classifier	N4	[0, 1]
	Fraction of hyperspheres covering data	T1	[0, 1]
	Local set average cardinality	LSC	$[0, 1 - \frac{1}{n}]$
Network	Density	Density	[0, 1]
	Clustering Coefficient	ClsCoef	[0, 1]
	Hubs	Hubs	[0, 1]
Dimensionality	Average number of features per dimension	T2	(0, m]
	Average number of PCA dimensions per points	T3	(0, m]
	Ratio of the PCA dimension to the original dimension	T4	[0, 1]
Class imbalance	Entropy of classes proportions	C1	[0, 1]
	Imbalance ratio	C2	[0, 1]

Note: n stands for the number of points in the dataset and m corresponds to its number of features.

Data typology: safe / borderline / rare / outlier. Following an established typology by Napierala et al. [22], it is useful to classify minority (although it can also be used for majority) instances according to the composition of their local neighbourhood (nearest neighbors). Concretely:

- **Safe instances.** Most of an instance's nearest neighbors (NNs) belong to the same class; these examples lie in relatively homogeneous regions and are easy to learn.
- **Borderline instances.** The neighbourhood contains a close-to-balanced mix of classes; these examples lie near decision boundaries.
- **Rare instances.** The majority of neighbours belong to the other class; such examples represent sparse local sub-concepts and are difficult to generalize.
- **Outliers.** All nearest neighbours belong to the other class; these are extreme or isolated points that are likely noise or exceptional cases.

This typology can complement the characterization of the difficulty of specific minority examples, and support the evaluation of some difficulty factors. For instance, *class overlap* (Section 2.1.3), is tightly tightly linked to the instance-level taxonomy above: it increases the proportion of borderline examples, amplifies the risk that oversampling methods (e.g., SMOTE) will generate synthetic points that cross class boundaries, and makes it harder to distinguish between true minority sub-concepts and noisy intrusions.

When assessing these measures, there are a few aspects to have under consideration:

1. These metrics cannot fully capture the data difficulty factors described above. With seldom exceptions (such as class imbalance measures), complexity measures aim to characterize general aspects of the data (e.g., class ambiguity, sparsity, boundary complexity) [23]. However, some can resonate with specific difficulty factors. For instance, some feature-based measures reflect some class overlap properties, whereas neighborhood or network measures can provide some insights into class decomposition, which relates to small disjuncts.
2. These metrics should not be interpreted in isolation. Indeed, each metric provides a different perspective on the problem and is often aligned with the behavior of specific classifier families. For this reason, they are best viewed as a complementary set of indicators: their true value lies in being jointly considered to form a more holistic understanding of dataset complexity.
3. Finally, although these metrics generally characterise the dataset as a whole, providing aggregate indicators over the entire domain, some can also be analysed per class (e.g., neighbourhood-based measures and data typology).

2.3. The Problem of Small Disjuncts

In this section, we focus on the specific problem of small disjuncts, presenting an overview on their impact on classification performance and the current related work and open challenges in the field.

2.3.1 Impact of Small Disjuncts in Classification Performance

The issue of *small disjuncts* was first identified by Holte et al. (1989), who defined them as “a disjunct that covers only a few training examples”. Their work was also the first to show that, despite representing only a small fraction of the data, small disjuncts accounted for a disproportionately large share of classification errors.

Building on this, Danyluk and Provost (1991) [24] demonstrated in a real-world telecommunications problem that small disjuncts posed an even greater challenge when combined with noise. Specifically, the presence of noise made it difficult to distinguish spurious patterns from true sub-concepts, aggravating the error rates.

Later, Weiss [25] investigated this relationship further. Using synthetic datasets, he confirmed that the joint presence of noise and small disjuncts significantly increased classification errors. In 1998, Weiss and Hirsh [26] revisited the problem, motivated by the findings of Danyluk and Provost. Their empirical results provided strong evidence that class noise not only increased the number of small disjuncts but also amplified the percentage of errors attributable to them. To some extent, they concluded that it was precisely this combination - noise and small disjuncts - that made learning particularly difficult.

Weiss and Hirsh extended this line of research in 2000 [27] by introducing the notion of *Error Concentration (EC)*, which quantifies the degree to which classification errors are concentrated in the smaller disjuncts. Their findings reinforced that most errors are indeed clustered within these regions. Furthermore, using the C4.5 decision tree algorithm, they showed that pruning strategies were ineffective in addressing the problem.

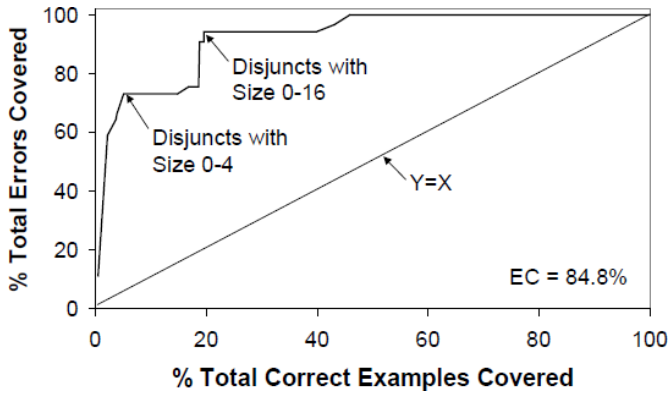


Figure 2.7: Error Concentration curve from Weiss, G.M. and Hirsh, H. 2000 [27]

Finally, Japkowicz and collaborators [9, 28] examined the related issue of class decomposition into multiple sub-concepts. While not always equivalent to small disjuncts, their work showed that the degree of decomposition could degrade performance more severely than class imbalance itself. When the decomposition yields sub-concepts small enough to cover very few examples, these can effectively be regarded as small disjuncts. This line of research thus reinforced the importance of small disjuncts and laid the groundwork for the development of specialized methods designed to address this persistent issue.

2.3.2 Specialized Methods to Handle Small Disjuncts

Research specifically targeting the problem of *small disjuncts* is relatively scarce. This scarcity stems not only from the limited amount of work devoted to the topic but also from the inherent difficulty of addressing such a challenging problem. Among the few existing strategies, the most well-known is the *Cluster-Based Oversampling (CBO)* method proposed by Jo and Japkowicz [15]. As its name suggests, CBO is a cluster-based approach that treats small disjuncts as sub-clusters or sub-concepts in the data. The method applies clustering - in their work, k-means - to identify these sub-clusters. Once identified, each sub-cluster is individually *inflated* using an oversampling procedure, thereby increasing the representation of minority regions that would otherwise remain underrepresented.

Other methods, while not explicitly designed for small disjuncts, could in theory, alleviate their impact indirectly by addressing class imbalance through refined oversampling techniques. One such example is MWMOTE [14], which first identifies hard-to-learn, informative minority class instances and assigns them weights according to their Euclidean distance to the nearest majority class samples. Synthetic instances are then generated from these weighted informative samples using a clustering mechanism, ensuring that all generated points lie within the boundaries of minority clusters. Other approaches that incorporate clustering-based procedures could also help alleviate the small disjuncts problems, such as SMOTE, SMOTE+TomekLinks, DBSMOTE, Cluster-SMOTE and CBO [29], which will be explained in further detail in Chapter 3.

Despite these contributions, one of the main challenges of such approaches lies in their experimental design. As noted by Santos et al. (2022) [5], many of these studies regarding small disjuncts rely on synthetic datasets with controlled characteristics, where the number, size and structure of sub-concepts are predefined by the researchers. This contrasts with real-world domains, where the number and structure of minority sub-concepts are rarely trivial to determine. Consequently, the applicability and effectiveness of these methods in practice remain an open research challenge.

2.3.3 Open Challenges

Open challenges in dealing with small disjuncts largely stem from the scarcity of research dedicated specifically to the problem and from its inherent difficulty, as already mentioned. Two major directions can be highlighted: the development of appropriate *metrics* and the design of more effective *approaches*.

Metrics. As discussed in previous sections, some existing data complexity measures indirectly capture aspects related to small disjuncts, such as local density or neighborhood overlap. However, these metrics are too generic to provide precise characterization. At present, there is no measure designed explicitly to quantify the presence, severity or impact of small disjuncts. The development of such metrics would constitute a significant step forward, as it would allow researchers to categorize small disjunct-related difficulties more systematically. This, in turn, would enable the tailoring of algorithms to address specific manifestations of the problem.

Approaches. The current algorithmic strategies also exhibit important limitations. Current cluster-based solutions are mostly based on k-means, for which the number of clusters k must be specified in advance and presupposes prior knowledge about the structure of the data. This is naturally problematic in real-world methods, where the number of concepts is unknown. Furthermore, k-means also assumes spherical clusters, which is often not a reality in most real-world domains.

Identification versus mitigation. It is also important to note that identification and mitigation of small disjuncts are distinct, altough related, research problems. Even if effective identification methods were available, the challenge of mitigating their negative impact would remain. Furthermore, small disjuncts rarely occur in isolation: they often appear alongside other data difficulty factors such as noise, class overlap and lack of density. Any practical solution must therefore consider these interactions, rather than treating small disjuncts as an isolated phenomenon.

In summary, the main open questions concern: (i) the absence of dedicated metrics to effectively identify small disjuncts in real-world domains, (ii) the dependency of existing mitigation methods on prior knowledge and (iii) the need for integrated approaches that combine identification and mitigation while accounting for concurrent data complexity factors.

3. Methodology

This chapter is organized into two main sections. The first concerns the development of our algorithm, IterDBSCAN, which we propose as an approach to the identification of small disjuncts. The second consists of an overview of the experimental study we conducted, including the motivations behind our design choices and the underlying factors that guided them, without yet presenting results. Later, Chapters 4 and 5 will further discuss the obtained results of our proposed approach and empirical benchmark, respectively.

3.1. Proposed Approach

3.1.1 IterDBSCAN Design

As introduced in earlier chapters and discussed in greater detail in Chapter 4, IterDBSCAN is our proposed strategy to address the problem of *within-class imbalance* manifested as small disjuncts - regions that "cover only a few training examples" [30]. Existing approaches rely heavily on clustering algorithms such as k-means or DBSCAN, but both present significant limitations.

For instance, k-means requires the number of clusters k to be specified a priori, while DBSCAN similarly requires the ε parameter to be defined in advance in order to establish neighborhoods. Both of these requirements assume prior knowledge about the data and its structure. However, as discussed in the previous chapter, there are currently no dedicated methods or metrics to support such analysis for small disjuncts. Additionally, k-means struggles with coping with complex, non-spherical clusters, as well as noisy or outlier examples.

In turn, DBSCAN is known to struggle with datasets containing regions of varying density. In such cases, DBSCAN may: (i) fragment a true cluster into multiple smaller clusters, or (ii) merge distinct clusters into a single one. The latter issue is often aggravated by the presence of what we refer to as "bridges" or "bridging points", which create artificial connections between clusters. While these concepts are examined in greater detail in the next chapter, it is important to note here that these limitations constitute the main motivation for the development of IterDBSCAN. Our algorithm is specifically designed to address these challenges and provide more robust handling of small disjuncts in imbalanced data.

3.1.2 Synthetic Datasets

To develop and validate IterDBSCAN, we employed several well-known synthetic datasets commonly used in the literature on imbalanced data - *paw*, *clover* and *subclus*.* What is important to emphasize here is their composition: although all datasets are two-dimensional, what is most relevant is that they differ substantially in structure and density. This diversity is crucial, as it allows us to validate IterDBSCAN across a range of distinct scenarios and ensures that the algorithm is not tailored to a single type of problem. Particular attention is given to the minority class, whose representation varies across datasets. In addition to these established benchmarks, we also generated a custom dataset, referred to as *seeds*, specifically designed to illustrate the bridging problem. By working with simple two-dimensional datasets, the manifestations of small disjuncts could be visually assessed and better understood before progressing to real-world data.

3.2. Empirical Benchmark

3.2.1 Experimental setup

After developing a stable version of IterDBSCAN, the next step was to test its robustness and behaviour with real-world datasets. An overview of the experimental pipeline is shown in Figure 3.1.

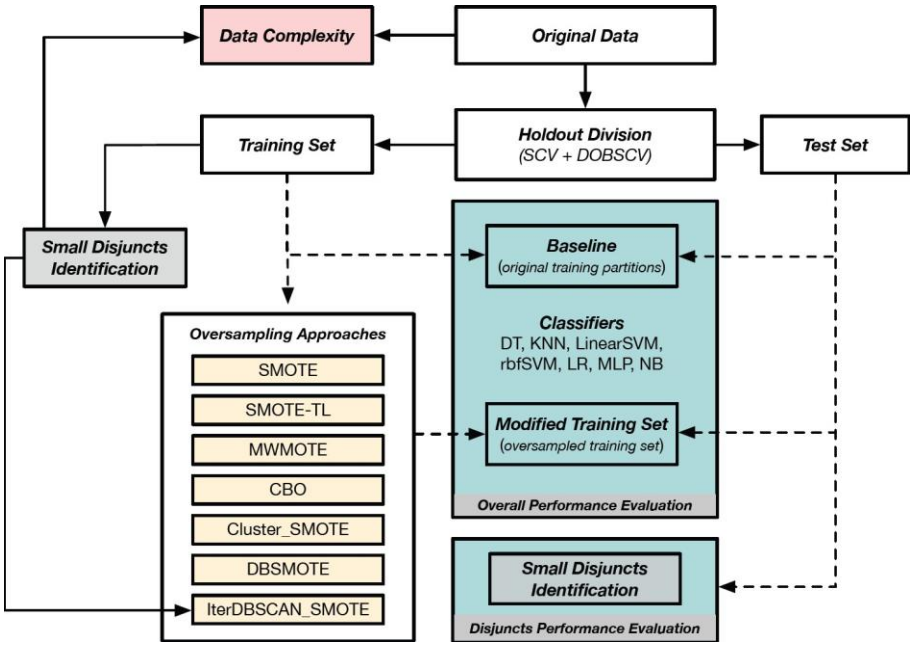


Figure 3.1: Experimental pipeline overview.

*Each of these datasets is described in greater detail in the following chapter.

We start by collecting real-world datasets, computing complexity metrics and performing a train/test split. Then, disjuncts are identified using IterDBSCAN in both the training and test partitions, and the classification results (overall and disjunct specific) are compared across several oversampling approaches.

In what follows, we provide the main details of each step of the experimental setup, while specifc details are discussed along the following sections:

Data collection. The benchmarking datasets were drawn from the KEEL imbalanced-data repository*, selecting binary classification problems with pronounced class skew (high imbalance ratio). In total we retained 78 datasets for the study.

Data Complexity. Each dataset was subjected to a data complexity analysis using the `problexity` library† [31], a Python package implementing a wide range of metrics that characterize the complexity of classification problems. The library builds on the work of Lorena et al. [12] and not only provides implementations of the metrics discussed in Section 2.2, but also offers functionalities for visualizing and comparing these metrics.

Train/test splitting. For each dataset we produced two distinct holdout configurations:

- **Split ratios:** two train/test partitions were used: (i) $50\%/50\%$ and (ii) $60\%/40\%$. These relatively large test partitions were chosen so that very small minority sub-clusters (small disjuncts) would not be entirely lost during splitting and so that their structure could be observed in both folds. This is required due to the significant impact of class imbalance, where datasets have a considerably high imbalance ratio.
- **Split strategies:** we used (i) standard stratified sampling (stratified holdout), and (ii) Distribution Optimally Balanced Stratified Cross-Validation (DOBSCV) [32], an alternative partitioning scheme that takes covariate shift into account (i.e., aims to keep the same structure in training and test partitions). Both strategies were applied to each ratio above, producing the experimental folds used throughout the remainder of the experiments.

*https://sci2s.ugr.es/keel/imbalanced.php
†https://problexity.readthedocs.io/en/latest/

IterDBSCAN configuration and disjunct identification. IterDBSCAN was applied to both training and test folds to map each instance to a cluster (or to noise). The following algorithmic parameters were explored:

- minPoints $\in \{2, 3\}$: the minimum number of points that can constitute a small disjunct.
- disjunct threshold $\tau \in \{0.3, 0.4, 0.5\}$: a cluster produced by IterDBSCAN is labelled a *small disjunct* if its size relative to the total number of minority examples in the fold satisfies the chosen threshold; clusters above the threshold are merged conceptually and treated as the main "minority cluster". The use of several τ values allows to perform a sensitivity analysis to the definition of what constitutes a small disjunct.

Resampling strategies. For each training fold and for each configuration of minPoints and τ, we considered a baseline with no resampling (No-SMOTE) and seven resampling methods. These include our proposed IterDBSCAN-focused oversampling strategy, which selectively oversamples the instances identified as belonging to small disjuncts (together with the main minority cluster), as well as six standard alternatives applied to the entire minority class: SMOTE, SMOTE+TomekLinks, MWMOTE, DBSMOTE, ClusterSMOTE and CBO.

Classification stage. After resampling, each training fold was used to train a suite of seven classifiers representing multiple model families. The classifiers were then evaluated on the untouched test folds (no resampling performed on test data).

Evaluation metrics. We computed standard performance metrics (e.g., AUC) as well as a set of disjunct-specific indicators that quantify error concentration and behaviour within identified small disjuncts. The full set of metric definitions and their motivations are presented later in the dedicated evaluation-metrics section.

Experimental combinations. Accordingly, our full experimental setup comprises:

$$78 \text{ datasets} \times 2 \text{ split ratios} \times 2 \text{ split strategies} \times 2 \text{ minPoints} \times 3\,\tau \times 8 \text{ resampling options} \times 7 \text{ classifiers} = 104{,}832 \text{ experimental runs in total.}$$

3.2.2 Datasets

Regarding the datasets used in our benchmarking, these were drawn from the KEEL imbalanced-data repository*. The selection criteria were based on ensuring variability in sample size, number of features and imbalance ratio (IR). Since KEEL is a publicly available repository widely adopted for benchmarking in imbalanced learning, the datasets were already preprocessed and cleaned, making them ready for direct use. A detailed summary of their main characteristics is provided in Table 3.1.

Table 3.1: Characteristics of imbalanced datasets.

Dataset	**Size**	**Features**	**IR**	***Dataset***	**Size**	**Features**	**IR**
ecoli-0-3-4_vs_5	200	7	9.00	glass5	214	9	22.78
yeast-2_vs_4	514	8	9.08	yeast-2_vs_8	482	8	23.10
ecoli-0-6-7_vs_3-5	222	7	9.09	lymphography-normal-fibrosis	148	18	23.67
ecoli-0-2-3-4_vs_5	202	7	9.10	flare-F	1066	11	23.79
glass-0-1-5_vs_2	172	9	9.12	car-good	1728	6	24.04
yeast-0-3-5-9_vs_7-8	506	8	9.12	car-vgood	1728	6	25.58
yeast-0-2-5-7-9_vs_3-6-8	1004	8	9.14	kr-vs-k-zero-one_vs_draw	2901	6	26.63
yeast-0-2-5-6_vs_3-7-8-9	1004	8	9.14	kr-vs-k-one_vs_fifteen	2244	6	27.77
ecoli-0-4-6_vs_5	203	6	9.15	yeast4	1484	8	28.10
ecoli-0-1_vs_2-3-5	244	7	9.17	winequality-red-4	1599	11	29.17
ecoli-0-2-6-7_vs_3-5	224	7	9.18	poker-9_vs_7	244	10	29.50
glass-0-4_vs_5	92	9	9.22	kddcup-guess_passwd_vs_satan	1642	41	29.98
ecoli-0-3-4-6_vs_5	205	7	9.25	yeast-1-2-8-9_vs_7	947	8	30.57
ecoli-0-3-4-7_vs_5-6	257	7	9.28	abalone-3_vs_11	502	8	32.47
yeast-0-5-6-7-9_vs_4	528	8	9.35	winequality-white-9_vs_4	168	11	32.60
vowel0	988	13	9.98	yeast5	1484	8	32.73
ecoli-0-6-7_vs_5	220	6	10.00	kr-vs-k-three_vs_eleven	2935	6	35.23
glass-0-1-6_vs_2	192	9	10.29	winequality-red-8_vs_6	656	11	35.44
ecoli-0-1-4-7_vs_2-3-5-6	336	7	10.59	ecoli-0-1-3-7_vs_2-6	281	7	39.14
led7digit-0-2-4-5-6-7-8-9_vs_1	443	7	10.97	abalone-17_vs_7-8-9-10	2338	8	39.31
glass-0-6_vs_5	108	9	11.00	abalone-21_vs_8	581	8	40.50
ecoli-0-1_vs_5	240	6	11.00	yeast6	1484	8	41.40
glass-0-1-4-6_vs_2	205	9	11.06	winequality-white-3_vs_7	900	11	44.00
glass2	214	9	11.59	winequality-red-8_vs_6-7	855	11	46.50
ecoli-0-1-4-7_vs_5-6	332	6	12.28	kddcup-land_vs_portsweep	1061	41	49.52
cleveland-0_vs_4	177	13	12.62	abalone-19_vs_10-11-12-13	1622	8	49.69
ecoli-0-1-4-6_vs_5	280	6	13.00	kr-vs-k-zero_vs_eight	1460	6	53.07
shuttle-c0-vs-c4	1829	9	13.87	winequality-white-3-9_vs_5	1482	11	58.28
yeast-1_vs_7	459	7	14.30	poker-8-9_vs_6	1485	10	58.40
glass4	214	9	15.47	shuttle-2_vs_5	3316	9	66.67
ecoli4	336	7	15.80	winequality-red-3_vs_5	691	11	68.10
page-blocks-1-3_vs_4	472	10	15.86	abalone-20_vs_8-9-10	1916	8	72.69
abalone9-18	731	8	16.40	kddcup-buffer_overflow_vs_back	2233	41	73.43
dermatology-6	358	34	16.90	kddcup-land_vs_satan	1610	41	75.67
zoo-3	101	16	19.20	kr-vs-k-zero_vs_fifteen	2193	6	80.22
glass-0-1-6_vs_5	184	9	19.44	poker-8-9_vs_5	2075	10	82.00
shuttle-c2-vs-c4	129	9	20.50	poker-8_vs_6	1477	10	85.88
shuttle-6_vs_2-3	230	9	22.00	kddcup-rootkit-imap_vs_back	2225	41	100.14
yeast-1-4-5-8_vs_7	693	8	22.10	abalone19	4174	8	129.44

3.2.3 Data Partitioning Schemes

In this work we employed two different strategies for splitting the data into training and testing folds. The first was the widely used *Stratified Holdout* (SCV), implemented via the `train_test_split` function of the `scikit-learn` library.

Nevertheless, SCV has a well-known drawback: since folds are generated through random subsampling, structural information in the data may not be preserved. As discussed

*https://sci2s.ugr.es/keel/imbalanced.php

earlier on the *Data Difficulty Factors* section, this can give rise to *dataset shift* [32], ultimately resulting in inaccurate or unstable performance estimates.

To consider an alternative method for comparison, we also adopted DOBSCV. This method, proposed by Moreno-Torres et al. [32], explicitly considers the structure of the data when generating folds. The key idea of DOBSCV is to avoid splitting nearby instances into the same fold, thereby ensuring that local neighborhoods and substructures remain represented across the partitions. DOBSCV therefore leads to the creation of more stable and representative folds that preserve the intrinsic structure of the data. In imbalanced datasets, ensuring a suitable training-test partition is crucial so that small disjuncts are not eliminated from the test set due to the splitting process.

3.2.4 Oversampling Approaches

In this study we benchmarked six oversamplers, in addition to our proposed method, yielding seven in total. When including the baseline (*No-SMOTE*) strategy, the total rises to eight. All the oversamplers, except for our proposal, were drawn from the `smote_variants`* Python library. In the baseline approach, no oversampling is performed, and the minority class distribution is left unchanged. For the remaining oversamplers (again, except our proposal), the minority class is oversampled until it matches the size of the majority class, following the standard behavior of `smote_variants`.

The considered oversamplers are as follows [29]:

- **SMOTE.** The classical Synthetic Minority Oversampling Technique interpolates between minority class samples and their nearest neighbors to generate synthetic instances.
- **Tomek Links.** A hybrid approach that first applies SMOTE and subsequently removes borderline or noisy instances by eliminating Tomek links, thereby refining the decision boundary.
- **MWMOTE.** Majority Weighted Minority Oversampling Technique assigns higher weights to hard-to-learn minority samples, where difficulty is determined by the Euclidean distance to their nearest majority neighbor (the “nearest enemy”). Synthetic samples are then generated preferentially from high-weight instances, reinforcing the difficult regions.

*https://pypi.org/project/smote-variants/

- **DBSMOTE.** A density-based approach that uses DBSCAN-like density estimation to identify dense minority regions. Synthetic points are generated within these dense areas, preserving the local distribution.
- **CBO (Cluster-Based Oversampling).** Based on the idea that small disjuncts represent minority sub-clusters, this method applies clustering (e.g., k-means) to identify minority subgroups. Each minority cluster is then individually oversampled. In our implementation, we applied a slight modification: only the minority class was oversampled (excluding the majority class). The number of clusters k was determined via the silhouette score, selected from the range $[2, \lfloor \sqrt{\text{size}_{\text{max}}/2} \rfloor + 1]$, where size_{max} is the size of the majority class.
- **Cluster-SMOTE.** Another clustering-based method that also applies k-means to partition the minority class into clusters, generating synthetic samples within them.

This selection of oversamplers covers a spectrum of strategies: a classical baseline (SMOTE), a hybrid method (Tomek Links), boundary-focused approaches (MWMOTE), density-based techniques (DBSMOTE), and clustering-based strategies (CBO, Cluster-SMOTE). Together, they provide a representative benchmark of state-of-the-art oversampling methods that might intuitively be suited to alleviate small disjuncts.

Our method. Building on the disjuncts identified by IterDBSCAN, our oversampling strategy differs from standard approaches in a key aspect: it preserves the relative representativity of clusters after oversampling. To illustrate, consider a dataset in which IterDBSCAN identifies one main minority cluster (defined by the threshold τ) and k small disjuncts. By definition, the main minority cluster will always be larger than the disjuncts. After oversampling, we ensured that this relative distribution was maintained, rather than inflating all clusters to the same size, which would not reflect reality.

Formally, the oversampling budget is computed as:

$$\Delta N = N_{\text{maj}} - N_{\text{main}} + \sum_{i=1}^{k} N_{\text{SD}_i},$$

where N_{maj} is the size of the majority class, N_{main} the size of the main minority cluster, and N_{SD_i} the size of each of the k identified small disjuncts.

This budget Δ is then divided equally among the $k + 1$ minority clusters (the main cluster plus the k disjuncts). Each cluster therefore receives an additional:

$$\Delta_c = \frac{\Delta}{k+1},$$

so that the updated size of each cluster is given by

$$N_c' = N_c + \Delta_c, \qquad c \in \{\text{main}, \text{SD}_1, \ldots, \text{SD}_k\}.$$

Example. Suppose the majority class has $N_{\text{maj}} = 100$ instances. IterDBSCAN identifies one main minority cluster with $N_{\text{main}} = 20$ points and two small disjuncts with $N_{\text{SD}_1} = 5$ and $N_{\text{SD}_2} = 3$. The current minority total is therefore $20 + 5 + 3 = 28$.

The oversampling budget is:

$$\Delta = 100 - 28 = 72.$$

Since there are $k + 1 = 3$ minority clusters (one main + two disjuncts), the budget is divided equally:

$$\Delta_c = \frac{72}{3} = 24.$$

The updated cluster sizes are then:

$$N_{\text{main}}' = 20 + 24 = 44, \quad N_{\text{SD}_1}' = 5 + 24 = 29, \quad N_{\text{SD}_2}' = 3 + 24 = 27.$$

The final minority total is $44 + 29 + 27 = 100$, perfectly matching the majority class while maintaining the original relative structure between clusters.

3.2.5 Classifiers

To evaluate the impact of different oversampling strategies we considered six well-known families of classifiers, spanning a spectrum from simple, interpretable models to more complex, flexible ones. This diversity was chosen deliberately, as is typical in benchmarking studies, in order to capture how oversampling methods interact with models of different capacities, inductive biases and sensitivities to data irregularities. In other words, by including classifiers from multiple learning paradigms, we aim to assess whether the observed effects are consistent across families, or whether certain oversamplers or data

structures favor particular types of learners. All classifiers were implemented using the scikit-learn* Python library with their default configurations.

- **Decision Trees.** From the decision tree family, we employed the C4.5 algorithm, a widely used tree-induction method. At each split, the attribute that best partitions the data - typically using information gain or a similar criterion - is chosen. This approach is simple, interpretable and able to capture non-linear decision boundaries.

- **Instance-based Learning.** As our instance-based model we selected k-Nearest Neighbors (k-NN). In this approach, a new sample is assigned to the class most frequently represented among its k closest neighbors in the training set, where closeness is usually determined by the Euclidean distance. This model is particularly attentive to local structure and class overlap, making it a relevant choice in the context of disjuncts.

- **Support Vector Machines (SVMs).** We included two SVMs: one with a linear kernel and another with a Radial Basis Function (RBF) kernel. SVMs aim to find an optimal separating hyperplane that maximizes the margin between classes. The use of kernels allows SVMs to implicitly map the data into higher-dimensional spaces, where separation may be more feasible. The linear kernel provides a simpler model, while the RBF kernel captures more complex non-linear relationships.

- **Linear Models.** From the linear family, we adopted Logistic Regression, a standard model for binary classification. Logistic regression fits a linear decision boundary by estimating parameters that maximize the likelihood of the observed class labels under a logistic function. It is interpretable and serves as a baseline for linearly separable data.

- **Bayesian Classifiers.** We employed Naïve Bayes, a probabilistic model grounded in Bayesian decision theory. It assumes conditional independence among features given the class label and builds a model from the prior probability of each class and the likelihood of features within that class.

- **Neural Networks.** Finally, we used a Multi-Layer Perceptron (MLP), a feedforward artificial neural network composed of layers of interconnected neurons. Each connection has an associated weight, which is optimized during training. The MLP

*https://scikit-learn.org/stable/

learns by backpropagation, where errors are propagated backward through the network and weights are updated via gradient descent. Its capacity to learn non-linear relationships makes it a natural representative of more complex, high-capacity models.

3.2.6 Performance Metrics

To ensure a comprehensive evaluation we employed two complementary groups of performance metrics. The first group consists of well-established global measures commonly used to evaluate any classifier, namely *Precision*, *Recall*, *F1-score* and the *Area Under the ROC Curve* (AUC)*. The second group focuses specifically on the behavior of classifiers with respect to *small disjuncts*, following the work of Weiss [33]. This distinction is important, as global measures alone may mask poor performance on some minority patterns, which are central to our study. In the following subsections, we present each metric in detail, providing both its formal definition and its methodological motivation.

3.2.6.1 Classification Metrics

The foundation of most classification metrics lies in the *confusion matrix*, which summarizes the counts of correctly and incorrectly classified instances across positive and negative classes. From this matrix, the following quantities are derived:

- **True Positive Rate (TPR)** or *Recall/Sensitivity*: proportion of actual positive instances that are correctly identified.
- **False Positive Rate (FPR)**: proportion of negative instances that are incorrectly identified as positive.

The Receiver Operating Characteristic (ROC) curve plots TPR against FPR across different decision thresholds, and the **AUC** provides a threshold-independent summary of classifier discrimination. Higher AUC values indicate stronger overall separability between classes.

In addition, we employ **Precision**, which measures the proportion of predicted positive instances that are indeed positive:

*All global metrics are computed with respect to the *positive class*, which in our experimental setting corresponds to the **minority class**.

$$\text{Precision} = \frac{TP}{TP + FP}$$

where TP denotes true positives and FP false positives. Complementarily, **Recall** measures the proportion of true positives that the classifier successfully identifies:

$$\text{Recall} = \frac{TP}{TP + FN}$$

where FN denotes false negatives. Because Precision and Recall often trade off against one another, we report their harmonic mean, the **F1-score**:

$$F1 = 2 . \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

This metric is particularly relevant in the context of imbalanced data, as it balances the need to correctly identify minority class instances (Recall) with the requirement to avoid excessive false positives (Precision). In the case of small disjuncts, Recall is especially critical, since misclassifying rare but informative patterns would significantly undermine classifier reliability.

3.2.6.2 Disjunct-Specific Metrics

While global measures provide an overall view of classifier performance, they do not adequately capture behavior on small disjuncts. As demonstrated by Weiss [33], minority classes frequently exhibit disproportionately higher error rates compared to majority classes, with misclassifications concentrated in rare regions of the input space. To address this, we propose a set of disjunct-oriented metrics, inspired by the Error Concentration (EC) metric developed by Weiss [27] for rule-based classifiers.

In our framework, disjuncts are defined relative to the minority class distribution using the threshold parameter τ introduced in Section 3.2.1. Specifically, a cluster produced by IterDBSCAN is considered a *small disjunct* if its size is smaller than $\tau \cdot N_{min}$, where N_{min} denotes the total number of minority instances in the current fold. Clusters larger than this threshold are conceptually merged and treated as the main minority cluster. Consequently, the minority class is composed of: (i) main minority cluster, (ii) all small disjuncts and (iii) minority-class instances labeled as noise.

To characterize classification performance over small disjuncts, we propose the following following four disjunct-specific measures:

- **MinPoints Covered**: proportion of minority-class instances that belong to a small disjunct

$$\text{MinPointsCovered} = \frac{|D|}{N_{min}},$$

  where D denotes the set of minority-class instances contained in small disjuncts.

- **Disjunct Error Rate**: proportion of misclassified instances within small disjuncts

$$\text{DisjunctErrorRate} = \frac{|D_{err}|}{|D|},$$

  where D_{err} denotes the subset of misclassified instances in disjuncts.

- **Error Coverage**: proportion of all minority-class misclassifications that occur within disjuncts

$$\text{ErrorCoverage} = \frac{|D_{err}|}{|M_{err}|},$$

  where M_{err} is the set of all misclassified minority-class instances.

- **Correct Coverage**: proportion of all correctly classified minority-class instances that belong to disjuncts

$$\text{CorrectCoverage} = \frac{|D_{corr}|}{|M_{corr}|},$$

  where D_{corr} is the set of correctly classified instances in disjuncts and M_{corr} the set of all correctly classified minority-class instances.

Together, these measures allow us to quantify classifier performance not only in aggregate but also within structurally rare and error-prone regions of the minority class. This dual perspective ensures that improvements on global performance metrics do not obscure deficiencies in capturing minority patterns.

4. Identifying Small Disjuncts

In this chapter we introduce and explore the IterDBSCAN algorithm, our proposed approach for the identification of small disjuncts. At its core IterDBSCAN is a modification of the DBSCAN clustering algorithm designed to address the issue of varying densities. When the input space has varying densities, DBSCAN struggles to find solutions that are able to represent the concepts in data, especially in the presence of some singular data points - referred to as *bridging points* or *bridges* - which create unintended or meaningless connections between distinct sub-concepts, compromising clustering quality by creating links where should not be any. To better understand what *bridges* represent, we should first examine how the current DBSCAN algorithm works - its behavior, its limitations, the motivations behind the development of IterDBSCAN as well as the process that brought it to its current state.

The goal of our approach is to eliminate these non-informative interconnections and with that produce more robust and meaningful concepts. Additionally, we incorporate an automatic optimization procedure that evaluates clustering results using a multi-criteria approach to identify the optimal solution.

4.1. DBSCAN Algorithm

4.1.1 Background

The DBSCAN algorithm, first introduced by Ester et al. [34], is a density-based spatial clustering method designed to identify both clusters and noise in a spatial dataset. It operates by grouping together points that are within a specified distance threshold and labeling as noise those points that lie in low-density regions - specifically, points that do not belong to the neighborhood of any other point.

DBSCAN operates based on two user-defined parameters: the neighborhood radius (ε) and the minimum number of points (minPts) required to characterize a region as dense. It is defined as follows:

- The collection of all points q in a dataset D such that the distance between p and q is less than or equal to ε is known as the ε-neighborhood of a point p.
- A data point p is called a *core point* if the number of data points contained in its ε-neighborhood is greater than or equal to the threshold value minPts.

- A point p is said to be *directly density-reachable* from a point q with respect to ε and minPts if p belongs to the ε-neighborhood of q and if the number of points within this neighborhood is at least minPts, meaning that q qualifies as a core point.
- A point p is said to be *density-reachable* from a point q with respect to ε and minPts if there exists a sequence of points $p_1, p_2, \ldots, p_n$ such that $p_1 = q$, $p_n = p$, and each point p_{i+1} is *directly density-reachable* from p_i.
- All points (core and non-core) that can be reached from p form a cluster if p is considered a *core point*.
- Any point that cannot be reached from another point is regarded as noise.

The algorithm starts by choosing a random point that has not been visited yet and retrieves the ε-neighborhood of that point. If the neighborhood contains a number of points equal to or greater than minPts, a new cluster is initiated. If not, the point is marked as noise. When a point is determined to be part of a dense region its ε-neighborhood is included in the cluster as well. All points within this neighborhood are added to the cluster and if any of those are also dense, their neighborhoods are likewise explored and added.

This process repeats until the entire density-connected cluster is identified. The algorithm then moves on to another unvisited point, which may lead to the formation of another cluster or the identification of more noise.

4.1.2 Limitations

While DBSCAN is a widely used clustering approach, it suffers with varying densities, as we explain in what follows.

For the sake of simplicity, consider the example shown in the following figure (Figure 4.1).

In the example, two clusters are linked by a few points lying in a moderately less populated region, forming what looks like a bridge between them. DBSCAN can return two different solutions, determined by the choice of ε:

- **Small** ε – If ε is not large enough to reach the bridge points, the algorithm identifies two separate clusters.
- **Large** ε – If ε is large enough to include the bridge points, they create a path that connects the two clusters, and DBSCAN merges them into a single cluster.

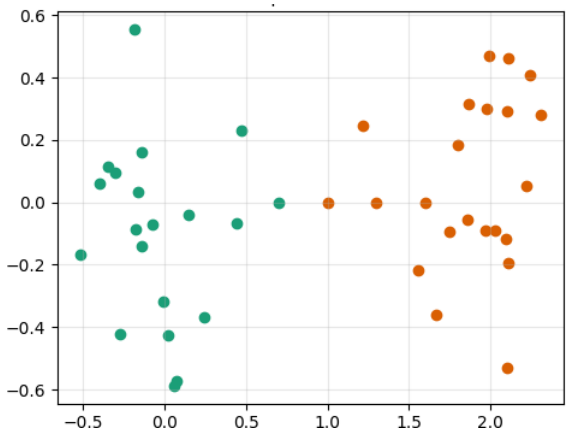


Figure 4.1: Two clusters separated by a bridge of points in a low density region.

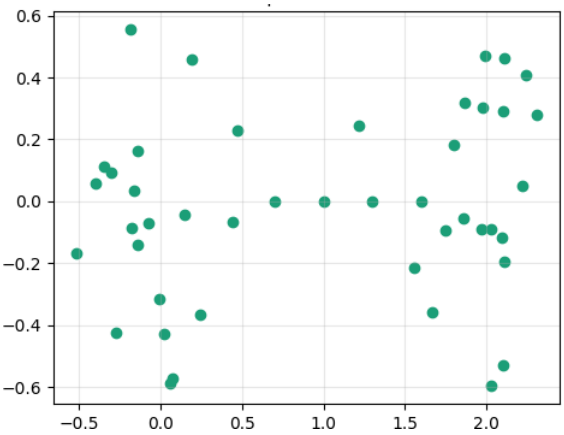


Figure 4.2: Two clusters unified by a bridge of points.

Even though Figure 4.1 does not show additional bridge points, it still illustrates a clear problem of varying densities which causes standard DBSCAN to perform poorly. With smaller values of ε, instead of producing two clusters, DBSCAN often generates multiple fragmented clusters. Conversely, with larger values of ε, distinct clusters may be incorrectly aggregated into a single cluster, especially if they are located in close proximity. Even when the underlying concepts are cohesive - so that a small ε could, in principle, separate them - bridging points between clusters can still cause DBSCAN to merge them into one.

In real-world domains, where the number and structure of existing concepts is unknown, the optimal value of epsilon is impractical to define beforehand. Smaller values of ε might work locally but may be too small to form meaningful connections elsewhere in the space whereas a larger ε might be preferable for the overall dataset, yet it still aggregating concepts that we would prefer to keep separate due to bridging points.

A perhaps more meaningful solution, given the structure of this domain, would be one where ε is large enough to establish well defined connections - those that can be validated by various clustering metrics, such as the Silhouette score - while ignoring the points that would link sub-concepts into a somewhat degenerated cluster, one where sub-concepts get diluted.

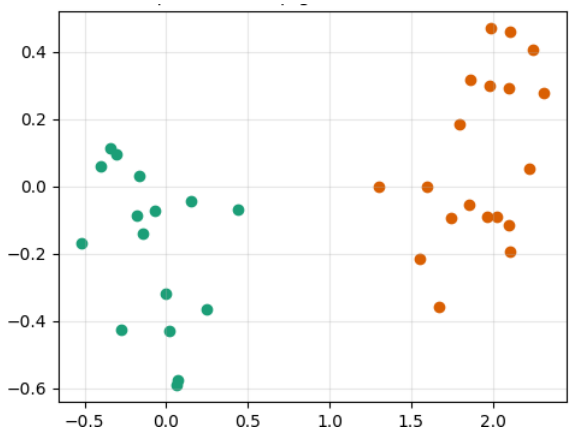


Figure 4.3: Clustering solution with bridge points removed.

4.2. IterDBSCAN Algorithm

In this section we will explain the development process of the IterDBSCAN algorithm - how it started, how it evolved and how it reached its final form. It was an iterative process that gradually refined the solution and walking through each step is essential to fully understand the problem and how our algorithm attempts to tackle it. Each stage will be explained in detail, including definitions, images and a final version of the algorithm.

4.2.1 Growth Computation

We began with a straightforward implementation of the core idea behind this algorithm: if a point truly belongs to a cluster, the density in its vicinity should be higher than that of a point forming a link between sub-clusters.

This intuition can be formalized using the concept of neighborhoods. If a point x_i has more neighbors than a point x_j for a given ε, it is reasonable to assume that x_i lies in a denser region of the space.

To address this, we introduced the notion of **growth** defined as the increase in a point's neighborhood size as the ε value grows from ε_1 to ε_2. Specifically, for each point, we compute the difference between the number of neighbors at ε_2 and those at ε_1. This growth-based logic forms the foundation for filtering out potential bridging points.

We then defined a **growth threshold**: a value below which points are excluded from the clustering process. To compute this threshold we first select only valid growth values (those not considered noise) and then compute their mean and standard deviation. The threshold is defined as:

$$threshold = \mu - \alpha \cdot \sigma$$

where μ and σ are the mean and standard deviation of the valid growth values and α is a user-defined scaling coefficient (GROWTH_THRESHOLD_MULTIPLIER) that controls sensitivity to low-growth points. Any point whose neighborhood growth falls below this threshold is excluded from clustering.

In the initial iterations, we adopted a fixed scaling factor for the second epsilon, such that:

$$\varepsilon_2 = \varepsilon_1 \times 1.5$$

However, this fixed scaling approach was discarded in later versions, for reasons discussed further in this section.

It is also important to note that the initial epsilon is defined as the smallest nonzero pairwise distance between any two points in the space. Additionally, the maximum pairwise distance is computed to define the upper bound for epsilon growth, allowing the algorithm to terminate once this maximum is reached.

Identifying bridge points based solely on the number of neighbors at a single value of ε across all points proved problematic. In particular, legitimate but less dense concepts were at risk of being misclassified as bridges and discarded prematurely. To address this limitation we moved away from a global approach - in which every point and its neighborhood evolution were compared against all other points and their neighborhoods - and adopted a local strategy.

In this revised approach, we first apply DBSCAN with a fixed ε to generate initial clusters. Then, **growth** comparisons are made strictly between the points within each individual cluster. This shift to intra-cluster comparison significantly mitigated the issue of erroneously eliminating points belonging to sparser, yet valid, clusters.

A natural question arises: what happens to the points that do not belong to any cluster at this stage? For these unclustered points we considered two possible strategies:

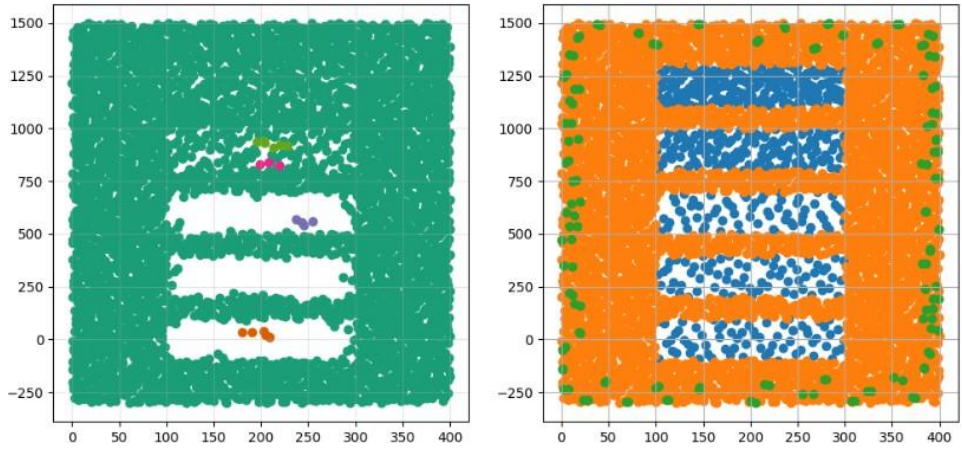


Figure 4.4: **Left:** Output of the old algorithm approach, which removed nearly all points within the rectangular spaces. **Right:** Ground truth showing the three classes in the original dataset.

1. **Compare them globally to all points in the dataset:** These points would be compared to every point, including those already assigned to clusters and those that are not.

2. **Restrict comparisons to only other unclustered points:** In this case, these points would only be compared to points that do not yet belong to any cluster at the current iteration.

The first option risks reintroducing the very problem we sought to solve - the premature removal of edge points that might later be incorporated into a valid cluster at a higher ε value. This occurs because points already assigned to a cluster are, by definition, in denser regions of the space. Their **growth** during clustering is therefore larger than that of unclustered points, which naturally exhibit smaller growth. As a result, comparing unclustered points against clustered ones could lead to the elimination of edge points that should otherwise remain candidates for inclusion. Therefore, we adopted the second strategy. By limiting comparisons to unclustered points we preserved the potential for those points to join meaningful structures in subsequent iterations.

4.2.2 Saturation

As development progressed we identified a new issue affecting neighborhood growth. Consider a dense, circular cluster with points distributed throughout - some located near the center and others at the periphery.

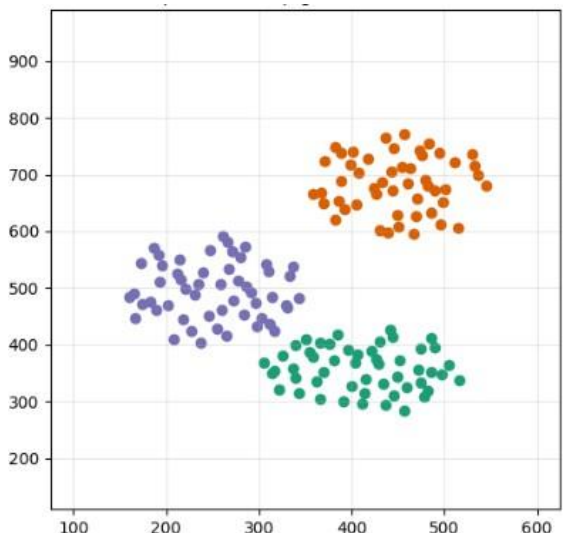


Figure 4.5: Illustration of dense circular clusters, with points near the center and at the edge.

As the iterative process continues and the ε value increases, the number of neighbors for each point generally grows. However, points near the center of a dense cluster reach a stable neighborhood size much earlier than those on the periphery.

This discrepancy results in central points appearing to exhibit little to no growth during later iterations - not because they are unimportant, but simply because they have already reached the cluster limit. Consequently, these central points risk being misclassified as non-contributory or even as noise. To prevent such misclassification, we introduced the concept of **saturation**.

A point is considered **saturated** if it already has a substantial number of neighbors and exhibits diminishing neighborhood growth. Saturation is controlled by a tunable parameter, SATURATION_THRESHOLD, which by default is set to 0.7.

We compute the **saturation** level of a point as the ratio of its current number of neighbors to its potential number of neighbors:

$$s(x) = \frac{n_{\text{current}}(x)}{n_{\text{potential}}(x)}$$

Where:

- $n_{\text{current}}(x)$ is the number of neighbors at the current ε_1, and
- $n_{\text{potential}}(x)$ is the number of neighbors at a larger ε value, ε_2*.

*At this time $\varepsilon_2 = \varepsilon_1 \times 1.5$

A point is considered saturated if:

$$s(x) \geq \text{SATURATION\_THRESHOLD}$$

Saturated points are not removed from the dataset. Instead, they are granted **immunity** from exclusion, ensuring that well-integrated points within dense regions remain protected from being mistakenly discarded.

To build an intuition for this behavior, consider the function $f(x) = \frac{x}{x+4}$ that can be seen on Figure 4.7. As x increases, the ratio approaches 1. Similarly, as the number of neighbors grows, the saturation level asymptotically approaches the threshold, signaling that the point is becoming saturated.

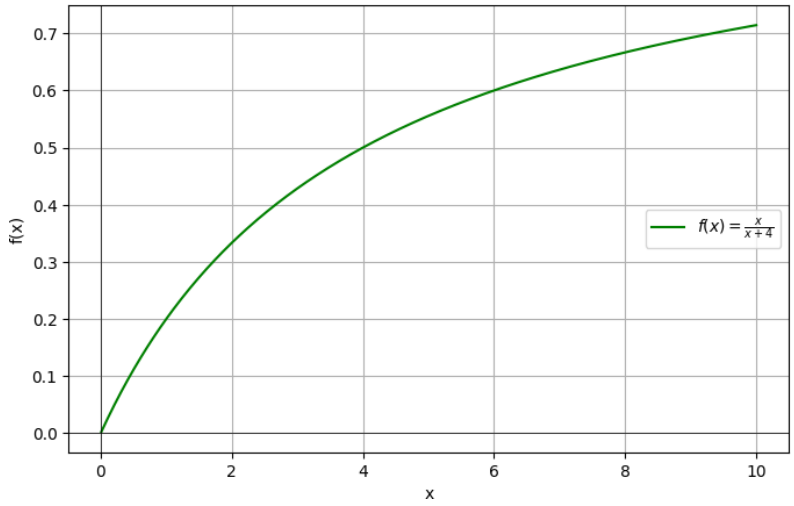


Figure 4.6: Saturation intuition: the function $\frac{x}{x+4}$ shows growth slowing, approaching a stable limit.

After saturation is accounted for, we update the mask of valid points to exclude both noise and saturated points. New growth statistics are computed only for these valid points and any valid point whose **growth** falls below the local threshold and is **not saturated** is marked as noise.

4.2.3 Adaptive Step Sizing

At this stage the algorithm had reached a relatively mature form: irrelevant points were being successfully filtered and relevant ones were retained. However, one component still required refinement - the growth behavior of the ε parameter across iterations.

It is important to distinguish between two uses of ε within the algorithm. First, there is the **iteration epsilon**, which governs the current DBSCAN run (e.g., ε_1, ε_2, etc., with $\varepsilon_1 < \varepsilon_2$).

Second, there is the **growth epsilon**, used to evaluate neighborhood expansion, the potential neighbors, by comparing a point's neighbors at ε and at a slightly larger value of ε. In earlier iterations, this larger value was calculated simply as:

$$\varepsilon_2 = \varepsilon_1 \times 1.5$$

While straightforward, this fixed scaling proved problematic, especially in the presence of dense clusters. The step between consecutive iteration values (i.e., $\varepsilon_2 - \varepsilon_1$) was often too large, causing the algorithm to expand neighborhoods too quickly. This overgrowth diluted the algorithm's ability to detect meaningful structural changes: nearly all points showed growth, and few (if any) were identified as uninformative. The process became too coarse-grained.

To address this issue, we turned to the idea of adaptive step sizing [5]. The goal is to use a clustering coefficient, Davies-Bouldin (DB) score to guide epsilon growth. The DB score evaluates clustering quality as the average similarity between each cluster and its most similar neighbor based on the ratio of within-cluster scatter to between-cluster separation. Since lower DB scores correspond to more distinct and well-separated clusters, our goal is to define an adaptive step sizing strategy that slows epsilon expansion as the clustering quality improves, allowing a convergence towards an optimal configuration.

We therefore define an adaptive **growth factor** as:

$$g = 1 + 0.1 \cdot \left(1 - e^{-\frac{D}{\rho}}\right)$$

where:

- g is the adaptive growth factor,
- D is the Davies-Bouldin score,
- ρ is a tunable parameter called the $\mathrm{STEP\_GROWTH\_RATIO}$.

And then update ε accordingly. The $\mathrm{STEP\_GROWTH\_RATIO}$ ρ controls the sensitivity of this adaptive mechanism and can be adjusted by the user. By default, it is set to 0.5.

This ensures that when the clustering quality is already high (i.e., when the DB score is low), the epsilon increases slowly, allowing the algorithm to explore the local structure in finer detail. Conversely, when the DB score is high (indicating poor clustering), the growth factor becomes larger, enabling faster expansion to escape suboptimal configurations.

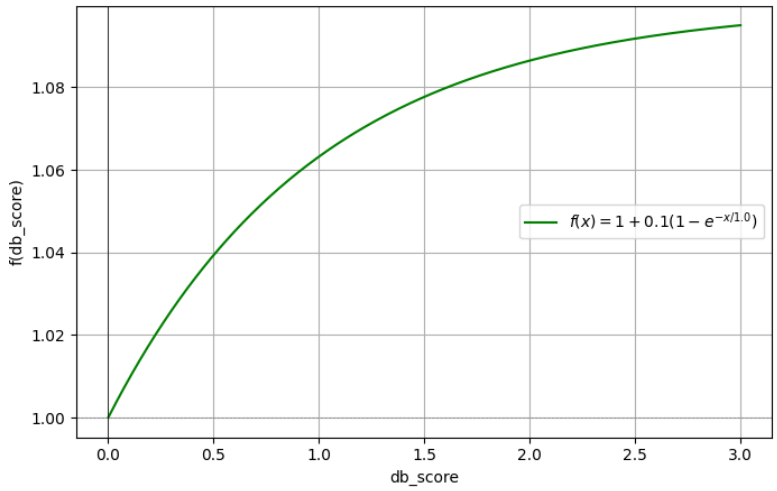


Figure 4.7: Growth factor with a STEP_GROWTH_RATIO of 1 and a DB Score between 0 and 3.

It is also important to note that, just as the process slows down when approaching a good solution, it also accelerates when moving away from poor ones. This occurs in two situations: the first is at the beginning of the process, when a small ε produces many small and uninformative clusters; in this case, the search speeds up to avoid wasting time on unpromising configurations. The second situation arises after a good solution has already been identified, at this stage the process has slowed to its minimum pace in order to refine that solution. However, as clustering quality begins to deteriorate beyond this point, the search once again accelerates to move away from it efficiently. This tuning mechanism is not blind: it is designed not only to identify what can be considered an optimal solution but also to do so in a time and cost-efficient manner.

Finally, we recalculated the **growth epsilon** used for neighborhood comparison (i.e., to compute the potential neighbors) in a way that reflects the most recent step using the difference between the current and previous ε values:

$$\varepsilon_{\text{growth}} = \varepsilon_t + (\varepsilon_t - \varepsilon_{t-1})$$

Taking all these considerations and developments into account the final version of the algorithm follows the flowchart shown in the next page.

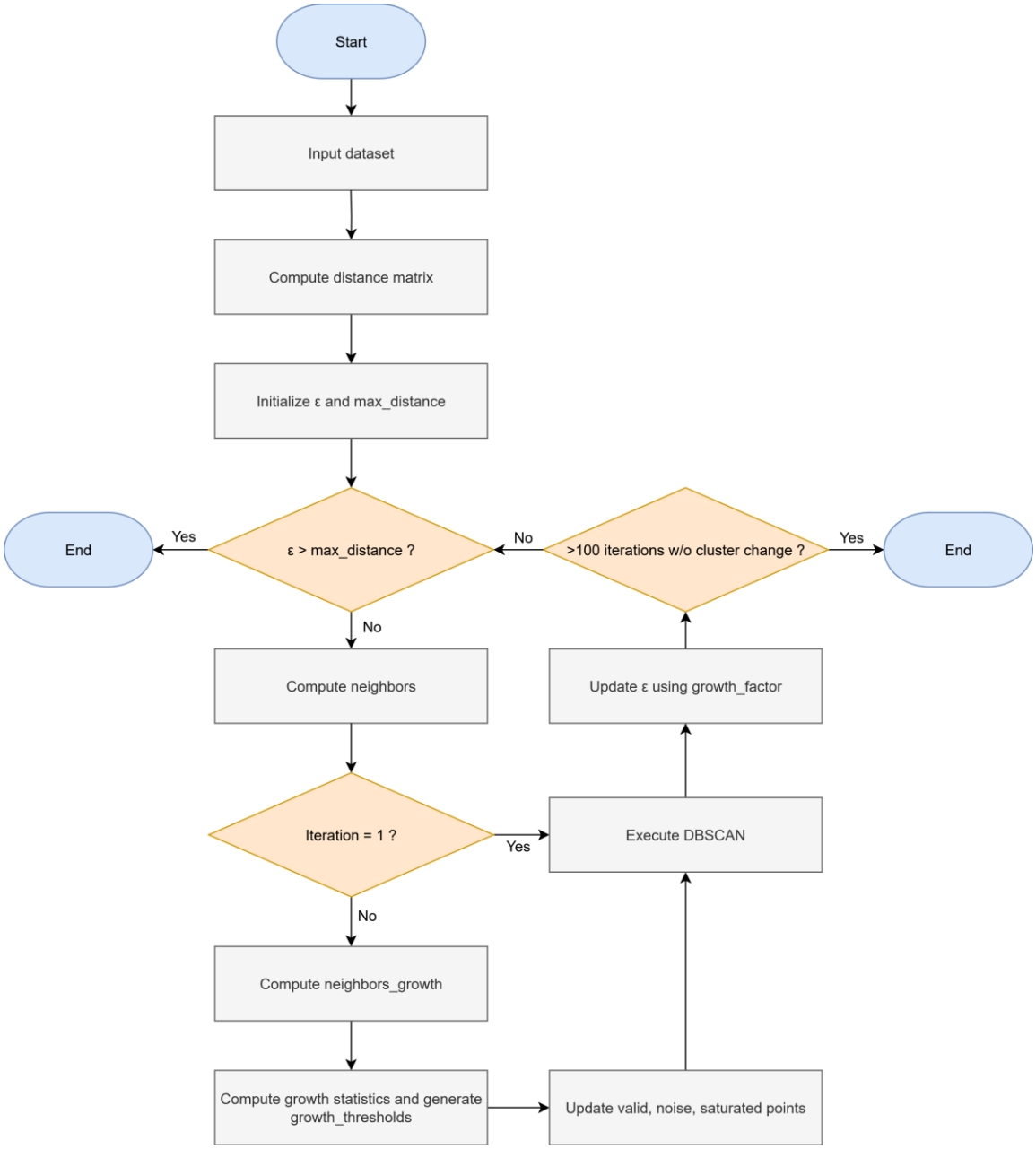


Figure 4.8: IterDBSCAN Algorithm flowchart.

4.2.4 Optimal Cluster Selection

Once the dataset has been fully processed through the previous algorithm, the final task is to determine which iteration yields the most meaningful clustering result. This is where the evaluation phase becomes important: an auxiliary algorithm is used to identify what we define as the *optimal configuration*, i.e., the clustering state at a specific iteration in which bridge points have been removed, relevant points retained and sub-concepts structure best revealed.

To guide this selection we employ a multi-criteria scoring strategy, evaluating each iteration using the following four metrics:

1. **Davies–Bouldin (DB) Score -** Evaluates clustering quality.
2. **Noise Point Ratio -** Measures the proportion of points marked as noise.
3. **Cluster Count Stability -** Rewards configurations with a reasonable* number of clusters.
4. **Iteration Progression -** Encourages mid-iteration selections, avoiding extremes.

We will explore each criteria in what follows. Then, we explain in detail how the overall scoring algorithm integrates them into a final clustering solution.

The **Davies–Bouldin (DB) Score**, as introduced in previous sections, is a key metric for evaluating clustering quality. This is our primary quality metric and carries the most weight in the final evaluation. Although we also experimented with the Silhouette Score, the results were largely consistent with those obtained using the DB Score. Given that the DB Score was already integrated into the clustering algorithm itself, and to maintain methodological consistency, we chose to proceed with this metric as the main basis for evaluation.

Since noise points (i.e., points classified as outliers and subsequently removed) accumulate and are carried forward through successive iterations, our approach must guard against configurations where a disproportionate fraction of the dataset is discarded - excessive pruning poses a significant risk of eliminating meaningful structural information from the data. To address this concern, we introduce the **Noise Point Ratio** metric. This

*By *reasonable*, we mean avoiding solutions that produce either too many clusters (fragmented solutions, typically resulting from very small ε values) or too few clusters (aggregated solutions, typically resulting from very large ε values).

metric quantifies the proportion of data points classified as noise during any given iteration.

$$\textbf{Noise Point Ratio} = \frac{\text{Number of noise points in iteration}}{\text{Total data points in original dataset}}$$

Lower values are preferable, as they indicate that the clustering configuration preserves the majority of the original data.

Cluster Count Stability is a metric that addresses a subtle but important issue: internal clustering scores like the Davies-Bouldin or Silhouette coefficient often favor extremes. For instance:

- At early iterations with very small ε the algorithm may produce a large number of small, tight clusters - sometimes hundreds. These configurations often receive deceptively high scores due to high intra-cluster cohesion.
- Conversely, at later iterations, ε may grow large enough that nearly all points are assigned to a single large cluster, which can also be scored favorably depending on the data distribution.

However, from a practical standpoint, both extremes are typically undesirable. A solution with an excessively large or small number of clusters rarely captures the true structure of a concept.

To regularize this we define a *window of optimal number of clusters* derived from the size of the dataset. The minimum and maximum acceptable number of clusters are derived from logarithmic and square-root transformations of the dataset cardinality.

$$\text{optimal}_{\min} = \max\left(2, \frac{\log(n)}{2}\right), \qquad \text{optimal}_{\max} = \max\left(5, \frac{\sqrt{n}}{4}\right)$$

If an iteration's number of clusters falls within this optimal range, it receives a full score. Otherwise, a smooth exponential decay function reduces the score proportionally to its distance from the range boundaries.

$$\text{decay}_{\text{factor}} = \max\left(1, \frac{\text{optimal}_{\max} - \text{optimal}_{\min}}{2}\right), \quad N(t) = 1 - \left(1 - e^{-\frac{\text{distance}}{\text{decay}_{\text{factor}}}}\right)$$

The **Iteration Progress** metric provides temporal context in the iteration timeline. The intuition is rooted in the typical behavior of both the cluster count and the Davies–Bouldin (DB) score over successive iterations:

- At early iterations (small ε), many small, well-separated clusters are formed, often resulting in good DB scores.
- As ε increases, the number of clusters begins to drop. The DB score may decrease or fluctuate depending on the merging dynamics.
- Eventually, the system converges to a small number of clusters, or even a single cluster, which often reflects a loss of meaningful resolution.

Our goal is to favor *middle iterations*, where the number of clusters is still informative and the clustering structure remains rich.

With the four evaluation metrics defined - **Davies–Bouldin (DB) Score**, **Noise Point Ratio**, **Cluster Count Stability**, and **Iteration Progress** - the final step is to normalize these scores so that each lies within the range $[0, 1]$ and then combine them into a unified scoring function to identify the most effective clustering iteration. According to our preliminary experiments, we have found the following weights to be informative:

- **Davies–Bouldin Score**: 2.0
- **Noise Point Ratio**: 1.5
- **Cluster Count Stability**: 1.5
- **Iteration Progress**: 0.5

The final score for each iteration is calculated as a weighted sum of the normalized values of these metrics.

$$\text{Final Score} = 2.0 \cdot \text{DB}_{\text{score}} + 1.5 \cdot \text{Noise}_{\text{score}} + 1.5 \cdot \text{Cluster}_{\text{score}} + 0.5 \cdot \text{Iter}_{\text{score}} \quad (4.1)$$

Finally, taking these metrics into account, we examine sliding windows - by default, windows of five iterations - to identify positive trends, i.e., sequences in which the evaluation score shows consistent improvement. In this context we focus specifically on the **Davies–Bouldin (DB) Score**. Among the solutions that exhibit such positive trends we then select the one with the highest final score, avoiding the selection of random local peaks and ensuring that the algorithm converges towards a stable solution.

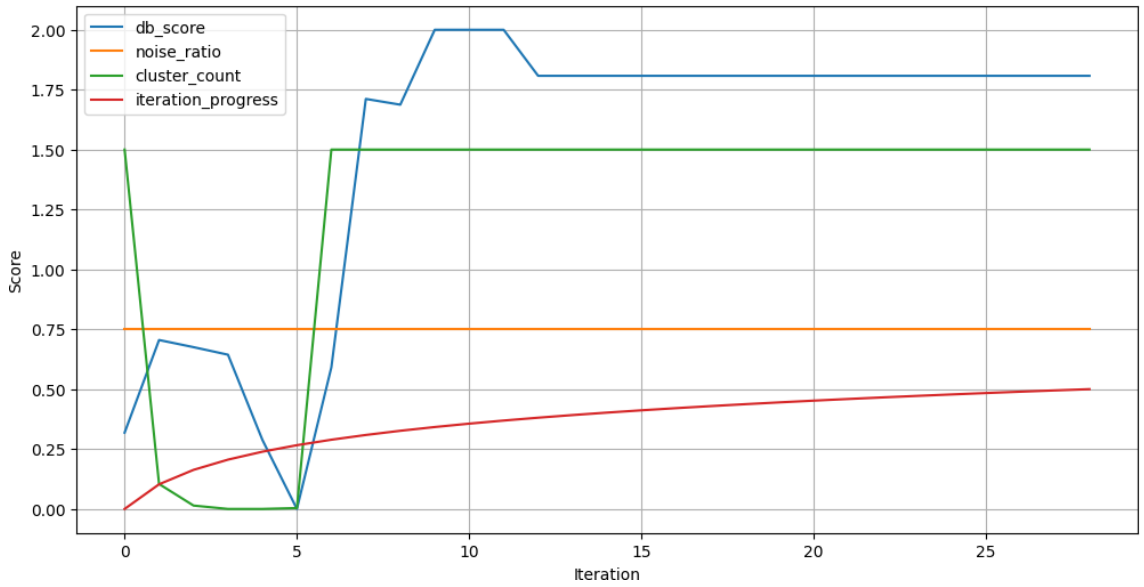


Figure 4.9: Breakdown of individual metric scores per iteration.

4.3. Validation on Synthetic Datasets

Throughout the development process we relied on synthetic datasets to test and tune the behavior of our proposed algorithm, IterDBSCAN. This served as an initial validation step, allowing the problem of small disjuncts to be visually assessed before moving to real-world datasets where the existence and structure of small disjuncts are unknown. Using low-dimensional synthetic data enabled us to directly evaluate whether our method was progressing toward a robust solution in well-established domains where the problem can be controlled (i.e., there is a known ground truth).

In real-world domains, where the number and shape of small disjuncts are not known in advance, we could rely on dimensionality reduction techniques such as PCA or t-SNE to visually assess IterDBSCAN. Such methods are indeed employed in our later results section for high-dimensional real-world datasets; however, at this stage, our focus was on immediate interpretability. It is important to note that dimensionality reduction comes with caveats and visual evaluation on a reduced input space can only be trusted to a limited extent.

For these reasons, we first pursued validation on well-established synthetic datasets, namely: *paw*, *clover/flower* and *subclus*, which have been widely employed in research on small disjuncts and data imbalance [10, 35, 36]. In addition, we generated an extra dataset, *seeds*, specifically designed to simulate the challenging scenario of touching clusters.

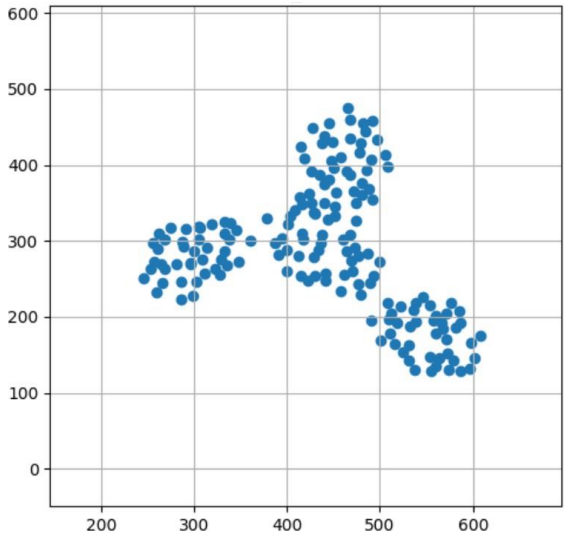
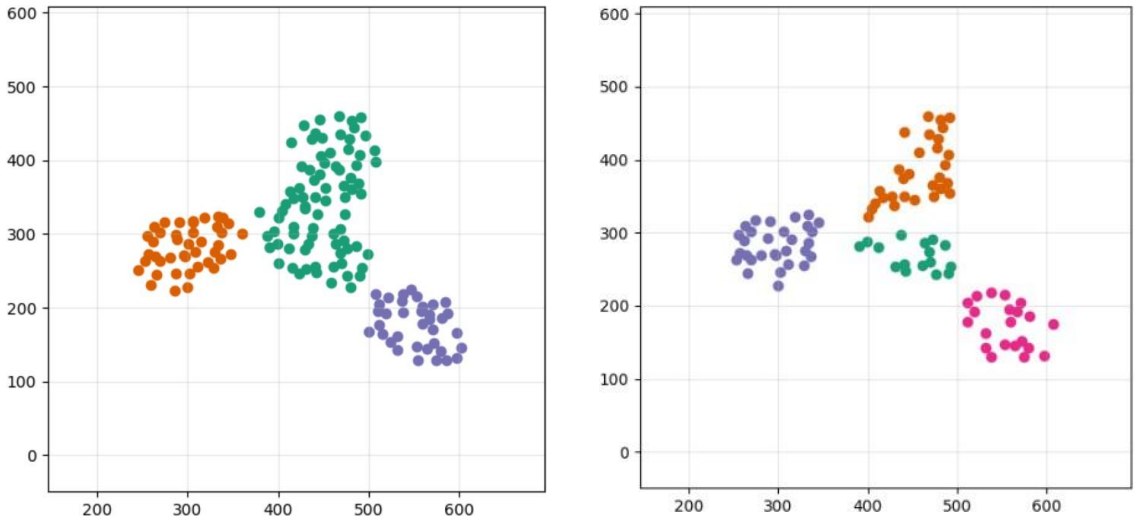


Figure 4.10: *paw* dataset.

As shown in Figure 4.10, the *paw* dataset exhibits a clear clustering structure with three to four distinct, well-separated, and rounded groups of observations. The left and right clusters maintain strong internal cohesion while remaining clearly distinguishable from the others, suggesting that they represent separate classes. The situation is less straightforward for the central regions: the middle and top groups may be interpreted either as a single elongated cluster with an “8” or peanut-like shape, or as two distinct clusters with partial overlap (the upper part of the middle cluster and the lower part of the top cluster).

Since clustering is an unsupervised technique, there is no single source of truth when evaluating cluster solutions; in other words, the output of a clustering algorithm is often subject to subjective interpretation. A common characteristic across multiple synthetic datasets we employed is that there is no single, indisputably correct clustering outcome. For example, in the *paw* dataset, Figure 4.10, one could reasonably argue for an optimal solution with three distinct clusters: left, right, and a combined middle-top region. Alternatively, a four - cluster interpretation could be proposed, assigning separate clusters to the left, right, middle, and top regions.

Our proposed approach is capable of producing both solutions by adjusting the `STEP_GROWTH_RATIO` parameter and modifying the optimization algorithm’s weightings. The results are presented below:



(a) The *paw* dataset with 3 clusters.

(b) The *paw* dataset with 4 clusters.

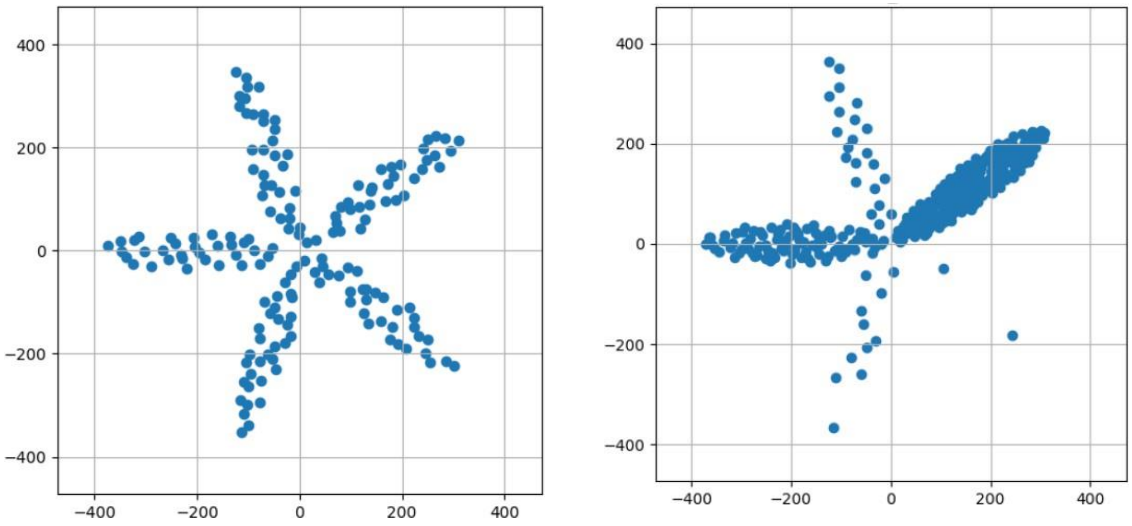
Figure 4.11: Comparison of *paw* with 3 and 4 clusters.

Although we were able to obtain a four-cluster segmentation, we ultimately deemed it suboptimal. In this case, the additional separation introduced excessive fragmentation and resulted in the loss of meaningful structural information, accordingly, we consider the three-cluster configuration to be the optimal solution. Standard DBSCAN can also achieve this result but only after carefully tuning the ε parameter, by contrast, our algorithm converges to this solution naturally by considering the adjusted step and multi-objective criteria.

The *clover* dataset, shown in Figure 4.12a, presents a clear structure composed of multiple elongated branches radiating from a central region. Each branch can be interpreted as a distinct cluster, although the connections at the origin introduce ambiguity regarding whether these should be considered independent groups or part of a larger unified structure.

The *sparse-clover* dataset, shown in Figure 4.12b, represents a more challenging variation. While some branches maintain dense, well-formed clusters, others are fragmented into sparser regions with lower point density.

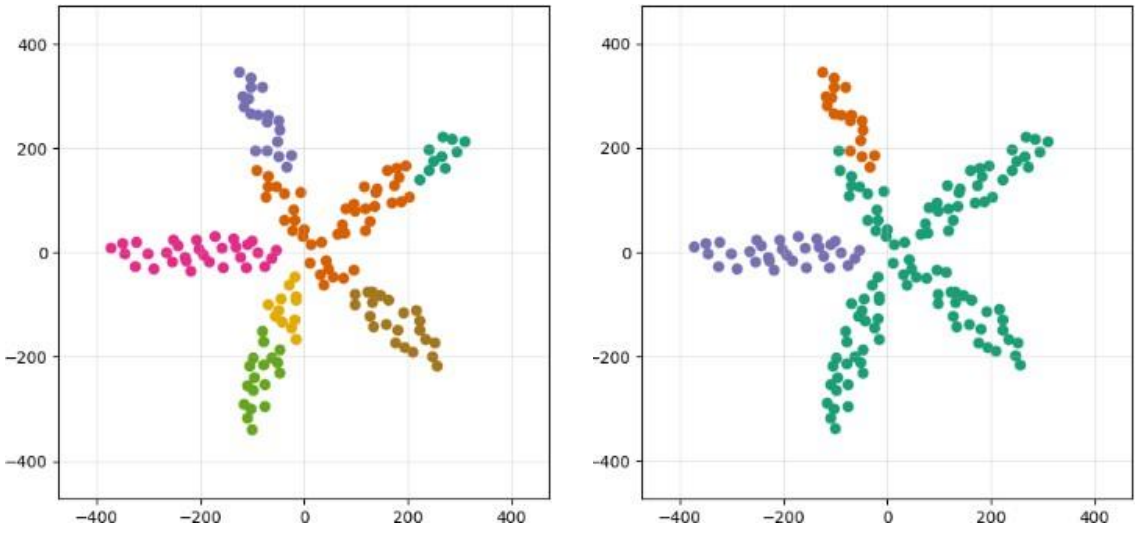
On the *clover* dataset, a standard DBSCAN approach would likely struggle to avoid connecting all petals, since the density across points appears relatively uniform. In contrast, when applied to the *sparse-clover* dataset, DBSCAN would instead have difficulty identifying the less dense petals as distinct clusters or would consider them as part of the main cluster.



(a) The *clover* dataset. (b) The *sparse-clover* dataset.

Figure 4.12: *clover* and *sparse-clover* datasets.

An important advantage of our approach is its ability to produce a valid and interpretable clustering solution immediately without requiring iterative parameter tuning. As with the *paw*, one could still engage in discussion about which solution is most appropriate: should we consider the segmentation shown in Figure 4.13a, or perhaps the configurations illustrated in Figure 4.13b? Such ambiguity reflects the inherent subjectivity in defining an "optimal" clustering in cases where multiple plausible partitions exist.



(a) *clover* variant 1. (b) *clover* variant 2.

Figure 4.13: Comparison of three *clover* dataset variants.

If we examine the *sparse-clover* dataset example, Figure 4.14, we can see that for the exact same ε value, our method is able to identify two branches of the clover by removing certain bridge points near the center. In contrast, standard DBSCAN merges them into

a single cluster. It is worth noting that the ε chosen in this example is intentionally small, such that the remaining branches are treated as noise rather than valid clusters.

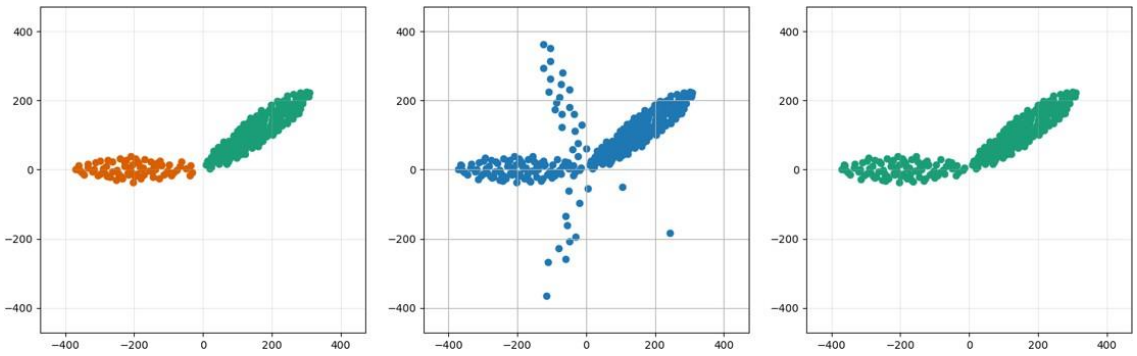


Figure 4.14: *sparse-clover* dataset - iterDBSCAN vs DBSCAN.

An interesting example is provided by the *seeds* dataset (Figure 4.15). As can be seen, the *seeds* dataset consists of three dense clusters that are close to one another with partial overlap between the left and bottom ones. While each cluster maintains internal cohesion, the proximity between them creates a challenging scenario for DBSCAN as relatively small ε values may split the bottom and left clusters into multiple fragments, whereas larger ε values risk merging them into a single aggregated structure. Unlike the previous cases this dataset presents a clear ground truth: we unanimously consider the optimal partition to consist of three clusters. However, two of these clusters lie in close proximity, making their separation challenging.

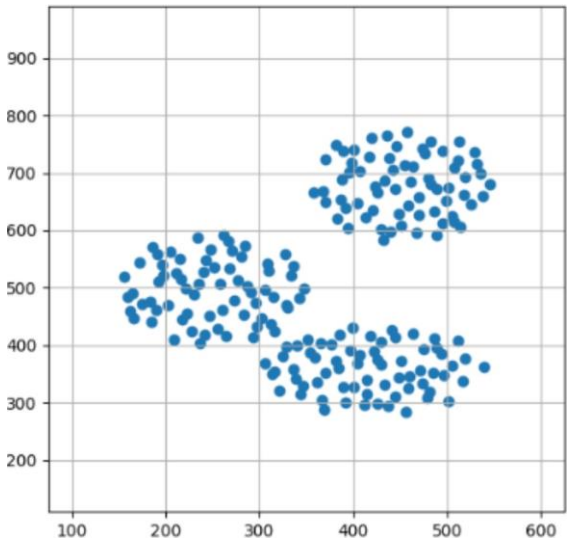
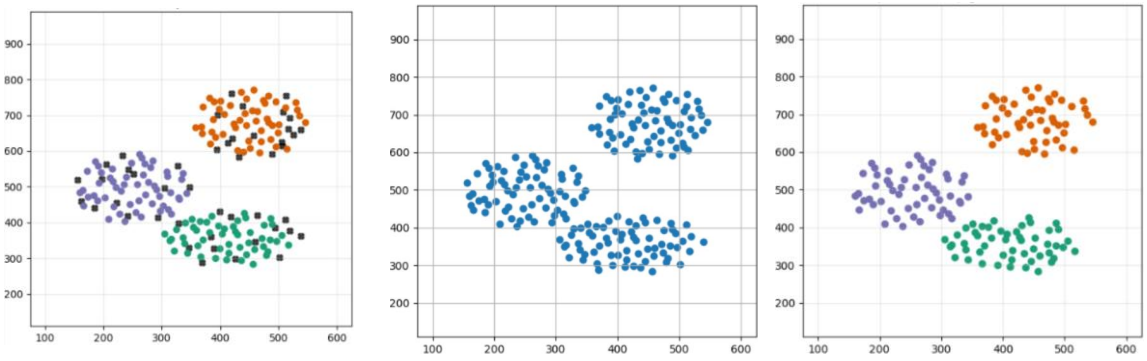


Figure 4.15: *seeds* dataset.

In our multiple attempts with the standard DBSCAN it failed to recover this three-cluster configuration. While it could be theoretically possible to achieve this by first identifying two preliminary clusters and then applying a distance-based separation procedure between

their boundary points, such an approach requires additional processing and is not inherent to DBSCAN.

In contrast, our algorithm produced the correct three-cluster solution on the first attempt, it effectively identified and removed a single bridge point situated between the two closely positioned clusters, thereby preventing their erroneous merging and yielding the intended separation.



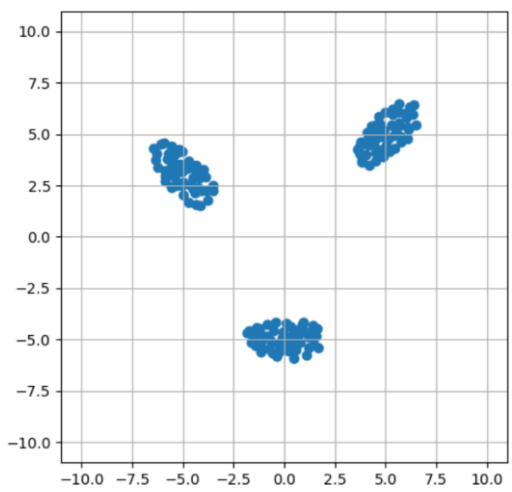
(a) *seeds* removed points.

(b) *seeds* dataset.

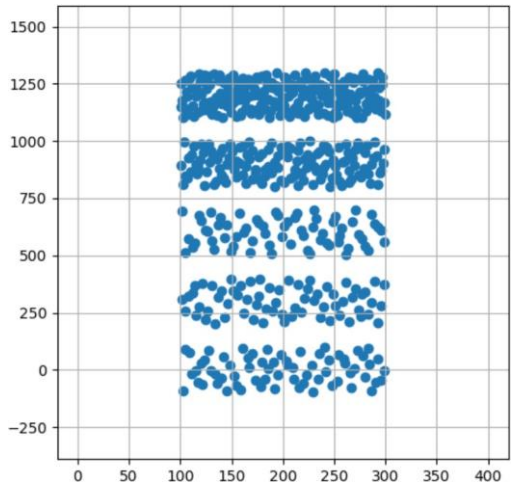
(c) *seeds* with 3 clusters.

Figure 4.16: IterDBSCAN results on the *seeds* dataset.

Additional datasets, including *paw_2* and *subclus*, were also examined. However, these proved to be less relevant from a methodological standpoint, as the bridging phenomenon was either absent or not sufficiently pronounced to meaningfully affect the clustering outcome. The *subclus* dataset does, however, present an interesting characteristic: it contains five clusters, each with a distinct density. The varying densities problem is already known to be problematic for DBSCAN. In such cases, a small value of ε would likely fragment denser clusters into multiple subclusters, whereas a large ε could cause sparser clusters to merge into a single cluster. Surprisingly, neither of these expected outcomes occurred in practice.



(a) *paw_2* dataset.



(b) *subclus* dataset.

Figure 4.17: *paw_2* and *subclus* datasets.

4.4. Discussion and Conclusions

With the methodology established and the results presented in the preceding and following chapters, it is clear that IterDBSCAN offers distinct advantages over classical approaches for addressing small disjuncts and, more broadly, improving clustering quality.

The strengths of IterDBSCAN can be summarized as follows:

- **Pruning of irrelevant or noisy points:** The algorithm is capable of removing points that are deemed misleading or noise-inducing. This pruning process prevents the formation of aggregated sub-concepts, resulting in more coherent and meaningful cluster structures.
- **No explicit specification of ε required and a multi-objective solution search:** Unlike DBSCAN, IterDBSCAN does not rely on a manually defined ε parameter. Instead, it employs a *multi-objective automatic solution search* that evaluates candidate solutions with multi-criteria scoring approach to identify the optimal result. Provided that the default parameters are appropriate for the user's application (these are discussed later in this section), the procedure is straightforward: the user supplies a dataset, specifies minPts*, and receives the solution the algorithm considers optimal. This simplicity makes the method practical, efficient and effective.

*minPts is strictly necessary as it reflects the user's definition of what constitutes a small disjunct. In our imbalanced datasets we established this value as 2 or 3. However, in large-scale datasets, a small disjunct could correspond to an arbitrary larger number of points

However, the approach is not without drawbacks as both the clustering procedure and the optimization component include several tunable parameters. This flexibility can be advantageous - since different datasets require different strategies, parameter tuning allows adaptation to specific problems, conversely, it also introduces additional complexity, requiring users to make informed choices. Ideally, the algorithm would:

1. Provide robust generalization across datasets, or
2. Adapt its parameters automatically based on dataset characteristics.

The clustering algorithm relies on the following four parameters:

- **minPts**: identical to DBSCAN's parameter, defining the minimum number of points to form a cluster.
- **GROWTH_THRESHOLD_MULTIPLIER**: a coefficient controlling sensitivity to low-growth points.
- **SATURATION_THRESHOLD**: the value above which points are considered saturated.
- **STEP_GROWTH_RATIO**: regulates the sensitivity of inter-iteration growth, with respect to the Davies–Bouldin score.

The optimization algorithm introduces its own set of parameters:

- **weights**: weights for the four evaluation metrics used to select the optimal clustering.
- **STABILITY_WINDOW**: the number of iterations considered when evaluating the stability trend of the final score (e.g., using a five-iteration window).
- **STABILITY_THRESHOLD**: the maximum allowed variation for determining whether a trend is stabilizing toward an optimal solution.

Although IterDBSCAN introduces a few number of parameters, for most of the tested datasets the default configuration already provides generally good results. Moreover, at the optimization level, the parameters appear to be relatively robust, as small adjustments

do not significantly affect the outcome. By contrast, parameters directly related to the clustering process may in some cases lead to noticeable variations, particularly in real-world scenarios. These will therefore be the most relevant candidates for dataset-specific exploration, an aspect that we discuss later on.

We also summarize additional directions for future research. One relates to the point removal process, while the current approach - based on saturation and data locality - is reasonable, incorporating a point's *relative position within and between clusters* could yield better decisions. For instance, if a point x_i in cluster C is at risk of removal but lies far from any other cluster, it might not warrant removal unless cluster pruning is explicitly desired. This positional awareness could also guide the automatic adjustment of thresholds and ratios, reducing or eliminating the need for manual parameter selection. Furthermore, these parameters might not need to be global; given that different substructures in the same dataset can vary in density and shape, locally adaptive parameters could prove more effective.

Another limitation is scalability. For very large datasets, the iterative nature of the algorithm - requiring repeated metric computations - can become computationally expensive. While this is inherent to the design, efficiency could be improved. For example, the growth ratio could be calculated based on the expansion within the current iteration, rather than relative to the previous one.

On the optimization algorithm the current weighting scheme is manually defined by the user. Ideally, the weights should be determined via a formal optimization procedure, ensuring they are statistically or algorithmically optimal for the given dataset.

Other limitation stems from the choice of distance metric. At present, the algorithm relies exclusively on Euclidean distance, which may not be ideal in all cases. Incorporating alternative distance measures could improve flexibility and performance across diverse domains. Likewise, the current implementation requires prior one-hot encoding (or similar transformations) to handle nominal features; exploring methods that directly accommodate categorical data would broaden the applicability of the approach.

5. Benchmark over Real-World Datasets

In this chapter we present the results obtained from applying the proposed experimental setup, detailed in Chapter 3. While the previous chapter described the methodological steps of the pipeline, this chapter focuses on analyzing and interpreting the experimental outcomes.

The results reported in this chapter were obtained following *minPoints* $= 2$ and the disjunct threshold $\tau = 0.4$, evaluated through DOBSCV with a $50\%/50\%$ train–test split. This choice reflects the parameter combination that provided the most consistent and beneficial results across the evaluated datasets regarding the existence of small disjuncts.

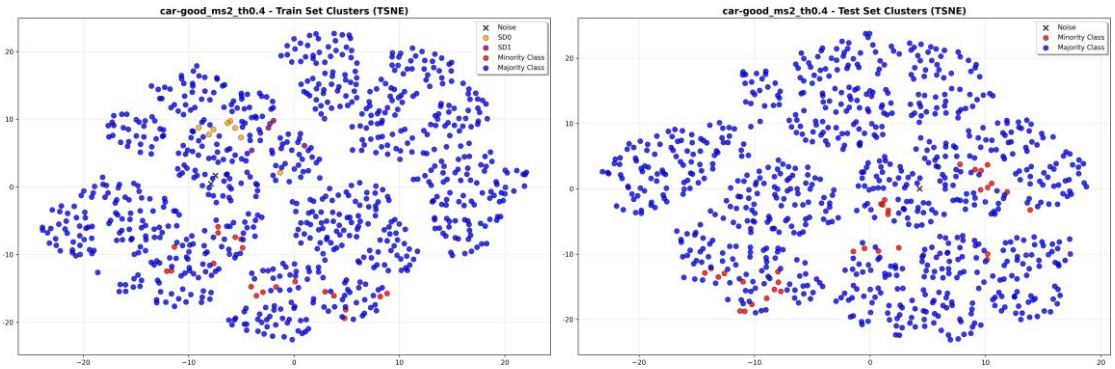
5.1. DOB vs. SCV

As introduced in Chapter 3, two partitioning approaches were employed to split the data into training and testing sets. The first was the typical stratified holdout provided by the scikit-learn library, while the second was DOBSCV, a partitioning scheme designed to preserve the structural properties of the data and to mitigate covariate shift between train and test splits. The latter approach was ultimately chosen for our analysis, as it reduced the mismatch between the distributions of the training and testing sets - a challenge that is particularly difficult to handle in the context of small disjuncts due to their inherently low cardinality, especially combined with a high imbalance ratio.

As illustrated in Figures 5.1 and 5.2, which show the t-Distributed Stochastic Neighbour Embedding (t-SNE) [37], partitions generated by DOBSCV tend to produce similar training/test partitions, within similar structures and proportion of small disjuncts and noisy points. In turn, in several of the tested datasets, the SCV partitions tended to lose the representation of the small disjuncts in the test set, which hinders the performance evaluation on disjunct-specific indicators.

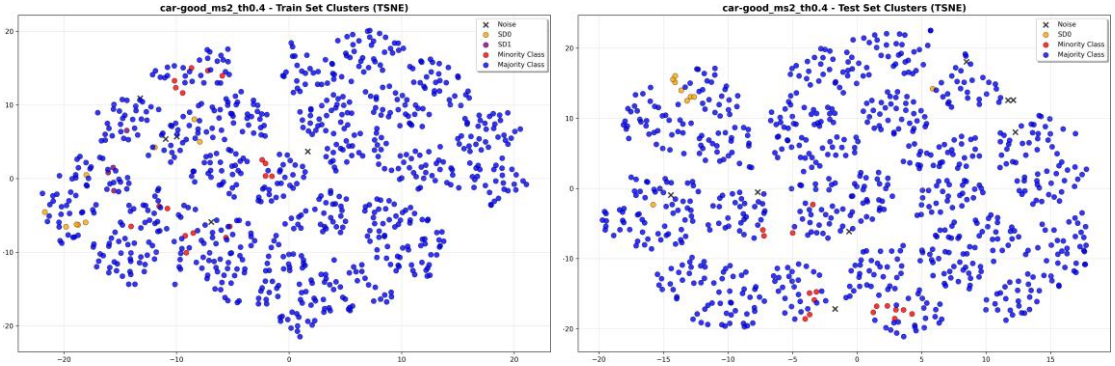
In another example (Figures 5.3 and 5.4) DOBSCV produces similar structural characteristics in both splits suggesting it is more effective at addressing covariate shift during partitioning.

For these reasons, it was selected as the partitioning strategy for all results in this chapter.



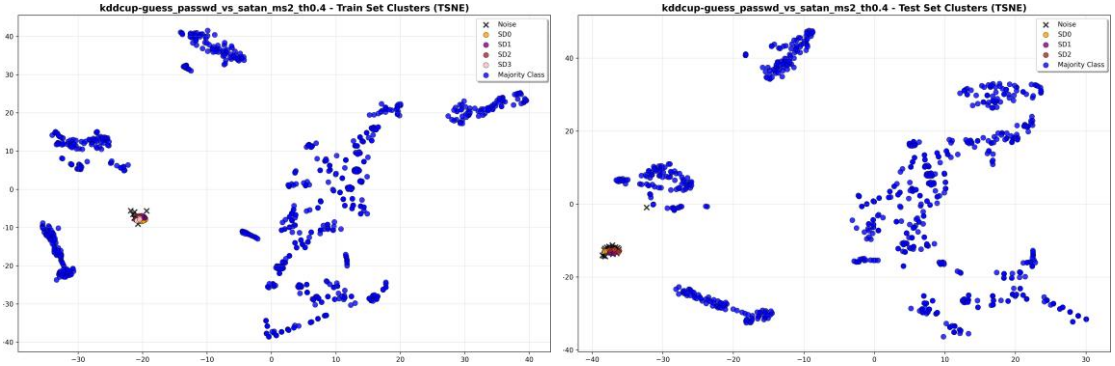
(a) *car-good* train dataset split with SCV. (b) *car-good* test dataset split with SCV.

Figure 5.1: *car-good* split (t-SNE projections of train vs. test set) with SCV.



(a) *car-good* train dataset split with DOB. (b) *car-good* test dataset split with DOB.

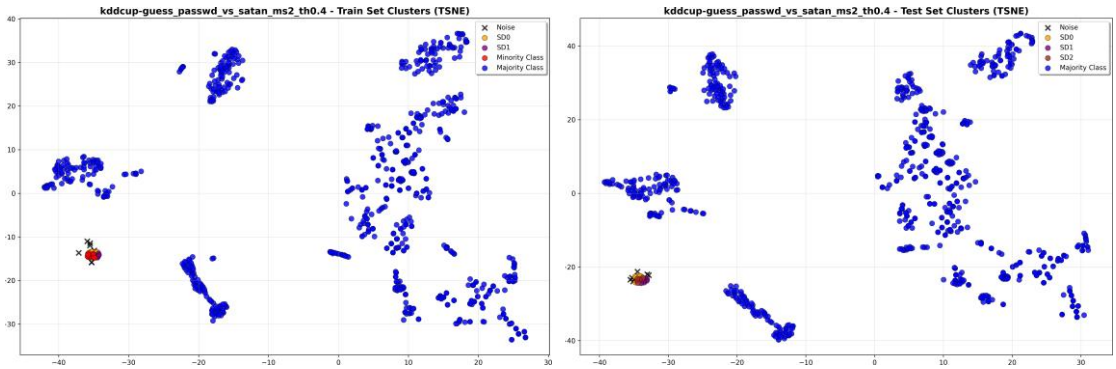
Figure 5.2: *car-good* split (t-SNE projections of train vs. test set) with DOB.



(a) *kddcup-guess_passwd_vs_satan* train dataset split with SCV.

(b) *kddcup-guess_passwd_vs_satan* test dataset split with SCV.

Figure 5.3: *kddcup-guess_passwd_vs_satan* split (t-SNE projections of train vs. test set) with SCV.



(a) *kddcup-guess_passwd_vs_satan* train dataset split with DOB.

(b) *kddcup-guess_passwd_vs_satan* test dataset split with DOB.

Figure 5.4: *kddcup-guess_passwd_vs_satan* (t-SNE projections of train vs. test set) split with DOB.

5.2. Overall Results

As discussed previously, we introduced four metrics for quantifying small disjuncts, grounded in the existing literature. From these, three were retained for systematic evaluation: *Disjunct Error Rate* (DER), *Error Coverage* (EC) and *Correct Coverage* (CC). In addition, we derived a composite metric, denoted as the *Disjunct Score* (DS), defined as:

$$DS = \frac{(1 - \text{DER}) + (1 - \text{EC}) + \text{CC}}{3}. \tag{5.1}$$

This formulation provides an aggregated perspective on disjunct-related performance. Since both DER and EC ideally should approach zero, we invert them via $(1 - \text{metric})$ before combining with CC. Consequently, higher values of DS indicate superior performance with respect to mitigating small disjuncts.

The first set of results corresponds to the average performance of each resampling strategy. Out of the total 78 benchmark datasets, IterDBSCAN identified small disjuncts in 28 datasets. Among these, all had disjuncts in the training set, whereas only 16 exhibited disjuncts in both the training and test sets. Accordingly, the disjunct-related metrics (DER, EC, CC and DS) are computed exclusively on those 16 datasets.

Table 5.1 reports the average values for the global performance metrics (recall, F1-score, precision and AUC), alongside the disjunct-specific metrics and the derived DS score.

Table 5.1: Global performance and disjunct-specific performance measures. Higher values of the Disjunct Score (DS) indicate better handling of small disjuncts. Bold values correspond to the best result for each metric.

SMOTE Method	Recall	F1	Precision	AUC	DER	EC	CC	DS
NO-SMOTE	0.498	0.454	**0.809**	0.879	0.424	0.195	0.320	0.567
CBO	0.810	0.547	0.498	0.893	0.184	0.135	0.307	0.663
Cluster-SMOTE	0.807	0.549	0.505	0.895	0.188	0.132	0.305	0.661
DBSMOTE	0.705	0.553	0.544	0.876	0.238	0.177	0.348	0.644
MWMOTE	0.821	0.555	0.499	0.894	0.196	0.150	0.309	0.655
SMOTE	**0.836**	0.544	0.487	**0.900**	0.185	0.154	0.303	0.655
SMOTE+TomekLinks	**0.836**	0.544	0.489	**0.900**	**0.165**	0.147	0.308	0.665
IterDBSCAN-SMOTE	0.762	**0.567**	0.548	0.888	0.179	**0.124**	**0.356**	**0.685**

The aggregated results reveal several key patterns. First, IterDBSCAN-SMOTE achieves competitive global performance with respect to recall, F1-score, precision and AUC, aligning closely with existing resampling strategies. However, when focusing on disjunct-related behavior, benefits emerge.

IterDBSCAN-SMOTE achieves the highest Disjunct Score ($DS = 0.685$), outperforming all other methods. In particular, it substantially reduces the proportion of errors originating from small disjuncts, as reflected by its lower Error Coverage (12.4%) compared to the NO-SMOTE baseline (19.5%). This confirms that a significant fraction of classification errors indeed arises from minority-class small disjuncts, and that our method effectively mitigates this issue. The only exception occurs with the Disjunct Error Rate, where SMOTE+TomekLinks achieves a slightly lower value. This could be attributed to the hybrid nature of Tomek Links, which combines oversampling with a data-cleaning mechanism. Nevertheless, when considering the aggregate DS metric, IterDBSCAN-SMOTE consistently demonstrates superior handling of small disjuncts relative to competing approaches.

It is also worth highlighting two additional aspects. First, IterDBSCAN-SMOTE attains the best overall F1-score (0.567), which indicates that it balances precision and recall more effectively than other methods. While the NO-SMOTE baseline reports the highest raw precision (0.809), this comes at the cost of a very low recall (0.498), ultimately yielding the worst F1-score. By contrast, IterDBSCAN-SMOTE achieves a strong trade-off between precision (0.548) and recall (0.762).

Secondly, when comparing IterDBSCAN-SMOTE to DBSMOTE - the most conceptually related methods - the improvements are consistent across all metrics, both global and disjunct-specific. This suggests that the modifications introduced in IterDBSCAN indeed

confer a tangible advantage, improving the ability to address small disjuncts without sacrificing global classification performance.

5.3. Analysis by Classifier

In addition to comparing resampling strategies, it is also informative to analyze the performance of different classifiers under the proposed pipeline. Table 5.2 summarizes the aggregate results across the selected classifiers. As before, global metrics are computed over the 28 datasets where IterDBSCAN has found small disjuncts (in the training set), while disjunct-related metrics are restricted to the 16 datasets containing small disjuncts both in their training and test partitions.

Table 5.2: Global and disjunct-specific metrics by classifier. Bold values indicate the best result for each metric and underlined values indicate the worst.

Classifier	Recall	F1	Precision	AUC	DER	EC	CC	DS
C4.5 (Decision Tree)	0.645	0.582	0.570	0.797	0.358	0.235	0.316	0.574
KNN	0.742	0.588	0.570	0.892	0.199	0.137	**0.352**	0.672
Linear SVM	0.796	0.547	0.550	0.922	**0.174**	0.125	0.330	**0.677**
Logistic Regression	0.786	0.528	0.535	0.925	0.194	0.151	0.313	0.656
MLP	0.801	**0.591**	0.587	**0.932**	0.179	0.151	0.325	0.665
Naive Bayes	**0.814**	0.370	0.383	0.839	0.203	0.147	0.286	0.645
RBF SVM	0.732	0.569	**0.637**	0.926	0.233	**0.115**	0.315	0.656

Several interesting patterns emerge from this comparison. First, C4.5 Decision Trees consistently present the weakest performance across both global and disjunct-oriented metrics. They achieve the lowest recall and AUC, as well as the lowest DER, EC and DS. This result aligns with our observations from the literature review (Section 2) where rule-based classifiers were found to be highly affected by the problem of small disjuncts. To some extent, these results provide confidence that our method, IterDBSCAN, is effective in identifying these critical regions of the input space.

Support Vector Machines, both linear and RBF-based, perform strongly overall, especially in terms of recall and AUC. In contrast, Naïve Bayes shows an odd pattern: it achieves the highest average recall (0.814) but at the cost of a very low precision (0.383). This suggests that Naïve Bayes is biased towards capturing minority instances, producing a high number of false positives, essentially overestimating the minority class. Consequently, its performance is closer to random guessing than to meaningful classification, as indicated by its low F1-score (0.569).

The purpose of this classifier-level analysis was twofold: to gain insight into the interaction between our identification method and different learners, and to select a base model for subsequent analyses. Based on the obtained results, the C4.5 Decision Tree was chosen:

beyond being widely regarded as an interpretable model, frequently used in studies of imbalanced data and small disjuncts, it also yielded the weakest results in terms of recall and AUC. In this context, we can assess whether resampling methods have the ability to improve performance, by changing only the training data, rather than modifying the learning paradigm itself.

5.4. C4.5 Decision Tree Footprints

Drilling down to the intended classifier, the C4.5 Decision Tree, Table 5.3 shows, for each dataset, the method (IterDBSCAN-SMOTE, Tomek Links, CBO, etc.) that achieved the best *Disjunct Score* (DS).

Several observations are worth noting. First, a high disjunct score does not necessarily translate into strong global predictive performance - for instance, the *yeast-1_vs_7* dataset exhibits a relatively high disjunct score while simultaneously achieving very low recall and F1 (0.067 and 0.050, respectively).

Second, our method generally produces a more balanced trade-off between recall and precision (and consequently F1) than competing approaches. In contrast, methods such as DBSMOTE and SMOTE+TomekLinks often achieve high recall at the expense of substantially lower precision, reflecting an increase in false positives. This pattern suggests that IterDBSCAN-SMOTE applies a more conservative oversampling strategy, likely because it preserves local structure and maintains the original proportion between concepts and disjuncts.

Finally, given that C4.5 was the weakest classifier overall in our experiments, it is particularly noteworthy that IterDBSCAN-SMOTE improved disjunct-related performance in half of the datasets considered.

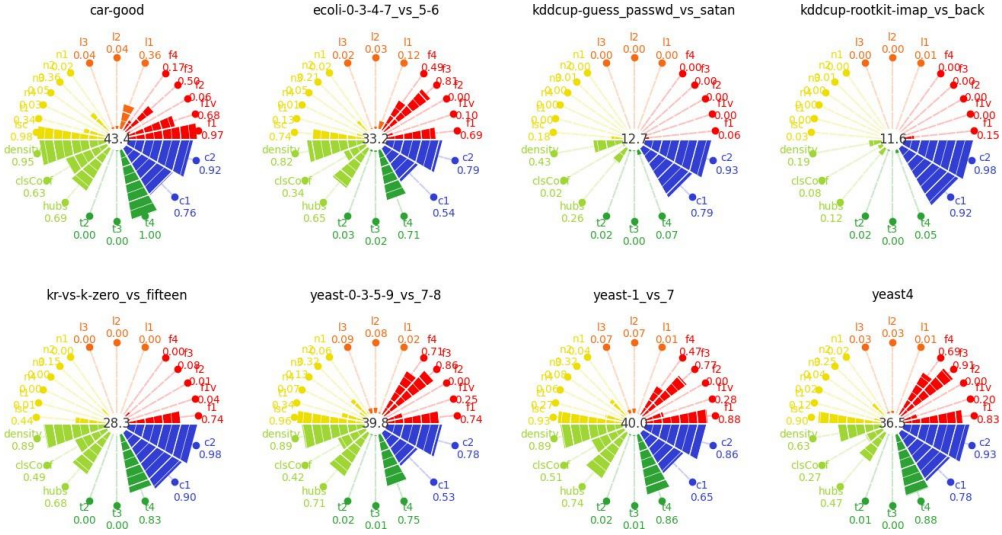
Table 5.3: Combined performance and disjunct-related results of C4.5 Decision Tree with the best *DS* method per dataset.

Dataset	SMOTE Method	Recall	F1	Precision	AUC	DER	EC	CC	DS	MPC*
abalone-17_vs_7-8-9-10	DBSMOTE	0.786	0.216	0.125	0.825	0.154	0.333	0.500	0.671	0.464
car-good	IterDBSCAN-SMOTE	0.576	0.655	0.760	0.784	0.222	0.143	0.368	0.668	0.273
ecoli-0-1-4-7_vs_5-6	NO_SMOTE	0.667	0.593	0.533	0.811	0.000	0.000	0.375	0.792	0.250
ecoli-0-3-4-7_vs_5-6	IterDBSCAN-SMOTE	0.750	0.643	0.563	0.845	0.333	0.333	0.222	0.519	0.250
kddcup-guess_passwd_vs_satan	IterDBSCAN-SMOTE	1.000	1.000	1.000	1.000	0.000	0.000	0.720	0.907	0.720
kddcup-rootkit-imap_vs_back	IterDBSCAN-SMOTE	1.000	1.000	1.000	1.000	0.000	0.000	0.364	0.788	0.364
kr-vs-k-three_vs_eleven	MWMOTE	0.923	0.947	0.973	0.961	0.000	0.000	0.528	0.843	0.487
kr-vs-k-zero_vs_fifteen	IterDBSCAN-SMOTE	1.000	1.000	1.000	1.000	0.000	0.000	0.250	0.750	0.250
vowel0	CBO	0.933	0.816	0.724	0.949	0.067	1.000	1.000	0.644	1.000
yeast-0-2-5-6_vs_3-7-8-9	SMOTE_TomekLinks	0.776	0.188	0.107	0.537	0.100	0.091	0.237	0.682	0.204
yeast-0-2-5-7-9_vs_3-6-8	NO_SMOTE	0.694	0.618	0.557	0.817	0.000	0.000	0.088	0.696	0.061
yeast-0-3-5-9_vs_7-8	IterDBSCAN-SMOTE	0.280	0.226	0.189	0.574	0.000	0.000	0.714	0.905	0.200
yeast-0-5-6-7-9_vs_4	NO_SMOTE	0.480	0.407	0.353	0.694	0.000	0.000	0.250	0.750	0.120
yeast-1-4-5-8_vs_7	NO_SMOTE	0.267	0.205	0.167	0.603	0.333	0.091	0.500	0.692	0.200
yeast-1_vs_7	IterDBSCAN-SMOTE	0.067	0.050	0.040	0.477	0.667	0.143	1.000	0.730	0.200
yeast4	IterDBSCAN-SMOTE	0.440	0.379	0.333	0.705	0.000	0.000	0.273	0.758	0.120

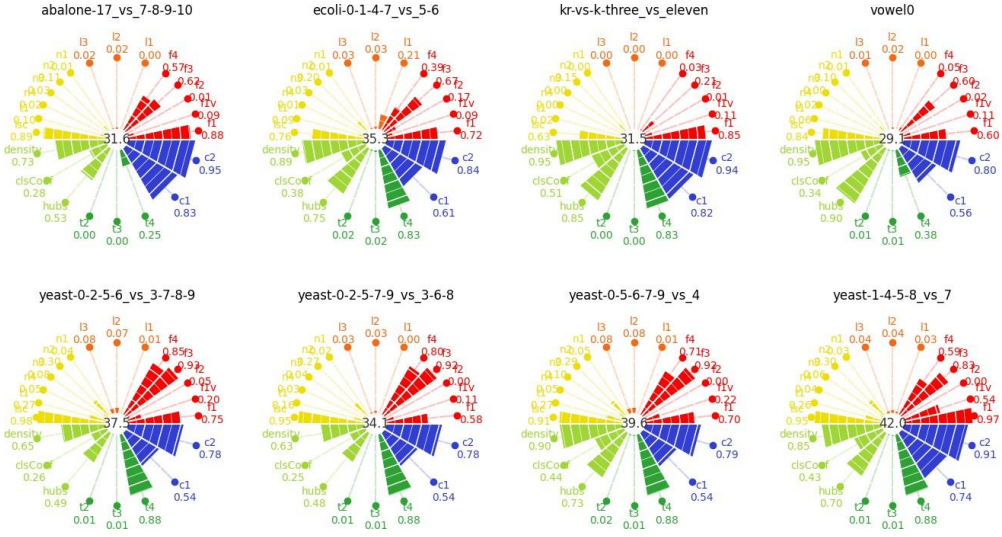
*min points covered.

5.5. Data Complexity Analysis

This section analyses our resampling methods' behaviour regarding the C4.5 Decision Tree by analysing the complexity metrics introduced in Sections 2 and 3, taken for each considered dataset in Table 5.3. Figure 5.5a shows the set of metrics (as produced by the problexity library) for the datasets where IterDBSCAN-SMOTE attained the best performance, while Figure 5.5b shows the same metrics for datasets where IterDBSCAN-SMOTE was surpassed by another method. From these plots we can draw three main observations.



(a) IterDBSCAN-SMOTE was superior.



(b) IterDBSCAN-SMOTE was not superior.

Figure 5.5: Complexity metrics taken from benchmark datasets where (a) IterDBSCAN-SMOTE was the top performer and (b) IterDBSCAN-SMOTE was surpassed by another method.

1. **Class imbalance (C1, C2).** The class imbalance metrics appear to influence the behaviour of IterDBSCAN-SMOTE. On average, the mean value of C1/C2 for datasets where IterDBSCAN-SMOTE is superior is 0.82, whereas it drops to 0.75 for datasets where it is not the best. Of the 8 datasets in which IterDBSCAN-SMOTE performed best, 5 have either C1 or C2 > 0.9, and 2 have both C1 and C2 > 0.9. In contrast, among the other 8 datasets where IterDBSCAN-SMOTE was surpassed, only 3 have either C1 or C2 > 0.9, and none has both metrics above 0.9. This pattern suggests that IterDBSCAN-SMOTE is suitable to work under strong class imbalance.

2. **Class overlap (Feature-based measures and Hub Score).** IterDBSCAN-SMOTE tends to struggle when the feature-based measures (the F-related metrics shown in red in Figure 5.5a) and the Hub Score (”hubs”) are relatively high. This indicates that datasets on which IterDBSCAN-SMOTE underperforms typically exhibit higher values of class overlap, which is consistent with the algorithm’s focus on modelling minority-class structure independently of the majority class.

3. **Class decomposition (LSC, Density, Clustering Coefficient).** Metrics related to class decomposition - such as LSC, Density and Clustering Coefficient (ClsCoef) - are consistently higher for datasets where IterDBSCAN-SMOTE achieves superior performance. This indicates that the method is most effective when the minority class is decomposed into several well-defined, small disjuncts, since IterDBSCAN explicitly targets that type of local structure.

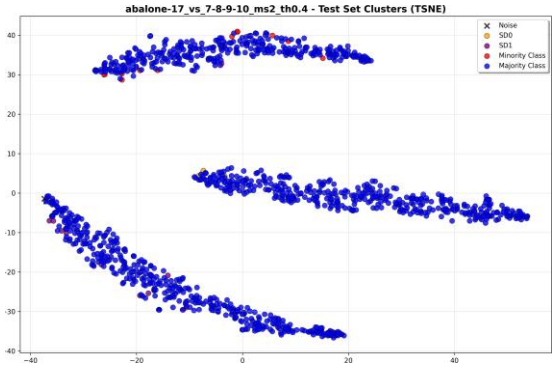
Taken together, these observations indicate that IterDBSCAN-SMOTE is most effective in datasets that combine pronounced class decomposition and structurally complex minority instances with relatively low class overlap. This behaviour is expected: IterDBSCAN-SMOTE explicitly preserves and exploits minority-class local structure when generating synthetic instances but it does not explicitly account for the majority class distribution in the input space. Consequently, in datasets with substantial overlap between classes, oversampling alone can lead to increased false positives; methods that perform explicit cleaning of the majority class boundary can therefore complement oversampling by mitigating overlap which helps explain why those methods (e.g., SMOTE+TomekLinks) tend to also achieve good results.

More broadly, these findings are consistent with the conclusions established in the literature: small disjuncts are inherently difficult to work with due to their low cardinality and

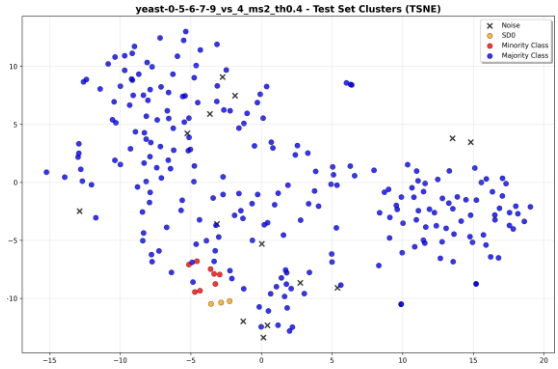
the presence of additional sources of complexity - such as the class overlap in this case - exacerbates this difficulty. When few minority instances exist and those instances are also entangled in overlap regions, the problem becomes exponentially harder, since the already limited evidence for the minority class is simultaneously affected by conflicting class boundaries.

5.6. Discussion and Conclusions

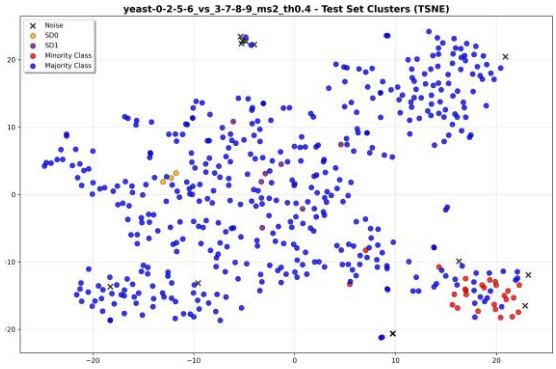
One additional perspective worth discussing is the analysis of situations where IterDBSCAN-SMOTE was superior and compare them with cases where it was not. Looking first at scenarios where IterDBSCAN-SMOTE underperformed (Figures 5.6a, 5.6b and 5.6c), we typically observe: (i) strong overlap between feature values (Figure 5.6a), (ii) a high prevalence of noisy instances (Figure 5.6b), or (iii) a combination of both factors (Figure 5.6c).



(a) *abalone-17_vs_7-8-9-10*



(b) *yeast-0-5-6-7-9_vs_4*

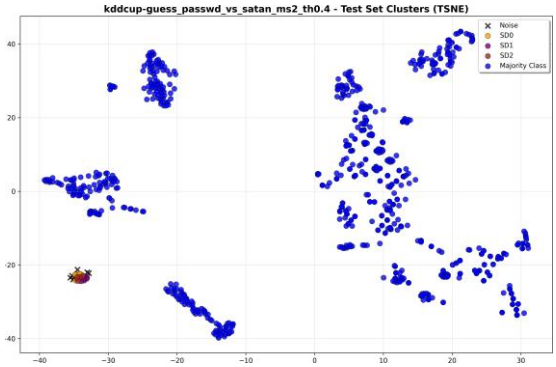


(c) *yeast-0-2-5-6_vs_3-7-8-9*

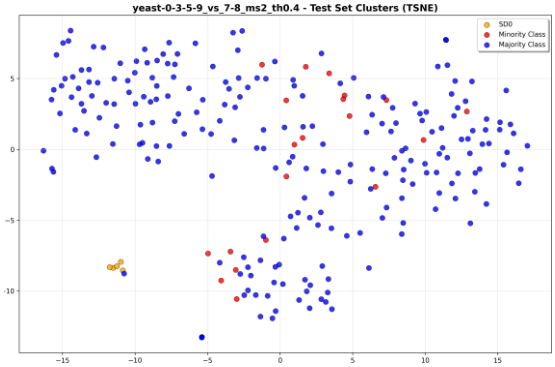
Figure 5.6: Datasets (t-SNE projections of test sets) where IterDBSCAN-SMOTE was surpassed, showing overlapping features and noisy instances.

In contrast, datasets where IterDBSCAN-SMOTE performed well generally exhibit compact disjuncts (Figure 5.7a), even when these disjuncts seem to exhibit somewhat different characteristics than the main minority concept (Figures 5.7b, 5.7c and 5.7d). These datasets also tend to show less class overlap and fewer noisy instances, although such

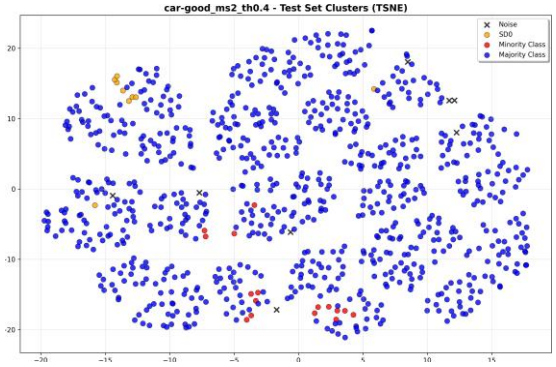
issues may still occur occasionally - on *car-good* and *yeast-0-3-5-9_vs_7-8* - overlap can be found, and on *car-good* noisy instances can also be seen.



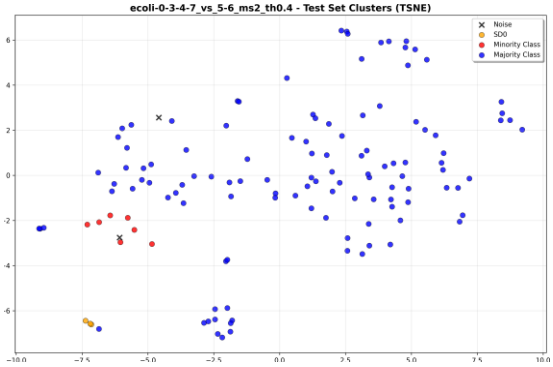
(a) *kddcup-guess_passwd_vs_satan* - Compact disjuncts.



(b) *yeast-0-3-5-9_vs_7-8* - Compact disjuncts.



(c) *car-good* - Compact disjuncts.



(d) *ecoli-0-3-4-7_vs_5-6* - Compact disjuncts.

Figure 5.7: Datasets (t-SNE projections of test sets) where IterDBSCAN-SMOTE was superior, showing compact disjuncts with lower degrees of class overlap and noisy instances.

A complementary analysis can be made by inspecting the t-SNE visualizations of the full datasets (comparing best- vs. worst-performing cases) and evaluating class complexity. Here, we used the N3 measure, which quantifies how often instances are misclassified by their nearest neighbors (considering both classes). Unlike the earlier global complexity analysis (Figure 5.5), this instance-level view highlights where errors are concentrated.

On datasets where IterDBSCAN-SMOTE performs well (Figure 5.8) the majority class is mostly composed of “safe” instances, while classification errors are concentrated in the minority class. Conversely, in datasets where IterDBSCAN performs poorly (Figure 5.9) the classes appear more intertwined, showing larger overlap regions throughout the dataset and there are more majority class instances flagged as complex (red balls). These majority-class complexities are not easily handled by IterDBSCAN-SMOTE, which helps explain its weaker performance.

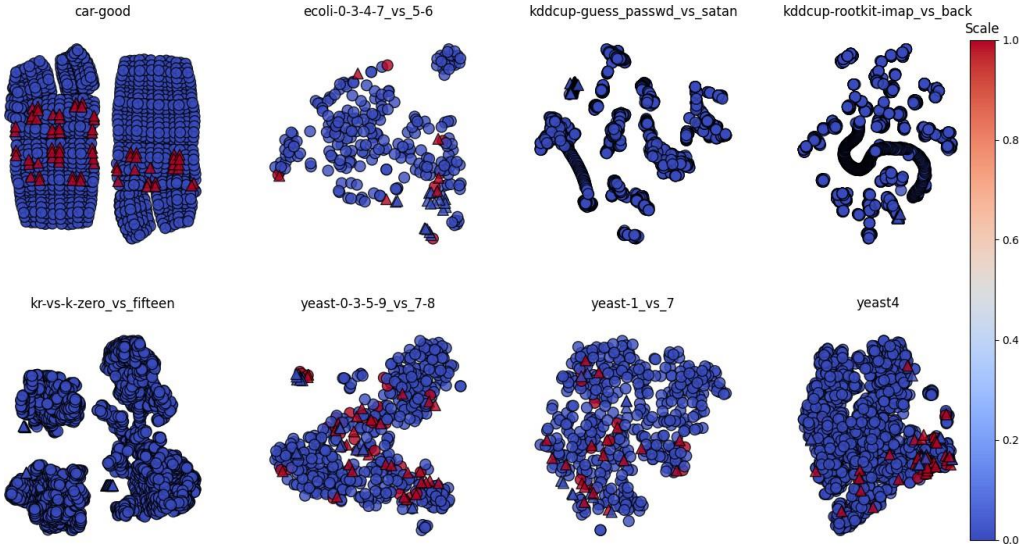


Figure 5.8: Datasets (t-SNE projections) where IterDBSCAN-SMOTE was superior. Individual examples are highlighted according to their complexity (N3 measure).

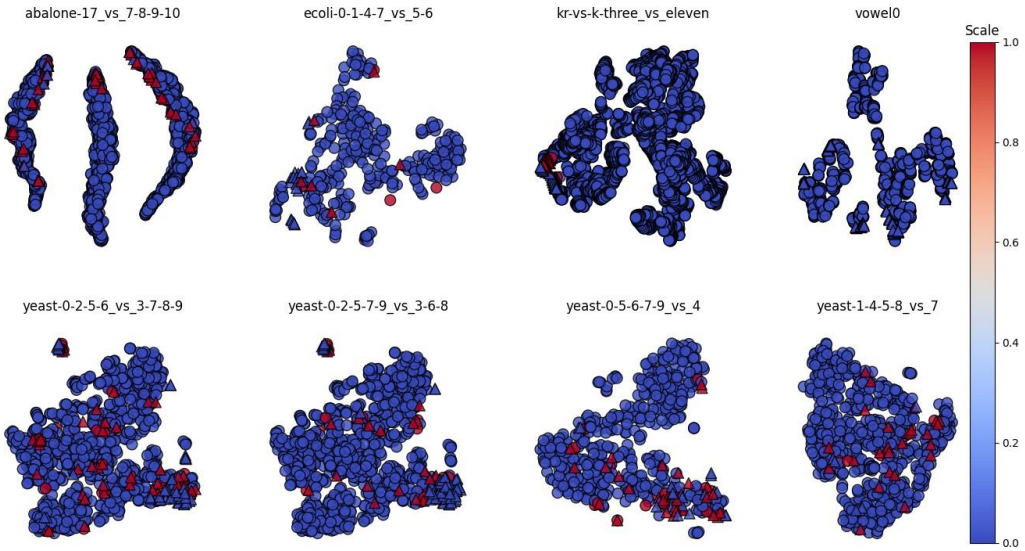


Figure 5.9: Datasets (t-SNE projections) where IterDBSCAN-SMOTE was surpassed. Individual examples are highlighted according to their complexity (N3 measure).

Taken together, these findings complement the earlier results, reinforcing the idea that IterDBSCAN-SMOTE is particularly effective in scenarios with compact, well-structured disjuncts and limited class overlap, but less effective when the data is dominated by noise and overlapping distributions.

Overall, the proposed method demonstrates a consistent ability to identify small disjuncts, suggesting its potential to be further adapted into a complexity metric. For instance, one possible direction would be to quantify the cumulative error within disjuncts for a single classifier, such as a Decision Tree, or for an ensemble of classifiers, thus creating a new measure of dataset complexity.

Additionally, the IterDBSCAN-SMOTE algorithm proved effective in amplifying the representation of small disjuncts, thereby improving classification performance on these critical regions. Nevertheless, challenges remain in addressing the problem of class overlap and incorporating a data-cleaning strategy into the method may help mitigate this limitation.

IterDBSCAN-SMOTE also shows particular promise in contexts of *extreme* class imbalance, where identifying and handling small disjuncts is especially difficult. This opens avenues for application in domains characterized by rare events, such as fraud detection or similar anomaly-driven problems.

Another important observation is robustness: whereas many oversampling methods tend to improve recall at the expense of precision, IterDBSCAN-SMOTE maintained a more balanced behavior, classifying minority points without generating excessive false positives.

In summary, despite its limitations, the method consistently outperforms alternative approaches with respect to the specific challenge of mitigating small disjuncts, as confirmed by the disjunct-specific metrics.

6. Conclusions and Future Work

The goal of this work was straightforward in its definition, though not in its execution: to develop a strategy for identifying and quantifying small disjuncts and subsequently create a complementary approach for mitigating this problem. To achieve these objectives, we conducted a comprehensive investigation that included an in-depth review of the problem and the current state-of-the-art approaches, an extensive experimentation with existing methods, and the development of our novel IterDBSCAN algorithm. Building upon these foundations, we then developed a mitigation strategy to address the second goal, IterDBSCAN-SMOTE.

To validate our approach, we constructed a robust experimental framework encompassing over 70 datasets, 2 split ratios, 2 split strategies, 6 combinations of minimum points and disjunct thresholds, 8 resampling options and 7 classifiers - resulting in more than 100,000 experimental runs. While only a fraction of these proved relevant to our analysis, this comprehensive setup enabled us to conduct an in-depth examination of results and dataset structures.

Our findings demonstrate positive outcomes across multiple dimensions. The IterDBSCAN method successfully identifies disjuncts and, more importantly, its combination with a simple strategy such as SMOTE improves their representation, leading to enhanced classification performance - particularly valuable in contexts of extreme class imbalance. The approach consistently maintains competitive performance on global metrics while demonstrating superior performance on disjunct-specific metrics. Notably, for the weakest performing classifier, we achieved improvements in 50% of the classification problems. Furthermore, the method exhibits robustness in managing the trade-off between global and local performance metrics.

Limitations. Despite these positive results, several limitations warrant consideration:

Algorithm scalability and efficiency. The iterative nature of IterDBSCAN, requiring repeated metric computations, presents scalability challenges for very large datasets. While this computational cost is inherent to the design, optimization opportunities exist. For instance, calculating the growth ratio based on expansion within the current iteration, rather than relative to the previous one, could improve efficiency.

Parameter selection and adaptation. The current point removal process, while reasonable in its use of saturation and data locality, could benefit from incorporating positional awareness. Specifically, considering a point's relative position within and between clusters could inform better removal decisions. Points far from other clusters might not warrant removal unless cluster pruning is explicitly desired. This positional information could also guide automatic parameter adjustment, potentially eliminating manual parameter selection. Moreover, given the varying density and shape of substructures within datasets, locally adaptive parameters may prove more effective than global ones.

Methodological constraints. The current implementation relies exclusively on Euclidean distance and requires prior feature encodings, for example, one-hot encoding for nominal features. Additionally, the weighting scheme in the optimization algorithm requires manual definition (in case the defaults are not selected) whereas a formal optimization procedure could ensure statistically or algorithmically optimal weights for each dataset.

Future Work. Several promising research directions emerge from this work:

Algorithm enhancements. These refer to addressing the liminations identified above, namely incorporting alternative distance measures beyond Euclidean distance to improve flexibility and performance across diverse domains, developing methods that directly accommodate categorical data without requiring prior encoding transformations and implementing formal optimization procedures for weight determination to ensure dataset-specific optimal configurations.

Oversampling strategy improvements. Given the characteristics of datasets that lead to good and poor behaviour of IterDBSCAN-SMOTE, future work could focus on exploring alternative SMOTE variants and synthetic data generation approaches to better balance small disjuncts while preserving structural integrity and investigating cleaning methods such as Tomek Links, which showed a powerful performance in our results and could help address class overlap issues and noisy points issues.

Novel data complexity metrics: The insights derived from the proposed disjunct-specific metrics could serve as a steppingstone to develop more suitable methods to quantify the impact of small disjuncts, considering single classifiers (e.g., decision trees or ensemble methods). Another direction would be to use IterDBSCAN to determine the number of existing small disjuncts, and complementing the analysis with additional metrics such as N3 or the data typology introduced in Section 2.2 (safe/borderline/rare/outlier) to help

characterise these specific points.

Meta-Analysis and Classification Footprints: Another direction for future research could be to perform an interpretable meta-analyses of regions of good and poor behaviour of standard classifiers. This would entail producing a meta-dataset for each classifier, where each observation is a dataset and the features are its data complexity measures. Then, we would associate each dataset to the best performing method per classifier. In this way, we could link global dataset characteristics to the methods that showed the best performance, thus deriving some recommendations on the best approach for each domain. This analysis could be further supported by interpretable strategies, such as SHAP or LIME [38, 39], to further explain particular associations between complexity and performance.

Application-oriented studies. Two case studies would provide valuable insights:

1. **Fairness analysis:** Investigating the relationship between small disjuncts and algorithmic fairness, as underrepresented sub-concepts may correspond to minority subgroups in real-world scenarios. Beyond theoretical analysis, applying our proposed algorithm in such contexts could provide practical insights and help assess its potential for mitigating fairness-related risks in predictive modeling.
2. **Fraud detection:** Exploring the methodology in high-stakes domains such as bank fraud detection, where small disjuncts often represent critical but rare patterns.

Overall, while some limitations remain and further refinements are needed, this thesis has tackled a difficult yet meaningful challenge, contributed significantly to the study of small disjuncts.

References

[1] D. Zha, Z. P. Bhat, K.-H. Lai, F. Yang, Z. Jiang, S. Zhong, and X. Hu, "Data-centric artificial intelligence: A survey," *ACM Comput. Surv.*, vol. 57, no. 5, Jan. 2025. [Online]. Available: https://doi.org/10.1145/3711118 [Cited on page 1.]

[2] F. Clemente, G. M. Ribeiro, A. Quemy, M. S. Santos, R. C. Pereira, and A. Barros, "ydata-profiling: Accelerating data-centric ai with high-quality data," *Neurocomputing*, vol. 554, p. 126585, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231223007087 [Cited on page 1.]

[3] J. Jakubik, M. Vössing, N. Kühl, J. Walk, and G. Satzger, "Data-centric artificial intelligence," *Business & Information Systems Engineering*, vol. 66, no. 4, pp. 507–515, 2024. [Online]. Available: https://doi.org/10.1007/s12599-024-00857-8 [Cited on page 1.]

[4] S. Das, S. Datta, and B. B. Chaudhuri, "Handling data irregularities in classification: Foundations, trends, and future challenges," *Pattern Recognition*, vol. 81, pp. 674–693, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320318300931 [Cited on pages 1 and 7.]

[5] M. R. S. P. S. Santos, "Research problems in data quality: Addressing imbalanced and missing data," Ph.D. dissertation, Universidade de Coimbra (Portugal), 2022. [Cited on pages 1, 18, and 40.]

[6] E. Alpaydın, *Machine Learning*. The MIT Press, 08 2021. [Online]. Available: https://doi.org/10.7551/mitpress/13811.001.0001 [Cited on page 1.]

[7] V. Werner de Vargas, J. A. Schneider Aranda, R. dos Santos Costa, P. R. da Silva Pereira, and J. L. Victória Barbosa, "Imbalanced data preprocessing techniques for machine learning: a systematic mapping study," *Knowledge and Information Systems*, vol. 65, no. 1, pp. 31–57, 2023. [Online]. Available: https://doi.org/10.1007/s10115-022-01772-8 [Cited on pages 1, 7, and 9.]

[8] W. Chen, K. Yang, Z. Yu, Y. Shi, and C. L. P. Chen, "A survey on imbalanced learning: latest research, applications and future directions," *Artificial Intelligence Review*, vol. 57, no. 6, p. 137, 2024. [Online]. Available: https://doi.org/10.1007/s10462-024-10759-6 [Cited on pages 1, 7, and 9.]

[9] N. Japkowicz, “Concept-learning in the presence of between-class and within-class imbalances,” in *Advances in Artificial Intelligence*, E. Stroulia and S. Matwin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 67–77. [Cited on pages 2 and 17.]

[10] V. López, A. Fernández, S. García, V. Palade, and F. Herrera, “An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics,” *Information Sciences*, vol. 250, pp. 113–141, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025513005124 [Cited on pages 2, 8, 9, 10, 11, 12, 13, and 46.]

[11] A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk, and F. Herrera, *Data Intrinsic Characteristics*. Cham: Springer International Publishing, 2018, pp. 253–277. [Online]. Available: https://doi.org/10.1007/978-3-319-98074-4_10 [Cited on pages 2 and 10.]

[12] A. Lorena, L. P. Garcia, J. Lehmann, M. de Souto, and T. Ho, “How complex is your classification problem?: A survey on measuring classification complexity,” *ACM Computing Surveys*, vol. 52, pp. 1–34, 09 2019. [Cited on pages 5, 14, 15, and 22.]

[13] G. Batista, A. Bazzan, and M.-C. Monard, “Balancing training data for automated annotation of keywords: a case study.” 01 2003, pp. 10–18. [Cited on pages 5 and 9.]

[14] S. Barua, M. M. Islam, X. Yao, and K. Murase, “Mwmote–majority weighted minority oversampling technique for imbalanced data set learning,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 2, pp. 405–425, 2014. [Cited on pages 5 and 18.]

[15] T. Jo and N. Japkowicz, “Class imbalances versus small disjuncts,” *SIGKDD Explor. Newsl.*, vol. 6, no. 1, p. 40–49, Jun. 2004. [Online]. Available: https://doi.org/10.1145/1007730.1007737 [Cited on pages 5 and 18.]

[16] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing, “Learning from class-imbalanced data: Review of methods and applications,” *Expert Systems with Applications*, vol. 73, pp. 220–239, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417416307175 [Cited on pages 7, 8, and 9.]

[17] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap, “Dbsmote: Density-based synthetic minority over-sampling technique,” *Applied Intelligence*, vol. 36, no. 3, pp. 664–684, 2012. [Online]. Available: https://doi.org/10.1007/s10489-011-0287-y [Cited on page 9.]

[18] K. Veropoulos, C. Campbell, and N. Cristianini, “Controlling the sensitivity of support vector machines,” *Proceedings of International Joint Conference Artificial Intelligence*, 06 1999. [Cited on page 10.]

[19] M. S. Santos, P. H. Abreu, N. Japkowicz, A. Fernández, C. Soares, S. Wilk, and J. Santos, “On the joint-effect of class imbalance and overlap: a critical review,” *Artificial Intelligence Review*, vol. 55, no. 8, pp. 6207–6275, 2022. [Online]. Available: https://doi.org/10.1007/s10462-022-10150-3 [Cited on page 10.]

[20] T. Ho and M. Basu, “Complexity measures of supervised classification problems,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, pp. 289–300, 03 2002. [Cited on page 13.]

[21] J. Sotoca, J. Sánchez, and R. Mollineda, “A review of data complexity measures and their applicability to pattern classification problems,” *Actas del III Taller Nacional de Mineria de Datos y Aprendizaje*, 01 2005. [Cited on page 14.]

[22] K. Napierala and J. Stefanowski, “Types of minority class examples and their influence on learning classifiers from imbalanced data,” *Journal of Intelligent Information Systems*, vol. 46, no. 3, pp. 563–597, 2016. [Online]. Available: https://doi.org/10.1007/s10844-015-0368-1 [Cited on page 15.]

[23] M. S. Santos, P. H. Abreu, N. Japkowicz, A. Fernández, and J. Santos, “A unifying view of class overlap and imbalance: Key concepts, multi-view panorama, and open avenues for research,” *Information Fusion*, vol. 89, pp. 228–253, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253522001099 [Cited on page 16.]

[24] A. P. Danyluk and F. J. Provost, “Small disjuncts in action: Learning to diagnose errors in the local loop of the telephone network,” in *Machine Learning Proceedings 1993*. San Francisco (CA): Morgan Kaufmann, 1993, pp. 81–88. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781558603073500174 [Cited on page 17.]

[25] G. M. Weiss, “Learning with rare cases and small disjuncts,” in *Proceedings of the Twelfth International Conference on International Conference on Machine Learning*, ser. ICML’95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, p. 558–565. [Cited on page 17.]

[26] G. Weiss and H. Hirsh, “The problem with noise and small disjuncts,” 09 1998. [Cited on page 17.]

[27] G. Weiss, “A quantitative study of small disjuncts,” 04 2000. [Cited on pages 17 and 30.]

[28] N. Japkowicz and S. Stephen, “The class imbalance problem: A systematic study,” *Intell. Data Anal.*, vol. 6, no. 5, p. 429–449, Oct. 2002. [Cited on page 17.]

[29] G. Kovács, “Smote-variants: A python implementation of 85 minority oversampling techniques,” *Neurocomputing*, vol. 366, pp. 352–354, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231219311622 [Cited on pages 18 and 25.]

[30] R. C. Holte, L. E. Acker, and B. W. Porter, “Concept learning and the problem of small disjuncts,” in *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1*, ser. IJCAI’89. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, p. 813–818. [Cited on page 20.]

[31] J. Komorniczak and P. Ksieniewicz, “problexity—an open-source python library for supervised learning problem complexity assessment,” *Neurocomputing*, vol. 521, pp. 126–136, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231222014461 [Cited on page 22.]

[32] J. G. Moreno-Torres, J. A. Saez, and F. Herrera, “Study on the impact of partition-induced dataset shift on k-fold cross-validation,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 8, pp. 1304–1312, 2012. [Cited on pages 22 and 25.]

[33] G. M. Weiss, “The effect of small disjuncts and class distribution on decision tree learning,” Ph.D. dissertation, New Brunswick Rutgers, The State University of New Jersey, 2003. [Cited on pages 29 and 30.]

[34] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in *Proceedings of the Second*

International Conference on Knowledge Discovery and Data Mining, ser. KDD'96. AAAI Press, 1996, p. 226–231. [Cited on page 32.]

[35] S. Wojciechowski and S. Wilk, "Difficulty factors and preprocessing in imbalanced data sets: An experimental study on artificial data," *Foundations of Computing and Decision Sciences*, vol. 42, no. 2, pp. 149–176, 2017. [Online]. Available: https://doi.org/10.1515/fcds-2017-0007 [Cited on page 46.]

[36] V. García, J. Sánchez, H. Ochoa, and L. Cleofas, "Dissimilarity-based learning from imbalanced data with small disjuncts and noise," vol. 9117, 06 2015. [Cited on page 46.]

[37] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008. [Online]. Available: http://jmlr.org/papers/v9/vandermaaten08a.html [Cited on page 55.]

[38] S. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," 2017. [Online]. Available: https://arxiv.org/abs/1705.07874 [Cited on page 69.]

[39] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?": Explaining the predictions of any classifier," 2016. [Online]. Available: https://arxiv.org/abs/1602.04938 [Cited on page 69.]