Adaptive Exploration of Intrinsic Data Properties for Clustering, Outlier Detection, and Dimensionality Reduction

Dissertation zur Erlangung des Doktorgrades

an der Fakultät für Mathematik, Informatik und Statistik

der Ludwig-Maximilians-Universität München

vorgelegt von

Li Qian

aus Chongqing, China

München, den 24.02.2025

Erstgutachter:	Prof. Dr. Christian Böhm
Zweitgutachter:	Prof. Dr. Ye Zhu
Vorsitz:	Prof. Dr. Albrecht Schmidt
Tag der Disputation:	27.05.2025

Eidesstattliche Versicherung

(siehe Promotionsordnung vom 12.07.2011, § 8, Abs. 2 Pkt. 5)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

München, den 24.02.2025

Li Qian

Acknowledgments

During my doctoral studies, I experienced both the challenges and unexpected opportunities brought by the COVID-19 pandemic. My academic and personal journeys in Germany, China, and Austria have profoundly shaped my life, leaving me with unforgettable memories that are both sweet and bitter. I want to express my deepest gratitude to everyone who has supported, guided, and encouraged me during this period.

First and foremost, I would like to express my sincere gratitude to my supervisor, Prof. Dr. Christian Böhm, who guided me into the field of data mining. His profound knowledge, rigorous scientific spirit, and patience have greatly benefited me. Under his supervision, I had the freedom to explore and received valuable guidance and full support that enabled me to complete my doctoral studies.

I am also deeply grateful to Prof. Dr. Claudia Plant, who treated academic work with a rigorous and responsible attitude. Her insightful advice for revision was invaluable during the critical stage of paper submission. In particular, her support and care during my research visit to the University of Vienna made me feel warm and further broadened my research horizons.

I would like to thank my doctoral examination committee members, Prof. Dr. Ye Zhu, Prof. Dr. Albrecht Schmidt, and Prof. Dr. Caroline Friedel, for their precious time and effort, as well as their constructive feedback and suggestions, which have helped me improve the quality of my thesis.

I am grateful to Prof. Dr. Wei Ye and Prof. Dr. Yllka Velaj for their insightful analysis and patient guidance whenever I encountered academic confusion. I also want to extend

my appreciation to my collaborators, Prof. Dr. Xin Sun, Dr. Yalan Qin, and Wengang Guo, for their suggestions and contributions throughout the writing and research process.

I wish to thank my colleagues at the Ludwig Maximilian University of Munich and the University of Vienna, in particular Dr. Dominik Mautz, Dr. Collin Leiber, Peiyan Li, Dr. Martin Perdacher, Dr. Lukas Miklautz, and Ylli Sadikaj. The discussions with them not only enriched my knowledge and provided inspiration but also helped me gain a deeper understanding of the local culture and academic environment. Additionally, I would like to thank Susanne Grienberger, Ulrike Robeck, Tugba Yilmaz, Ewald Hotop, and Tanja Schwind for their dedicated support with technical and administrative tasks.

I am profoundly grateful to my family. My husband, Jing Qian, not only collaborated with me in research and offered unexpected perspectives for solving technical problems but also provided endless understanding, patience, and love. I am deeply grateful to my parents for their full support after the pandemic so that I can focus on my doctoral research without any worries. I extend my heartfelt thanks to my little son, whose bright smile is a constant source of strength for me to persevere. I also wish to thank my friend, Xizi Xiong, for her companionship and generous help, which made me feel at home even in a foreign country.

Finally, I would like to express my gratitude to the China Scholarship Council (CSC) for providing financial support, which has provided an important guarantee for my doctoral research.

Abstract

In the data-driven era, data has become a key element in supporting decision-making, scientific research, and technological innovation. As the scale and complexity of data continue to grow, it becomes increasingly important to extract actionable knowledge from it. Given this background, data mining is essential for discovering interesting patterns from large datasets, especially in unsupervised learning tasks such as clustering, outlier detection, and dimensionality reduction. However, existing methods often face critical challenges, including adapting to diverse and arbitrary data distributions, handling datasets with varying densities, and reducing dependence on dataset-specific parameters. Since each dataset exhibits unique intrinsic properties, such as local density and neighborhood relationships, this thesis focuses on developing an adaptive exploration framework for unsupervised learning by leveraging intrinsic data properties to enhance the adaptability, accuracy, and efficiency of unsupervised learning algorithms.

For the clustering task, this thesis proposes the DBADV algorithm. Existing density-based methods, such as DBSCAN, although capable of recognizing clusters of arbitrary shapes and sizes, are ineffective when dealing with density variations. DBADV calculates the local density information of each object using perplexity, which not only reflects the individual properties of each object but also describes the density distribution of clusters and finds the adaptive search range of each object by collecting information from its neighbors. In addition, a new metric is designed to obtain the mutual nearest neighbors of each object to better distinguish objects near the cluster boundaries, significantly improving the clustering accuracy in the presence of noise and outliers.

For the outlier detection task, this thesis proposes the ADOD algorithm. Existing proximity-based methods rely on either a globally fixed radius or a fixed number of neighbors as a parameter, both of which are prone to misjudgments when there is significant variation in data density. ADOD uses perplexity to calculate the local scale of each object and dynamically adjusts the neighborhood boundaries based on this scale to adapt to data with varying densities. Meanwhile, ADOD designs a density consistency score that determines the outlier score by calculating the local density difference between an object and its mutual neighbors, effectively identifying outliers that significantly deviate from their surroundings. Additionally, ADOD extends its utility in real-time applications by generalizing to unknown data through comparison with known data.

For the dimensionality reduction task, this thesis proposes the DynoGraph algorithm. DynoGraph addresses two main issues of the existing graph-based methods like t-SNE and UMAP: how to construct a good graph and how to maintain the similarity structure of high-dimensional data in low-dimensional space. DynoGraph first develops an adaptive neighborhood graph construction method that accurately captures the intrinsic geometry of the high-dimensional data. Then, for the first time, it introduces a dynamic graph modification process during dimensionality reduction, guaranteeing that the data structure in the low-dimensional space faithfully reflects the high-dimensional data. Moreover, DynoGraph sets an adaptive threshold that is automatically adjusted according to the intrinsic structure of data, thus guiding the insertion and deletion of edges. These adjustments help to update their positions in subsequent embeddings, aligning them with the high-dimensional data.

In summary, this thesis demonstrates how three adaptive algorithms can effectively leverage intrinsic data properties to adapt to arbitrary distributions, capture local density, and design self-adaptive parameters, thereby improving the performance of unsupervised learning tasks. This adaptive exploration framework not only enriches the theoretical approaches to data analysis, but also provides new perspectives and effective solutions for handling complex data in real-world scenarios.

Zusammenfassung

Im Zeitalter von datengetriebenen Technologien sind Daten zu einem Schlüsselelement geworden, um Entscheidungsfindung, wissenschaftliche Forschung und technologische Innovation zu unterstützen. Mit dem kontinuierlichen Wachstum der Datenmengen und ihrer Komplexität wird es zunehmend wichtiger, verwertbares Wissen aus den Daten zu extrahieren. Vor diesem Hintergrund spielt Data Mining eine entscheidende Rolle bei der Entdeckung interessanter Muster in großen Datensätzen, insbesondere bei Aufgaben des unüberwachten Lernens wie der Clusteranalyse, Ausreißererkennung und Dimensionsreduktion. Bestehende Methoden stehen jedoch oft vor großen Herausforderungen, insbesondere in Anbetracht der Anpassung an vielfältige und beliebige Datenverteilungen, dem Umgang mit Datensätzen unterschiedlicher Dichte und der Reduzierung der Abhängigkeit von datensatzspezifischen Parametern. Da jeder Datensatz einzigartige intrinsische Eigenschaften, wie lokale Dichte und Nachbarschaftsbeziehungen aufweist, konzentriert sich diese Dissertation darauf, ein Rahmenwerk für die adaptive Exploration im Bereich des unüberwachten Lernens zu entwickeln, das diese intrinsischen Eigenschaften nutzt, um die Anpassungsfähigkeit, Genauigkeit und Effizienz von Algorithmen im Kontext des unüberwachten Lernens zu verbessern.

Für die Clusteranalyse schlägt diese Dissertation den Algorithmus DBADV vor. Bestehende dichtebasierte Methoden wie DBSCAN, die zwar in der Lage sind, Cluster beliebiger Formen und Größen zu erkennen, sind bei der Handhabung von Dichtevariationen ineffektiv. DBADV berechnet die lokale Dichteinformation jedes Objekts mithilfe der Perplexität, die nicht nur die individuellen Eigenschaften jedes Objekts widerspiegelt,

sondern auch die Dichteverteilung der Cluster beschreibt. Dabei wird der adaptive Suchbereich jedes Objekts durch das Sammeln von Informationen seiner Nachbarn ermittelt. Zusätzlich wird eine neue Metrik entwickelt, um die gegenseitigen nächsten Nachbarn jedes Objekts zu bestimmen, was es ermöglicht, Objekte in der Nähe von Clustergrenzen besser zu unterscheiden und die Clustering-Genauigkeit in Situationen erheblich zu verbessern, in denen Rauschen und Ausreißer vorhanden sind.

Für die Ausreißererkennung schlägt diese Dissertation den Algorithmus ADOD vor. Bestehende nahebasierte Methoden verlassen sich entweder auf einen global festgelegten Radius oder eine feste Anzahl von Nachbarn als Parameter, die bei signifikanten Variationen in der Datendichte anfällig für Fehlurteile sind. ADOD verwendet Perplexität, um das lokale Ausmaß jedes Objekts zu berechnen, und ermittelt die Nachbarschaftsgrenzen dynamisch auf der Grundlage dieses Ausmaßes, um sich an Daten mit unterschiedlichen Dichten anzupassen. Darüber hinaus entwickelt ADOD eine Dichtekonsistenzbewertung, die den Ausreißerwert durch Berechnung des lokalen Dichteunterschieds zwischen einem Objekt und seinen gegenseitigen Nachbarn bestimmt und so effektiv Ausreißer identifiziert, die erheblich von ihrer Umgebung abweichen. Zusätzlich erweitert ADOD seine Anwendbarkeit in Echtzeitszenarien, indem es unbekannte Daten durch einen Vergleich mit bekannten Daten generalisiert.

Für die Dimensionsreduktion schlägt diese Dissertation den Algorithmus DynoGraph vor. DynoGraph adressiert zwei Hauptprobleme bestehender graphenbasierter Methoden wie t-SNE und UMAP: Wie werden qualitativ hochwertige Graphen konstruiert und wie werden Ähnlichkeitsstrukturen hochdimensionaler Daten in einem niedrigdimensionalen Raum beibehalten. DynoGraph entwickelt zunächst eine adaptive Methode zur Konstruktion eines Nachbarschaftsgraphen, die die intrinsische Geometrie hochdimensionaler Daten präzise erfasst. DynoGraph führt erstmals einen anschließenden dynamischen Graphmodifikationsprozess während der Dimensionsreduktion ein, der sicherstellt, dass die Datenstruktur im niedrigdimensionalen Raum die hochdimensionalen Daten originalgetreu widerspiegelt. Darüber hinaus legt DynoGraph einen adaptiven Schwellenwert fest, der automatisch entsprechend der intrinsischen Datenstruktur angepasst wird, um

das Einfügen und Löschen von Kanten zu steuern. Diese Anpassungen tragen dazu bei, die Positionen der Daten in den nachfolgenden Einbettungen zu aktualisieren und sie mit den hochdimensionalen Daten in Einklang zu bringen.

Zusammenfassend zeigt diese Dissertation, wie drei adaptive Algorithmen die intrinsischen Eigenschaften von Daten effektiv nutzen können, um sich an beliebige Verteilungen anzupassen, lokale Dichte zu erfassen und selbstadaptive Parameter zu entwerfen, wodurch die Leistung bzgl. Aufgaben des unüberwachten Lernens verbessert wird. Dieses Framework für adaptive Exploration bereichert nicht nur die theoretischen Ansätze zur Datenanalyse, sondern bietet auch neue Perspektiven und effektive Lösungen für den Umgang mit komplexen Daten in realen Anwendungen.

Contents

Acknowledgments v

Abstract vii

Zusammenfassung ix

List of Figures xvii

List of Tables xix

1 Introduction 1

1.1 Knowledge Discovery in Databases 1

1.2 Unsupervised Learning in Data Analysis 3

1.3 Challenges in Unsupervised Learning 5

1.4 Adaptive Exploration through Intrinsic Data Properties 6

1.5 Outline of the Thesis 8

2 Background 11

2.1 Nearest Neighbor Methods 11

2.1.1 Nearest Neighbor 11

2.1.2 k-Nearest Neighbors 12

2.1.3 ϵ-Nearest Neighbors 12

2.1.4 ϵ_i-Nearest Neighbors 13

2.1.5 Mutual Nearest Neighbors 13

2.1.6 Nearest Neighbor Search 14
2.2 Graph Construction 16
2.2.1 Neighbor-Based Graph Construction 17
2.2.2 Ensuring Graph Connectivity 18
2.3 Evaluation Metrics 20
2.3.1 Normalized Mutual Information 22
2.3.2 Adjusted Mutual Information 23
2.3.3 F-measure 24
2.3.4 Accuracy of the k-NN Classifier 26
2.3.5 Receiver Operating Characteristic 27
2.3.6 Average Precision 28
2.3.7 Precision at N 29
2.3.8 Procrustes Analysis 29

3 Density-Based Clustering for Adaptive Density Variation 31
3.1 Introduction 32
3.2 Related Work 34
3.3 Methodology 37
3.3.1 Problem Definition 37
3.3.2 Local Density Information 37
3.3.3 Adaptive Search Range 38
3.3.4 Mutual Nearest Neighbors 41
3.3.5 Algorithm Overview 43
3.3.6 Determining the Parameters 43
3.3.7 Complexity Analysis 47
3.4 Experiments 48
3.4.1 Experimental Setup 48
3.4.2 Parameter Sensitivity Analysis 51
3.4.3 Results on Synthetic Dataset 51

3.4.4 Robustness Analysis 56
3.4.5 Results on Real-World Datasets 57
3.5 Conclusion 57

4 Adaptive Density Outlier Detection 61
4.1 Introduction 62
4.2 Related Work 65
4.3 Methodology 67
4.3.1 Problem Definition 67
4.3.2 Algorithm Description 68
4.3.3 Algorithm Overview 73
4.3.4 Generalization to Unknown Data 74
4.3.5 Complexity Analysis 75
4.3.6 Efficiency Optimization 77
4.4 Experiments 77
4.4.1 Experimental Setup 77
4.4.2 Parameter Sensitivity Analysis 80
4.4.3 Decision Boundaries Comparison 81
4.4.4 Results on Real-World Datasets 90
4.4.5 Runtime Analysis 93
4.4.6 Visualization on Real-World Datasets 93
4.5 Conclusion 95

5 Dynamic Graph Construction for Nonlinear Dimensionality Reduction 97
5.1 Introduction 98
5.2 Related Work 101
5.3 Methodology 102
5.3.1 Problem Definition 103
5.3.2 Adaptive Neighborhood Graph 103

5.3.3 Graph Modification . . . 106
5.3.4 Objective Function for Embedding . . . 110
5.3.5 Algorithm Overview . . . 115
5.3.6 Complexity Analysis . . . 116
5.4 Experiments . . . 117
5.4.1 Experimental Setup . . . 117
5.4.2 Results on Synthetic Dataset . . . 119
5.4.3 Results on Real-World Datasets . . . 121
5.4.4 Ablation Studies . . . 123
5.4.5 Visualization on Real-World Datasets . . . 125
5.5 Conclusion . . . 126

6 Conclusion and Future Work **135**
6.1 Conclusion . . . 135
6.2 Future Work . . . 137

References **139**

List of Figures

2.1 Confusion matrix for TP, FP, TN, and FN. 25

3.1 The clustering process of DBSCAN and DBADV with different search ranges. 34

3.2 Assign points around the boundary between clusters with different densities. 41

3.3 Effect of bandwidth on the probability density distribution and corresponding search ranges of points with different densities. 42

3.4 NMI score against different search ranges of parameter p_{cum} on synthetic *ThreeBlobs* dataset. 45

3.5 NMI score against different search ranges of parameters $perp$ and $MinPts$ on synthetic *ThreeBlobs* dataset. 46

3.6 NMI score against different search ranges of parameters $perp$ and $MinPts$ on four real-world datasets. 52

3.7 Clustering performance on synthetic *Shape3-Outlier* dataset. 54

3.8 Clustering performance on synthetic *Shape5* dataset. 55

3.9 Clustering performance on synthetic *Shape3* dataset with outlier proportions ranging from 0% to 30%. 56

4.1 Decision boundaries comparison on synthetic *ThreeBlobs-Outlier* dataset using k-NN, LOF, and ADOD. 64

4.2 Illustration of key ADOD steps on synthetic *ThreeBlobs-Outlier* dataset. . . 68

4.3 Average ROC score against different search ranges of parameters p_{cum} and $perp$ on real-world datasets. 81

4.4 Average P@N score against different search ranges of parameters p_{cum} and $perp$ on real-world datasets. 82
4.5 Average AP score against different search ranges of parameters p_{cum} and $perp$ on real-world datasets. 82
4.6 Decision boundaries comparison on synthetic *ThreeBlobs-Outlier* dataset using various outlier detection algorithms. 83
4.7 CD diagram illustrating pairwise statistical difference comparison. 91
4.8 Run time comparison of 16 algorithms on 32 real-world datasets. 92
4.9 Visualization of musk, magic_gamma, and pima datasets using UMAP. . . . 94

5.1 Graph construction techniques on the toy dataset. 99
5.2 Overview of the DynoGraph algorithm. 103
5.3 Edge modification strategies. 107
5.4 Visualization of synthetic *3D Scurve-hole* dataset. 120
5.5 k-NN classifier accuracy on real-world datasets. 124
5.6 Visualization of WarpPIE10P dataset. 127
5.7 Visualization of COIL-20 dataset. 128
5.8 Visualization of LandsatSatellite dataset. 129
5.9 Visualization of COIL-100 dataset. 130
5.10 Visualization of HAR dataset. 131
5.11 Visualization of Fashion-MNIST dataset. 132
5.12 Visualization of MNIST dataset. 133

List of Tables

2.1 Comparison of nearest neighbor methods. 14
2.2 Comparison of libraries for nearest neighbor search. 16
2.3 Comparison of graph construction methods. 21

3.1 Computational complexity of DBADV and baseline methods. 47
3.2 Statistics of datasets used in DBADV. 49
3.3 Parameters and their search range for each algorithm. 50
3.4 NMI and F-measure scores on real-world datasets. 58

4.1 Statistics of the 32 real-world datasets used in ADOD. 79
4.2 ROC score on real-world datasets. 85
4.3 P@N score on real-world datasets. 86
4.4 AP score on real-world datasets. 87
4.5 Run time on real-world datasets (Part 1). 88
4.6 Run time on real-world datasets (Part 2). 89

5.1 Statistics of datasets used in DynoGraph. 118
5.2 AMI score on real-world datasets. 122

Chapter 1

Introduction

1.1 Knowledge Discovery in Databases

In recent years, the proliferation of Internet of Things (IoT) devices, the widespread use of cloud computing, and the rapid development of artificial intelligence and machine learning technologies have together driven data generation to unprecedented levels. According to SOAX[1] research, the amount of data generated globally is estimated to reach 402.89 million terabytes[2] per day in 2024, for a total of 147 zettabytes. This rapid growth trend is expected to continue, with the total amount of data expected to reach 181 zettabytes by 2025. IDC[3] predicts that the global datasphere will expand to 393.8 zettabytes by 2028, a figure that is almost identical to that predicted by Statista[4], reflecting an increase of 9.8 times compared to 2018. It's worth noting not only the total amount of data but also the speed at which it is generated. In 2023, the global data generation rate reached 4.2 petabytes per second, and this rate is expected to reach 12.5 petabytes per second by 2028[5].

This exponential growth of data not only brings significant opportunities but also

[1] https://soax.com/research/data-generated-per-day

[2] 1 ZB (zettabyte) $= 10^3$ EB (exabyte) $= 10^6$ PB (petabyte) $= 10^9$ TB (terabyte) $= 10^{12}$ GB (gigabyte) $= 10^{15}$ MB (megabyte) $= 10^{18}$ KB (kilobyte) $= 10^{21}$ Bytes

[3] https://www.idc.com/getdoc.jsp?containerId=US52076424

[4] https://www.statista.com/statistics/871513/worldwide-data-created/

[5] https://www.idc.com/getdoc.jsp?containerId=prCHC52667624

poses serious challenges for data analysis. Volume, Variety, Velocity, Veracity, and Value, known as the ”five Vs” of big data [RY21], make it increasingly complex to efficiently extract meaningful knowledge from massive amounts of data. Addressing these challenges requires a series of powerful methods to effectively manage large, multimodal, and dynamically evolving datasets. As a systematic theoretical framework, Knowledge Discovery in Databases (KDD) [FPSS96] transforms *raw data* into actionable *knowledge* through a structured multi-step process. Its main steps are as follows:

1. **Data Selection**: Selects *target data* from *raw data*, retrieving a subset relevant to the analysis task from the database.

2. **Data Preprocessing**: Cleans and prepares *target data* as *preprocessed data*, dealing with issues such as missing values, noise, duplicates, and inconsistencies to ensure the quality and reliability of the data.

3. **Data Transformation**: Converts *preprocessed data* into *transformed data*, generating suitable data representations for downstream tasks through techniques such as dimensionality reduction, normalization, or feature engineering.

4. **Data Mining**: Extracts *patterns* from *transformed data*, applying intelligent methods to identify meaningful relationships, trends, or models relevant to data mining tasks.

5. **Interpretation/Evaluation**: Explains and evaluates *patterns*, refining them into actionable *knowledge* through visualization and knowledge representation techniques.

Steps 1 to 3 of the KDD process focus on the preparation and processing of *raw data* into *transformed data* suitable for mining. Data mining [HKP11], as a key step in KDD, is defined as the process of extracting and discovering interesting patterns and knowledge from large amounts of data. The functionality of data mining determines the types of patterns that need to be discovered in different tasks. These tasks can be broadly classified

into two categories: predictive and descriptive. Predictive tasks, such as classification and regression, aim at inductive reasoning based on known data to predict unknown or future values. In contrast, descriptive tasks, such as clustering, summarization, and association rule learning, focus on revealing the inherent structure or relationships within known data. Furthermore, outlier detection (also known as anomaly detection) can be either a predictive or descriptive task, depending on the task goal.

1.2 Unsupervised Learning in Data Analysis

The KDD framework encompasses various tasks that typically adopt three main learning paradigms: supervised, semi-supervised, and unsupervised learning [JM15]. Each paradigm represents a distinct approach to analyzing data and extracting knowledge, depending on the availability of labeled data. Supervised learning maps input data to corresponding labeled output values by learning from a training set to predict unknown instances on a test set while minimizing generalization error. Semi-supervised learning, as an extension of supervised learning, combines a small amount of labeled data with a large amount of unlabeled data during training. It improves the prediction accuracy on the test set by utilizing the few labeled samples to guide the learning process while mining potential information from the large unlabeled data. In contrast, unsupervised learning analyzes the entire dataset without predefined labels and focuses on discovering inherent patterns, structures, or relationships within the data.

In many real-world scenarios, obtaining labeled data is expensive, time-consuming, or even impossible, making unsupervised learning indispensable for analyzing large and complex datasets. This thesis develops novel approaches in unsupervised learning, specifically in its core tasks: clustering, outlier detection, and dimensionality reduction. While clustering and dimensionality reduction are essentially unsupervised learning, outlier detection [HHH$^+$22] can be approached in a supervised, semi-supervised, or unsupervised manner. However, this thesis focuses on the unsupervised learning paradigm of outlier detection, emphasizing its ability to identify outliers without labeled data.

Clustering. The clustering task [HKP11] involves grouping similar data points into the same cluster based on a defined similarity metric, such that data points within the same cluster are more similar to each other than to those in other clusters. The primary goal of clustering is to reveal the inherent structure and natural groupings within an unlabeled dataset. Clustering has a variety of applications in several fields, including customer segmentation in marketing, where businesses identify groups of customers with similar behaviors; image segmentation in computer vision, which partitions images into meaningful regions; and biology, where clustering aids in identifying gene expression or grouping organisms based on phenotypic traits.

Outlier Detection. The outlier detection task [HKP11] identifies rare or unusual data points that deviate significantly from the majority, often arising from a different generating mechanism or inconsistent with the rest of the data. The goal of outlier detection is twofold: data cleaning by identifying and removing errors or irrelevant data points and uncovering rare observations that provide valuable insights. Outlier detection has a variety of applications in several fields, such as in fraud detection, where suspicious transactions or activities are flagged for further investigation; in cybersecurity, where intrusion behaviors or unusual patterns are identified; and in medical diagnostics, where abnormal conditions or potential health problems are identified through the early detection of outliers in patient data.

Dimensionality Reduction. The dimensionality reduction task [HKP11] transforms high-dimensional data into a low-dimensional representation while preserving the essential characteristics of the original data. The goal is to address the curse of dimensionality, where high-dimensional data can cause sparsity, computational inefficiency, or degradation of model generalization, while preserving the relationships and structures in the data. Dimensionality reduction is widely applied across various fields. For example, in bioinformatics, it helps visualize complex gene expression data; in signal processing, it extracts key features from time-series data; and in computer vision, it simplifies image representation for tasks such as object recognition. Dimensionality reduction promotes efficient storage, visualization, and downstream analysis.

1.3 Challenges in Unsupervised Learning

While most existing methods have demonstrated some success in many applications, they often exhibit inherent limitations that restrict their adaptability to diverse and complex datasets. The primary limitations are as follows:

Assumptions on Data Distributions. Many methods are built on strict assumptions about the overall data distributions, such as Gaussian, uniform, or exponential distributions [BN06, Mur12]. While these assumptions can simplify modeling and provide robust performance when data follows the expected distribution, in practice, the distribution of real-world data is often diverse and unpredictable, and these assumptions are often inadequate. For example, practical datasets often exhibit complex characteristics, such as multimodal distributions, heavy-tailed or skewed probabilities. They may even arise from a combination of multiple underlying distributions that deviate significantly from idealized models.

Insensitivity to Density Variations. Many methods struggle to handle datasets with varying densities due to their reliance on predefined criteria consistently applied across the entire dataset [SSE+17, BKNS00]. These methods often treat all regions of the dataset as having similar density, a simplification that hardly reflects the complexity of real-world data. By ignoring density differences across regions, these methods fail to adapt to the diverse structures present in the data. For example, in dense regions, these methods may over-segment the data, splitting continuous and meaningful clusters into multiple smaller groups. Conversely, in sparse regions, they tend to overlook subtle but important patterns, often considering sparse structures or boundary points as outliers.

Dependence on Dataset-Specific Parameters. Many methods rely on dataset-specific parameters[6] settings [EKSX96, ABKS99], which often require careful tuning for optimal performance. However, when the dataset changes, these parameters often lose

[6] In this thesis, *parameters* refer to any settings determined before algorithm execution, excluding *model parameters*, encompassing both *primary parameters*, which significantly influence performance and require manual tuning, and *default parameters*, which are predefined by the authors or implementations. No distinction is made between *parameters* and *hyperparameters* for simplicity.

effectiveness and need to be manually readjusted. This dependency poses significant obstacles in practical applications, as obtaining prior knowledge about optimal parameter values is challenging. Common strategies such as grid search or random search [BB12] enumerate candidate parameter combinations to find the best solution, but these methods are computationally expensive and time-consuming. In addition, it is challenging to choose a universally applicable evaluation metric to guide parameter optimization because most metrics only reflect a specific aspect of performance.

These three limitations are prevalent across many machine learning methods. In supervised learning, these issues can be mitigated to some extent. For instance, labeled data can provide guidance for validating distributional assumptions, understanding density variations, and optimizing parameters through techniques such as cross-validation. In contrast, unsupervised learning faces inherent challenges due to the lack of labeled data, with no direct feedback for guidance. As an alternative, unsupervised methods often rely on internal metrics or heuristics to evaluate performance [LLX+10]. While these techniques can provide some guidance, they often struggle to align with the true underlying structure of the data, which may lead to unreliable results.

The limitations above present three critical challenges in the field of unsupervised learning, which can be formulated as key research questions:

- **RQ1:** *How can methods adapt to diverse and arbitrary data distributions?*
- **RQ2:** *How can methods handle datasets with varying densities?*
- **RQ3:** *How can methods reduce dependence on dataset-specific parameters?*

1.4 Adaptive Exploration through Intrinsic Data Properties

This thesis presents an adaptive exploration framework to address the three major challenges in unsupervised learning. This framework enables the dynamic adaptation of

algorithms by leveraging intrinsic data properties, aiming to eliminate reliance on prior assumptions or predefined criteria. Specifically, when a dataset changes in terms of density, structure, size, or other key attributes, adaptive exploration can effectively capture these changes and adjust critical elements such as threshold selection and neighborhood size, thereby maintaining consistent performance. By dynamically adapting to intrinsic data properties, adaptive exploration can effectively address challenges related to arbitrary data distributions, density variations, and excessive dependence on dataset-specific parameters, providing a more flexible and generalizable solution for unsupervised learning tasks.

Intrinsic data properties refer to the inherent characteristics of a dataset that reveal its internal structure and relationships. These properties are not dependent on external annotations or a priori assumptions but are obtained by directly uncovering patterns in data distribution, interrelationships, and both global and local structural characteristics of the data points. For example, data size reflects the total number of points in the dataset, which influences the selection of neighborhood size when analyzing local structures. Local density describes the compactness around a data point, which helps to differentiate between dense and sparse regions. Neighborhood relationships reveal local connectivity structures by measuring the distance or similarity between data points. Additionally, global and local relationships between data points can be captured through graph representations, where nodes represent data points and edges indicate the similarity or adjacency between points. These properties provide a reliable basis for dynamic adaptation of the algorithm, allowing the algorithm to be optimized and adapted to the specific characteristics of the data.

Building on these intrinsic data properties, we develop the adaptive exploration framework to address the following three research questions:

Adapting to Arbitrary Distributions (RQ1). Adaptive exploration eliminates predefined assumptions about data distributions, enabling algorithms to adapt to any distribution without prior knowledge. Unlike methods that require predefined assumptions, such as Gaussian or uniform distributions, adaptive exploration relies only on the intrin-

sic data properties to capture its actual distribution characteristics. In this way, algorithms can reflect the natural structure of the data and adapt to diverse, unpredictable, and highly heterogeneous distributions, ensuring a fair and unbiased representation of potential patterns.

Capturing Local Density (RQ2). Real-world datasets often exhibit varying densities in different regions. Adaptive exploration accurately characterizes the density around individual points by focusing on the local properties of each data point rather than treating all points as having the same density. By emphasizing the local environment, adaptive exploration can maintain stable performance in both dense and sparse regions, avoiding excessive segmentation of dense clusters while also identifying subtle but important patterns in sparse regions, thus accurately capturing the density differences between regions in the data.

Designing Self-Adaptive Parameters (RQ3). Adaptive exploration leverages intrinsic data properties to design parameters that self-adapt to the dataset. Parameters such as thresholds, neighborhood sizes, and scaling factors can be dynamically derived from the internal structure of the data, or sensitive parameters can be replaced with more robust parameters, thereby minimizing the need for manual parameter adjustment. With this adaptive exploration, algorithms can automatically adjust primary parameters when the data size, local characteristics, or even the overall structure change, always maintaining efficient and stable performance and adapting to different datasets.

1.5 Outline of the Thesis

The primary goal of this thesis is to develop an adaptive exploration framework for unsupervised learning by leveraging intrinsic data properties. Specifically, we focus on three fundamental tasks: clustering, outlier detection, and dimensionality reduction. By addressing critical challenges such as adapting to diverse and arbitrary data distributions, handling datasets with varying densities, and reducing dependence on dataset-specific parameters, this framework aims to enhance the adaptability, accuracy, and efficiency of

unsupervised learning algorithms. The thesis is structured as follows:

Chapter 1 introduces the motivation and scope of this thesis, with a focus on unsupervised learning. It identifies critical challenges in the field and proposes an adaptive exploration framework to address three key research questions.

Chapter 2 provides the background knowledge to support the proposed algorithms. It introduces and compares various nearest neighbor methods and graph construction techniques, highlighting their definitions, characteristics, and limitations. Additionally, it presents the evaluation metrics used for the various tasks in this thesis.

Chapter 3 proposes DBADV, a density-based clustering algorithm. It highlights the novel definition of local density information for each data point and the enforcement of strong constraints on neighbors, enabling the algorithm to adapt to varying densities and identify clusters with arbitrary shapes and sizes while maintaining robustness to noise and outliers.

Chapter 4 proposes ADOD, a proximity-based unsupervised outlier detection algorithm. It highlights the novel estimation of local density for each data point and compares density differences with mutual neighbors to identify outliers that significantly deviate from their surroundings, enabling adaptation to varying densities and self-adaptive parameter settings.

Chapter 5 proposes DynoGraph, a nonlinear graph-based dimensionality reduction algorithm. It highlights the novel adaptive neighborhood graph construction and dynamic graph modification processes, ensuring that the low-dimensional embedding accurately reflects the intrinsic structure of the high-dimensional data. Additionally, DynoGraph employs self-adaptive parameter settings.

Chapter 6 concludes the thesis by summarizing the main contributions and discussing potential limitations. It also highlights possible directions for future research and reflects on its broader impact.

Chapter 2

Background

2.1 Nearest Neighbor Methods

Nearest neighbor methods are fundamental techniques in data analysis and machine learning, serving as core building blocks for tasks such as classification, clustering, outlier detection, and graph construction. These methods rely on spatial proximity and similarity within a feature space to establish local relationships among data points. This section introduces several common nearest neighbor methods and discusses computational techniques for efficient nearest neighbor searches.

2.1.1 Nearest Neighbor

Given a dataset $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \in \mathbb{R}^{n \times d}$, where n denotes the number of data points and d denotes the dimension of the feature space. For a given query point $\mathbf{x}_i \in \mathbf{X}$, its nearest neighbor (NN) [CH67] is defined as the point $\mathbf{x}_j \in \mathbf{X}$ that minimizes the distance to $\mathbf{x}_i$, excluding the query point itself. The NN of $\mathbf{x}_i$ is formally defined as follows:

$$\text{NN}(\mathbf{x}_i) = \arg \min_{\mathbf{x}_j \in \mathbf{X}, j \neq i} \text{distance}(\mathbf{x}_i, \mathbf{x}_j) \tag{2.1}$$

where $\text{distance}(\cdot, \cdot)$ is a general distance metric, such as the Euclidean distance, Manhat-

tan distance, cosine similarity, etc.

2.1.2 k-Nearest Neighbors

For a given query point $\mathbf{x}_i \in \mathbf{X}$, its k-nearest neighbors (k-NN) [CD21] are defined as the k points in $\mathbf{X}$ that have the smallest distances to $\mathbf{x}_i$, excluding the query point itself. The k-NN of $\mathbf{x}_i$ are formally defined as follows:

$$k\text{-NN}(\mathbf{x}_i) = \{\mathbf{x}_j \in \mathbf{X} \mid \text{distance}(\mathbf{x}_i, \mathbf{x}_j) \text{ is among the smallest } k \text{ to } \mathbf{x}_i, j \neq i\} \tag{2.2}$$

where k is a user-defined parameter that specifies the number of neighbors and distance$(\cdot, \cdot)$ is a general distance metric.

The k-NN method takes into account multiple points in the neighborhood, unlike the NN method, which considers only a single point. This makes k-NN less sensitive to outliers.

2.1.3 ϵ-Nearest Neighbors

For a given query point $\mathbf{x}_i \in \mathbf{X}$, its ϵ-nearest neighbors (ϵ-NN) [SSE+17] are defined as all points $\mathbf{x}_j \in \mathbf{X}$ that lie within a distance threshold ϵ from $\mathbf{x}_i$, excluding the query point itself. The ϵ-NN of $\mathbf{x}_i$ are formally defined as follows:

$$\epsilon\text{-NN}(\mathbf{x}_i) = \{\mathbf{x}_j \in \mathbf{X} \mid \text{distance}(\mathbf{x}_i, \mathbf{x}_j) \leq \epsilon, j \neq i\} \tag{2.3}$$

where $\epsilon > 0$ is a user-defined distance threshold and distance$(\cdot, \cdot)$ is a general distance metric. For symmetric distance metrics, such as the Euclidean distance or the Manhattan distance, ϵ-NN exhibits symmetry.

Unlike the k-NN method, which assigns a fixed number of neighbors to each data point, the number of neighbors in the ϵ-NN method is determined by the local density of

the data points. In dense regions, ϵ-NN tends to include more neighbors, while in sparse regions, the number of neighbors is usually small or may even be zero.

2.1.4 ϵ_i-Nearest Neighbors

For a given query point $\mathbf{x}_i \in \mathbf{X}$, its ϵ_i-nearest neighbors (ϵ_i-NN) are defined as all points $\mathbf{x}_j \in \mathbf{X}$ that lie within a distance threshold ϵ_i from $\mathbf{x}_i$, excluding the query point itself. This concept serves as a key component of the DBADV algorithm in Chapter 3, the ADOD algorithm in Chapter 4, and the DynoGraph algorithm in Chapter 5. The ϵ_i-NN of $\mathbf{x}_i$ are formally defined as follows:

$$\epsilon_i\text{-NN}(\mathbf{x}_i) = \{\mathbf{x}_j \in \mathbf{X} \mid \text{distance}(\mathbf{x}_i, \mathbf{x}_j) \leq \epsilon_i, j \neq i\} \tag{2.4}$$

where $\epsilon_i > 0$ is an adaptive distance threshold specific to $\mathbf{x}_i$ and distance$(\cdot, \cdot)$ is a general distance metric. Since ϵ_i typically varies across different data points, the ϵ_i-NN relationship is not symmetric.

Unlike the global fixed threshold ϵ shared by all data points, ϵ_i is determined adaptively based on the local characteristics of $\mathbf{x}_i$. The ϵ_i-NN method is not limited by density variations, as ϵ_i for $\mathbf{x}_i$ can be calculated using perplexity [vdMH08] or heuristic methods [İnk23]. In dense regions, ϵ_i is usually small, while in sparse regions, ϵ_i is usually large, so that ϵ_i-NN can provide a relatively balanced number of neighbors for points in different regions.

2.1.5 Mutual Nearest Neighbors

For a given query point $\mathbf{x}_i \in \mathbf{X}$, its mutual nearest neighbors (MNN) [AEZS21] are defined as all points $\mathbf{x}_j \in \mathbf{X}$ such that $\mathbf{x}_i$ is a neighbor of $\mathbf{x}_j$ and $\mathbf{x}_j$ is a neighbor of $\mathbf{x}_i$. The MNN of $\mathbf{x}_i$ are formally defined as follows:

$$\text{MNN}(\mathbf{x}_i) = \{\mathbf{x}_j \in \mathbf{X} \mid \mathbf{x}_i \in \mathcal{N}(\mathbf{x}_j) \quad \text{and} \quad \mathbf{x}_j \in \mathcal{N}(\mathbf{x}_i)\} \tag{2.5}$$

Table 2.1: Comparison of nearest neighbor methods.

Method	Definition	Characteristics	Limitations
NN	The closest point to a query point	Simple and intuitive	Sensitive to outliers
k-NN	The k closest points to a query point	Considers multiple neighbors, less sensitive to outliers	Sensitive to k, computationally expensive for large k
ϵ-NN	The points within a fixed distance threshold ϵ from a query point	Number of neighbors varies with local density	Sensitive to ϵ, a fixed threshold may fail in datasets with varying densities
ϵ_i-NN	The points within an adaptive distance threshold ϵ_i from a query point	Adapts to density variations, relatively balanced number of neighbors	Sensitive to ϵ_i, relies on the method used to calculate ϵ_i
MNN	Points that share a mutual neighborhood relationship with a query point	Robust to outliers, ensures symmetric relationships	Sensitive to neighborhood definition, computationally expensive for large datasets

where $\mathcal{N}(\mathbf{x}_i)$ denotes the neighborhood of $\mathbf{x}_i$. The neighborhood $\mathcal{N}(\mathbf{x}_i)$ can take on various forms, such as NN$(\mathbf{x}_i)$, k-NN$(\mathbf{x}_i)$, and ϵ_i-NN$(\mathbf{x}_i)$, which are defined in Eqs. (2.1), (2.2), and (2.3). For symmetric distance metrics, ϵ-NN$(\mathbf{x}_i)$ is inherently MNN.

Unlike the NN, k-NN, or ϵ_i-NN methods, which rely on unidirectional neighborhood relationships, MNN ensures symmetric and bidirectional neighborhood relationships, making it more robust in dealing with outliers. The number of mutual neighbors for each point depends on the definition of the neighborhood and the local data distribution.

Table 2.1 summarizes the key differences among these nearest neighbor methods by comparing their definitions, characteristics, and limitations. In this thesis, we adopt the MNN method, where the neighborhood is defined as ϵ_i-NN$(\mathbf{x}_i)$. This method can adapt to density variations and impose stronger constraints on the neighbors, thereby effectively resisting outliers.

2.1.6 Nearest Neighbor Search

Nearest neighbor search (NNS) [ML14] is an optimization problem that aims to find one or more nearest neighbors for a given query point in a dataset based on a defined distance

metric. Unlike nearest neighbor methods, such as NN, k-NN, ϵ-NN, ϵ_i-NN, and MNN, that define relationships between data points, NNS focuses on how to efficiently identify these relationships, particularly in large or high-dimensional datasets. NNS methods can be broadly categorized into exact methods and approximation methods.

Exact methods for NNS aim to identify the exact nearest neighbors by systematically traversing the dataset. The simplest method, often referred to as linear or brute force search, is implemented by calculating the distance between the query point and all other points in the dataset and finding the closest match. While this method is intuitive and applicable to any distance metric, it can be computationally expensive for large datasets. To improve efficiency, spatial partitioning techniques, such as KD-trees [Ben75] and R-trees [Gut84], can be used to organize the dataset into a hierarchical structure. This hierarchical structure allows regions irrelevant to the query point to be pruned during the search process.

Approximation methods for NNS focus on improving computational efficiency by relaxing the requirement to find the exact nearest neighbors. Instead, they aim to find neighbors that are approximately close to the query point, seeking a trade-off between accuracy and computational speed. Commonly used methods include graph-based methods, such as Hierarchical Navigable Small World (HNSW) [MY20], which uses greedy traversal on a preconstructed graph to iteratively optimize the search path, thereby efficiently locating nearby data points. In addition, Locality Sensitive Hashing (LSH) [GIM99] groups similar data points together using a chosen distance metric, ensuring that similar points are more likely to be mapped to the same bucket.

In practice, NNS is often implemented using highly optimized libraries that support both exact and approximate methods. These implementations use advanced data structures, such as trees, graphs, and hash-based indices, along with techniques like vectorized operations, quantization, parallelization, and hardware acceleration. These libraries are widely used for machine learning tasks, especially dimensionality reduction. For example, PyNNDescent[7] [DML11] is the default for UMAP [MHM18],

[7] `https://github.com/lmcinnes/pynndescent`

Table 2.2: Comparison of libraries for nearest neighbor search.

Library	Exact	Approx.	GPU	Key Features
Faiss	✓	✓	✓	Highly scalable, GPU-accelerated, supports IVF, PQ, HNSW
Annoy	×	✓	×	Memory efficient, random projection trees, supports static index sharing across processes
NMSLIB	✓	✓	×	Flexible for metric and non-metric spaces, supports HNSW and custom distance functions
PyNNDescent	×	✓	×	Nearest neighbor descent, random projection trees, graph diversification
`sklearn.neighbors`	✓	×	×	Part of scikit-learn, uses KD-tree and Ball-tree for exact search

while sklearn.neighbors[8], NMSLIB[9] [BN13], and PyNNDescent are commonly employed for t-SNE [vdMH08]. Faiss[10] [JDJ19, DGD+24], with its GPU acceleration, supports SpaceMap [ZT22], and Annoy[11] is used for methods such as PaCMAP [WHRS21], TriMap [AW19], and LargeVis [TLZM16]. Table 2.2 summarizes common libraries for NNS, highlighting their support for exact and approximate search, GPU acceleration, and key features.

In this thesis, we perform exact NNS in Python using the Faiss library with GPU acceleration support. Specifically, the `GpuIndexFlatL2` index[12] is used to efficiently calculate pairwise distances and identify nearest neighbors in large, high-dimensional datasets.

2.2 Graph Construction

Graphs provide a structured representation to model relationships among data points in a dataset. Formally, a graph $G = (V, E, \mathbf{W})$ consists of three main parts: a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$, where each vertex v_i corresponds to a data point $\mathbf{x}_i$ in the dataset $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$; a set of edges E, representing the connections between vertices,

[8] `https://scikit-learn.org/stable/modules/neighbors.html`
[9] `https://github.com/nmslib/nmslib`
[10] `https://github.com/facebookresearch/faiss`
[11] `https://github.com/spotify/annoy`
[12] `https://faiss.ai/cpp_api/class/classfaiss_1_1gpu_1_1GpuIndexFlatL2.html`

defined according to a specific criterion; and an optional weight matrix $\mathbf{W}$, where w_{ij} represents the distance or similarity between two connected vertices v_i and v_j. In this thesis, we consider G to be an undirected graph, meaning that every edge $(v_i, v_j) \in E$ represents a bidirectional connection with weight $w_{ij} = w_{ji}$.

Graph construction [QZCS18] refers to the process of defining the edges E and weights $\mathbf{W}$ to capture meaningful relationships between data points. The most commonly used methods rely on neighborhood relationships, such as k-nearest neighbor graph, ϵ-nearest neighbor graph, and mutual nearest neighbor graph. Furthermore, in specific tasks, it is crucial to ensure the graph is connected to facilitate the information flow between all vertices. To achieve this, additional strategies, such as using the minimum spanning tree, can be employed to ensure graph connectivity.

2.2.1 Neighbor-Based Graph Construction

Neighbor-based graph construction methods intuitively define edges based on the proximity between points, making them suitable for capturing the local structure of the data. Three commonly used methods are introduced below:

k-Nearest Neighbor Graph (k-NNG) In a k-NNG [HAYSZ11], each vertex $v_i \in V$ is connected to its k-nearest neighbors based on a chosen distance metric, such as the Euclidean distance, Manhattan distance, or cosine similarity. The edges are defined as follows:

$$E = \{(v_i, v_j) \mid v_j \in k\text{-NN}(v_i)\} \tag{2.6}$$

where k-NN(v_i) denotes the set of k-nearest neighbors of vertex v_i, determined by the data point $\mathbf{x}_i$ associated with v_i, as defined in Eq. (2.2).

ϵ-Nearest Neighbor Graph (ϵ-NNG) In an ϵ-NNG [CPnZ04], each vertex $v_i \in V$ is connected to other vertices within a predefined distance threshold ϵ. The edges are

defined as follows:

$$E = \{(v_i, v_j) \mid v_j \in \epsilon\text{-NN}(v_i)\} \tag{2.7}$$

where ϵ-NN(v_i) denotes the set of ϵ-nearest neighbors of vertex v_i, determined by the data point $\mathbf{x}_i$ associated with v_i, as defined in Eq. (2.3).

Mutual Nearest Neighbor Graph (MNNG) In an MNNG [BCQY97, OSKM11], each vertex $v_i \in V$ is connected to another vertex $v_j \in V$ if and only if they are mutual nearest neighbors. The edges are defined as follows:

$$E = \{(v_i, v_j) \mid v_j \in \text{MNN}(v_i)\} \tag{2.8}$$

where MNN(v_i) denotes the set of mutual nearest neighbors of vertex v_i, determined by the data point $\mathbf{x}_i$ associated with v_i, as defined in Eq. (2.5). The neighborhood $\mathcal{N}(v_i)$ used to determine mutual nearest neighbors can take on various forms, such as k-NN(v_i), or ϵ_i-NN(v_i).

In addition to defining the edge set E, graph G may also contain an optional weight matrix $\mathbf{W}$, where w_{ij} quantifies the strength of the connection between vertices. In the simplest case, an unweighted adjacency matrix ($w_{ij} \in \{0, 1\}$) is used to encode the existence of edges, where $w_{ij} = 1$ indicates $(v_i, v_j) \in E$ and $w_{ij} = 0$ otherwise. For weighted graphs, w_{ij} is a real-valued weight that reflects the similarity or distance between vertices, usually calculated using a metric such as the Gaussian kernel function or inverse distance.

2.2.2 Ensuring Graph Connectivity

Specific tasks, such as dimensionality reduction, require the graph to be connected to preserve global structure and facilitate information flow between vertices. However, most graph construction methods, such as neighbor-based methods, do not inherently guarantee connectivity. One straightforward strategy is to increase k, expand the distance

threshold ϵ, or relax the constraints to include more edges. Although these strategies may improve connectivity, they usually introduce redundant edges with limited relevance, increasing computational complexity. However, they may still fail to ensure connectivity when dealing with datasets with varying densities. A more effective strategy is to use a minimum spanning tree to ensure graph connectivity.

Minimum Spanning Tree (MST) The MST [PR02] of a connected, weighted, undirected graph $G = (V, E, \mathbf{W})$ is a subset of the edges, $T \subseteq E$, that connects all vertices in V without forming any cycles, while minimizing the total edge weight. The MST is defined as follows:

$$\text{MST}(G) = \arg\min_{T \subseteq E} \sum_{(v_i, v_j) \in T} w_{ij} \tag{2.9}$$

where T is a spanning tree of G that ensures connectivity and acyclicity, and $|T| = |V| - 1$. Commonly used algorithms for finding the MST include Kruskal's algorithm and Prim's algorithm [CLRS09]. Kruskal's algorithm constructs the MST by sorting all edges in ascending order of weight and incrementally adding edges to the spanning tree, ensuring no cycles are formed. In contrast, Prim's algorithm constructs the MST by iteratively selecting the edge with the smallest weight that connects a vertex in the current tree to a vertex outside the tree and continuously expanding the tree until it contains all vertices.

The MST is primarily designed to solve optimization problems in network design and cost minimization. The goal is to minimize the total edge weight while ensuring that all vertices are connected. MST can be applied to graph construction using this property to ensure connectivity. However, the MST is rarely used directly as the final graph due to its sparsity, containing only $|V| - 1$ edges. Instead, it is often used as a supplementary part combined with other graph construction methods to achieve connectivity and a granular representation of data relationships. There are two main ways to combine MST with other graph construction methods:

MST for Initial Graph Construction. The MST [ARBM15] can serve as an initial

structure in graph construction. A sparse but connected graph can be obtained by generating the MST on a complete graph with edges weighted by the distance or similarity between vertices. Additional edges are incorporated with other graph construction methods to capture more granular relationships.

MST for Merging Disconnected Components. When a graph constructed by other methods remains disconnected, the MST [VCP16] can merge isolated components. Each disconnected component is treated as a super vertex, and the MST is then constructed on these super vertices based on the relationship between them, such as distances between centroids or representative points. This method ensures the overall connectivity of the graph by introducing minimal additional edges required to bridge these components.

Table 2.3 summarizes the graph construction methods discussed above, highlighting their definitions, key characteristics, and limitations. ANG is a graph construction method proposed in Chapter 5.3.2 as an essential part of the DynoGraph algorithm.

2.3 Evaluation Metrics

Evaluation metrics are crucial for quantitatively measuring the performance of algorithms, providing a standardized way to compare proposed algorithms with baseline methods. This section introduces the evaluation metrics used in this thesis for clustering, outlier detection, and dimensionality reduction tasks. These metrics are selected to evaluate the quality of the proposed algorithms from various perspectives.

We denote the dataset as $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \in \mathbb{R}^{n \times d}$, where each $\mathbf{x}_i \in \mathbb{R}^d$ is a d-dimensional feature vector, and n denotes the total number of samples in the dataset. Each sample $\mathbf{x}_i$ is associated with a ground truth label y_i^{true}, forming the set of ground truth label $Y^{\text{true}} = \{y_1^{\text{true}}, y_2^{\text{true}}, \ldots, y_n^{\text{true}}\}$. Similarly, the predicted labels are denoted as $Y^{\text{pred}} = \{y_1^{\text{pred}}, y_2^{\text{pred}}, \ldots, y_n^{\text{pred}}\}$. In dimensionality reduction, the original high-dimensional dataset $\mathbf{X}$ is transformed into a low-dimensional representation $\mathbf{Z} = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n\} \in \mathbb{R}^{n \times m}$, where each $\mathbf{z}_i \in \mathbb{R}^m$ is an m-dimensional vector corresponding to the original sample $\mathbf{x}_i$.

Table 2.3: Comparison of graph construction methods. Note: Method X refers to any graph construction method.

Method	Definition	Characteristics	Limitations
k-NNG	Each vertex connects to its k nearest neighbors	Captures local structure, simple, easy to implement	Sensitive to k, struggles with varying densities, may cause disconnections
ϵ-NNG	Each vertex connects to others within a fixed distance threshold ϵ	Captures local structure, interpretable	Sensitive to ϵ, struggles with varying densities, may cause disconnections
MNNG	Two vertices connect if they are mutual nearest neighbors	Captures local structure, symmetric, robust to outliers	Sensitive to neighborhood definition, may cause disconnections
MST	A tree connecting all vertices with minimal total edge weight	Ensures connectivity, minimal edge cost, avoids cycles	Highly sparse, may not represent meaningful structures as a final graph
MST + Method X	Constructs the MST, then adds edges from Method X	Ensures connectivity, enriches granular relationships	Relies on the quality of Method X, may introduce unnecessary edges
Method X + MST	Constructs a graph with Method X, then connects components using the MST	Ensures connectivity, introduces minimal additional edges	Relies on the quality of Method X, requires additional connectivity checks
ANG	Constructs a MNNG with ϵ_i-NN as $\mathcal{N}(v_i)$, then connects components using the MST	Adaptive to local structure, symmetric, robust, connectivity, introduces minimal additional edges	Relies on the quality of method used to calculate ϵ_i, requires additional connectivity checks

2.3.1 Normalized Mutual Information

The Normalized Mutual Information (NMI) [VEB10] is an external evaluation metric for clustering task, which quantifies the similarity between the clustering results and the ground truth. By measuring the mutual information between the predicted labels Y^{pred} and the ground truth labels Y^{true}, NMI provides a normalized score that indicates how well the clustering results agree with the ground truth. The NMI is defined as follows:

$$\text{NMI}(Y^{\text{true}}, Y^{\text{pred}}) = \frac{\text{MI}(Y^{\text{true}}, Y^{\text{pred}})}{\frac{1}{2}\left(H(Y^{\text{true}}) + H(Y^{\text{pred}})\right)} \tag{2.10}$$

where $\text{MI}(Y^{\text{true}}, Y^{\text{pred}})$ denotes the mutual information, which quantifies the amount of information shared between two sets of labels, and is calculated as follows:

$$\text{MI}(Y^{\text{true}}, Y^{\text{pred}}) = \sum_{y^{\text{true}} \in Y^{\text{true}}} \sum_{y^{\text{pred}} \in Y^{\text{pred}}} P(y^{\text{true}}, y^{\text{pred}}) \log \frac{P(y^{\text{true}}, y^{\text{pred}})}{P(y^{\text{true}})P(y^{\text{pred}})} \tag{2.11}$$

where $H(Y^{\text{true}})$ and $H(Y^{\text{pred}})$ denote the entropy of the ground truth and predicted labels, respectively, which measure the uncertainty in each label set, and are calculated as follows:

$$H(Y^{\text{true}}) = -\sum_{y^{\text{true}} \in Y^{\text{true}}} P(y^{\text{true}}) \log P(y^{\text{true}}) \tag{2.12}$$

$$H(Y^{\text{pred}}) = -\sum_{y^{\text{pred}} \in Y^{\text{pred}}} P(y^{\text{pred}}) \log P(y^{\text{pred}}) \tag{2.13}$$

The NMI score ranges from $[0, 1]$, where a score of 1 indicates that the predicted labels match the ground truth labels perfectly, while a score of 0 indicates that the predicted labels are completely random and have very little correlation with the ground truth labels.

In this thesis, the NMI is implemented using the `normalized_mutual_info_score`[13]

[13] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html`

function from the `sklearn.metrics` module in Python.

2.3.2 Adjusted Mutual Information

The Adjusted Mutual Information (AMI) [VEB10] is an external evaluation metric for clustering task, which measures the similarity between the predicted labels Y^{pred} and the ground truth labels Y^{true}. Unlike NMI, AMI accounts for chance, providing a more accurate reflection of clustering performance by correcting for random label assignments. This adjustment makes it particularly useful when comparing clustering results with different numbers of clusters. The AMI is defined as follows:

$$\text{AMI}(Y^{\text{true}}, Y^{\text{pred}}) = \frac{\text{MI}(Y^{\text{true}}, Y^{\text{pred}}) - \mathbb{E}[\text{MI}(Y^{\text{true}}, Y^{\text{pred}})]}{\frac{1}{2}\left(H(Y^{\text{true}}) + H(Y^{\text{pred}})\right) - \mathbb{E}[\text{MI}(Y^{\text{true}}, Y^{\text{pred}})]} \tag{2.14}$$

where $\text{MI}(Y^{\text{true}}, Y^{\text{pred}})$ denotes the mutual information, and $H(Y^{\text{true}})$ and $H(Y^{\text{pred}})$ denote the entropy of the ground truth and predicted labels, respectively. These quantities are defined in Eqs. (2.11), (2.12), and (2.13). $\mathbb{E}[\text{MI}(Y^{\text{true}}, Y^{\text{pred}})]$ is the expected mutual information, which corrects for the overlap that may occur due to random chance. By introducing this correction term, AMI provides a more robust similarity measure, which corrects for the phenomenon that the mutual information value increases due to an increase in the number of clusters, even if there is no additional information sharing between these clusters.

The AMI score ranges from $[0, 1]$, where a score of 1 indicates that the predicted labels agree perfectly with the ground truth labels, while a score of 0 indicates that the agreement is no better than random chance.

In this thesis, the AMI is implemented using the `adjusted_mutual_info_score`[14] function from the `sklearn.metrics` module in Python.

[14] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_mutual_info_score.html`

2.3.3 F-measure

The F-measure [Chi92], also known as the F1 score, is an external evaluation metric for tasks such as classification, clustering, and information retrieval. It combines precision and recall into a harmonic mean, providing a comprehensive evaluation of model performance. The F-measure is defined as follows:

$$\text{F-measure} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.15}$$

where precision measures the proportion of true positive samples out of all predicted positive samples, and recall measures the proportion of true positive samples out of all actual positive samples. These metrics are defined as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\sum_{i=1}^{n} \mathbb{I}(y_i^{\text{pred}} = 1 \wedge y_i^{\text{true}} = 1)}{\sum_{i=1}^{n} \mathbb{I}(y_i^{\text{pred}} = 1)} \tag{2.16}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\sum_{i=1}^{n} \mathbb{I}(y_i^{\text{pred}} = 1 \wedge y_i^{\text{true}} = 1)}{\sum_{i=1}^{n} \mathbb{I}(y_i^{\text{true}} = 1)} \tag{2.17}$$

where $\mathbb{I}(\cdot)$ is an indicator function that returns 1 if the condition inside is true and 0 otherwise.

The confusion matrix in Figure 2.1 illustrates the relationship between the predicted labels Y^{pred} and the ground truth labels Y^{true}, with a detailed layout of the relevant terms used to compute the precision, recall, and ultimately the F-measure. These terms are defined as follows:

- **True Positives (TP)**: The number of samples that are correctly predicted as positive, where both the predicted labels and the ground truth labels are positive.

- **False Positives (FP)**: The number of samples that are incorrectly predicted as positive, where the predicted labels are positive, but the ground truth labels are negative.

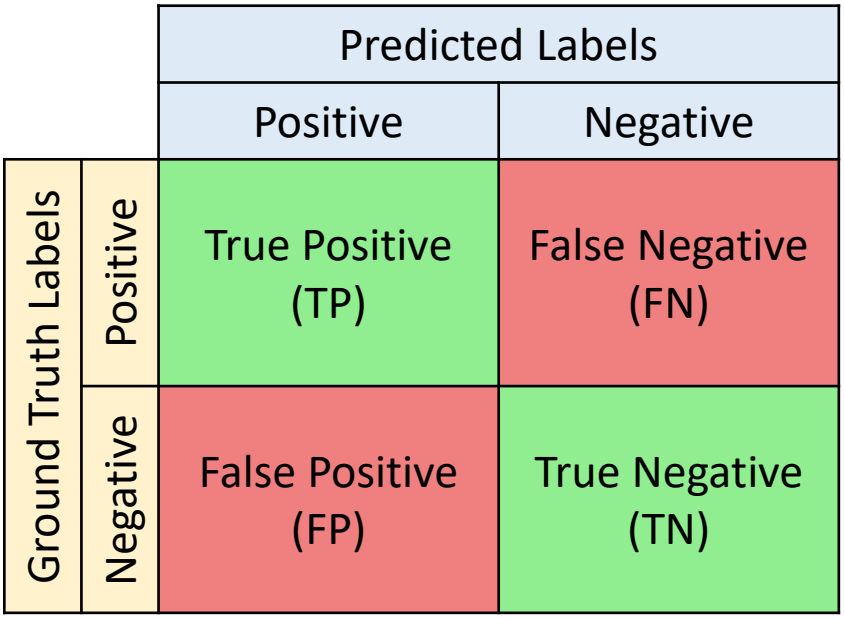


Figure 2.1: Confusion matrix for TP, FP, TN, and FN.

- **False Negatives (FN)**: The number of samples that are incorrectly predicted as negative, where the predicted labels are negative, but the ground truth labels are positive.

- **True Negatives (TN)**: The number of samples that are correctly predicted as negative, where both the predicted labels and the ground truth labels are negative.

The performance of multiple clusters is commonly evaluated using two methods: Macro-averaged F-measure and Micro-averaged F-measure. The Macro-averaged F-measure calculates the F-measure for each cluster separately and then averages these scores to assign the same weight to each cluster. In contrast, the micro-average F-measure calculates the overall F-measure score by weighting the contribution of each cluster according to its size.

In clustering evaluation, the Macro-averaged F-measure is usually preferred [ZTJA22], as it treats all clusters equally and provides a more balanced evaluation. In contrast, the Micro-averaged F-measure may potentially be biased toward clusters with more samples. The Macro-averaged F-measure is calculated as follows:

$$\text{Macro-averaged F-measure} = \frac{1}{n_c}\sum_{i=1}^{n_c}\text{F-measure}_i \tag{2.18}$$

where n_c is the number of clusters.

The F-measure ranges from $[0, 1]$, where a score of 1 indicates a perfect balance between precision and recall, meaning that the predicted labels are highly consistent with the ground truth labels, while a score of 0 indicates that the clustering result is completely inconsistent with the ground truth.

In this thesis, the Macro-averaged F-measure is used to evaluate clustering performance. Its implementation is based on the original MATLAB code provided by the authors of the DBSCAN-DScale algorithm [ZTA18][15], which was converted to Python for compatibility with our implementation.

2.3.4 Accuracy of the k-NN Classifier

Accuracy [MPM07] is an external evaluation metric, most commonly used for classification tasks, measuring the proportion of correctly classified samples out of the total samples. In the k-nearest neighbors (k-NN) [CH67] classification method, the label of each sample is determined by the majority label of its k nearest neighbors in the dataset. The dataset $\mathbf{X}$ is divided into a training set $\mathbf{X}_{\text{train}}$ and a test set $\mathbf{X}_{\text{test}}$, where the k-NN classifier is trained on $\mathbf{X}_{\text{train}}$ and evaluated on $\mathbf{X}_{\text{test}}$. Subsequently, the accuracy is obtained by calculating the proportion of correctly predicted samples in $\mathbf{X}_{\text{test}}$. The accuracy of the k-NN classifier is defined as follows:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{1}{n_{\text{test}}}\sum_{i=1}^{n_{\text{test}}}\mathbb{I}(y_i^{\text{pred}} = y_i^{\text{true}}) \tag{2.19}$$

where n_{test} is the number of samples in the test set.

The accuracy score ranges from $[0, 1]$, where a score of 1 indicates that all predicted labels match the ground truth labels exactly, while a score of 0 indicates that the classifi-

[15] https://sourceforge.net/projects/distance-scaling/

cation results completely mismatches the ground truth.

In this thesis, we refer to the accuracy of the k-NN classifier as the k-NN classifier accuracy for brevity. The k-NN classifier accuracy is implemented using the `KNeighborsClassifier`[16] and `cross_val_score`[17] functions from the `sklearn.neighbors` and `sklearn.model_selection` modules in Python.

2.3.5 Receiver Operating Characteristic

The Receiver Operating Characteristic (ROC) [Faw06] curve is a graphical representation widely used to evaluate the performance of binary classification models, such as classification and outlier detection. The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at different thresholds on a coordinate graph.

The TPR, also known as sensitivity or recall, measures the proportion of actual positive samples correctly identified as positive. The FPR, on the other hand, represents the proportion of actual negative samples that are misclassified as positive. These rates are calculated as follows:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\sum_{i=1}^{n} \mathbb{I}(y_i^{\text{true}} = 1 \wedge y_i^{\text{pred}} = 1)}{\sum_{i=1}^{n} \mathbb{I}(y_i^{\text{true}} = 1)} \tag{2.20}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} = \frac{\sum_{i=1}^{n} \mathbb{I}(y_i^{\text{true}} = 0 \wedge y_i^{\text{pred}} = 1)}{\sum_{i=1}^{n} \mathbb{I}(y_i^{\text{true}} = 0)} \tag{2.21}$$

The Area Under the ROC Curve (AUC-ROC) [HM82] summarizes the ROC curve into a scalar value that measures the overall performance of the model in distinguishing between predicted and ground truth labels. The AUC-ROC is defined as follows:

$$\text{AUC-ROC} = \int_0^1 \text{TPR}(\text{FPR})\, d(\text{FPR}) \tag{2.22}$$

[16] `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html`

[17] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html`

Here, TPR(FPR) denotes that TPR is a function of FPR, and d(FPR) denotes the change in FPR along the ROC curve.

The AUC-ROC score ranges from $[0, 1]$, where a score of 1 indicates that the predicted labels are identical to the ground truth labels at all thresholds. A score of 0.5 indicates that the predictions are no better than random guessing, showing a lack of discriminative ability. A score of 0 indicates that the predicted results are consistently inverted relative to the ground truth.

In this thesis, we refer to AUC-ROC score as the ROC score to emphasize its direct connection to the ROC curve. The ROC score is implemented using the `roc_auc_score`[18] function from the `sklearn.metrics` module in Python.

2.3.6 Average Precision

The Average Precision (AP) [Zhu04] is an external evaluation metric for tasks such as ranking, classification, and outlier detection. It summarizes the Precision-Recall curve into a scalar value by calculating the weighted mean of the precisions at different thresholds, with the increase in recall as the weight. The AP is defined as follows:

$$\text{AP} = \sum_i (\text{Recall}_i - \text{Recall}_{i-1}) \cdot \text{Precision}_i \tag{2.23}$$

where Precision_i and Recall_i denote the precision and recall at the i-th threshold, respectively. The term $(\text{Recall}_i - \text{Recall}_{i-1})$ denotes the increase in recall between two consecutive thresholds.

The AP score ranges from $[0, 1]$, where a score of 1 indicates that the predicted labels agree exactly with the ground truth labels across all levels of recall, while a score of 0 indicates that the predicted results fail to align with the ground truth as recall increases.

In this thesis, the AP is implemented using the `average_precision_score`[19] function from the `sklearn.metrics` module in Python.

[18] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html

[19] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html

2.3.7 Precision at N

The Precision at N (P@N) [JK17] is an external evaluation metric for tasks such as ranking and outlier detection. It measures the proportion of correctly predicted samples in the top N ranked results, focusing on the performance of the model in high-priority predictions. By concentrating on the precision of the top N samples, P@N is particularly useful in scenarios where the most relevant results need to be identified accurately. The P@N is defined as follows:

$$\text{P@N} = \frac{\sum_{i=1}^{N} \mathbb{I}(y_i^{\text{true}} = 1)}{N} \tag{2.24}$$

where N denotes the number of top-ranked samples considered.

The P@N score ranges from $[0, 1]$, where a score of 1 indicates that the predicted labels for all the top N samples are identical to the ground truth labels, while a score of 0 indicates that the top N predictions are completely different from the ground truth.

In this thesis, N is set to the number of ground truth outliers in the dataset. The P@N is implemented using the `precision_n_scores` function from the `pyod` library[20], specifically located in the `pyod.utils.utility` module in Python.

2.3.8 Procrustes Analysis

The Procrustes analysis [Gow75] is a statistical method for quantitatively assessing the geometric dissimilarity between two datasets. This technique aligns one dataset to another through optimal scaling, translation, and rotation. Given a target dataset $\mathbf{X}$ and a comparison dataset $\mathbf{Z}$, Procrustes analysis aligns by minimizing the disparity between $\mathbf{X}$ and $\mathbf{Z}$. The disparity is defined as follows:

$$\text{Disparity}(\mathbf{X}, \mathbf{Z}) = \min_{b, \mathbf{T}, \mathbf{c}} \|b\mathbf{Z}\mathbf{T} + \mathbf{c} - \mathbf{X}\|_F^2 \tag{2.25}$$

where b is a scaling factor that adjusts the size of $\mathbf{Z}$ to match $\mathbf{X}$, $\mathbf{T}$ is an orthogonal

[20] `https://github.com/yzhao062/pyod`

rotation matrix that aligns $\mathbf{Z}$ with the orientation of $\mathbf{X}$, and $\mathbf{c}$ is a translation vector that shifts the centroid of $\mathbf{Z}$ to coincide with that of $\mathbf{X}$. The Frobenius norm $\|\cdot\|_F$ measures the element-wise differences between the transformed $\mathbf{Z}$ and $\mathbf{X}$. By solving for b, $\mathbf{T}$, and $\mathbf{c}$, Procrustes analysis provides an optimal transformation that minimizes the disparity between the two datasets.

The disparity value quantifies the alignment error to measure the similarity between $\mathbf{X}$ and $\mathbf{Z}$. The lower the disparity, the higher the similarity between the two datasets.

In this thesis, we use Procrustes analysis to evaluate the performance of dimensionality reduction techniques. Specifically, we assess how well the geometric structure of the original data is preserved after dimensionality reduction by comparing the low-dimensional embedding with the original data. The Procrustes analysis is implemented using the `procrustes`[21] function from the `scipy.spatial` module in Python.

[21] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.procrustes.html`

Chapter 3

Density-Based Clustering for Adaptive Density Variation

Cluster analysis plays a crucial role in data mining and knowledge discovery. Although many researchers have investigated clustering algorithms over the past few decades, most well-known algorithms have limitations in dealing with clusters of arbitrary shapes and varying sizes in the presence of noise and outliers. Density-based methods have partially solved these issues but failed to discover clusters with varying densities. In this chapter, we propose a novel **D**ensity-**B**ased clustering algorithm for **A**daptive **D**ensity **V**ariation (DBADV), which is based on the classic density-based clustering algorithm DBSCAN. To address the problem of density variation, we define the local density information to reflect the individual property of each object and describe the density distribution of clusters, then find the adaptive search range of each object by collecting information from its neighbors. Moreover, we design a new metric to obtain the mutual nearest neighbors of each object and detect the objects around the boundaries between clusters more effectively. We show the effectiveness of our algorithm in extensive experiments on synthetic and real-world datasets, demonstrating that the performance of DBADV is superior to other competitive clustering algorithms.

Parts of the material presented in this chapter have been published in [QPB21]:

"Li Qian, Claudia Plant, Christian Böhm. Density-based Clustering for Adaptive Density Variation. 2021 IEEE International Conference on Data Mining (ICDM), pp. 1282-1287, 2021."

where Li Qian was primarily responsible for the development of the main concepts, the implementation of the algorithm, the experimental evaluation, and the writing of the paper. Christian Böhm proposed the initial idea of applying perplexity to density-based clustering. Christian Böhm and Claudia Plant supervised and guided the research, reviewed the manuscript, and provided suggestions for revisions.

3.1 Introduction

As one of the most important unsupervised learning tasks, clustering methods are applied in various data analysis fields [HKP11, ZTY+20]. Clustering aims to find natural groupings of data, such that objects within the same cluster are similar while objects in different clusters are dissimilar. The existing clustering methods are generally divided into the following categories [SPG+17]: partitioning-based, hierarchical-based, density-based, and graph-based methods. However, most of these methods have some limitations: they hardly handle clusters of arbitrary sizes and shapes, are sensitive to noise and outliers, require prior knowledge (e.g., a pre-defined number of clusters), and are formidable to maintain a relatively reasonable computational complexity.

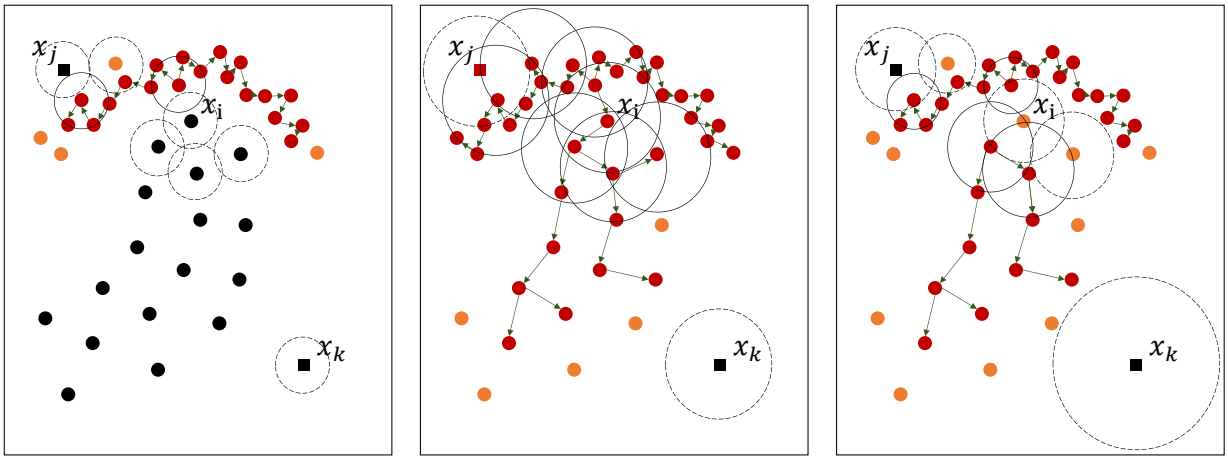
To tackle the challenges above, we aim to define the local density information, which contributes to describing the density distribution of clusters with fine granularity. Considering that density-based methods focus on density information but fail to deal with clusters with varying densities, we propose a novel Density-Based clustering algorithm for Adaptive Density Variation (DBADV) that inherits such density property and attempt to use the local density information to solve the problem of density variation. As the classic representative of density-based methods, the basic idea of Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [EKSX96] is adopting a global search

range (ε-$neighborhood$) for all objects to distinguish relatively dense regions from sparse regions. In contrast, DBADV regards the local density information as the individual property of each object. Namely, each object has its own adaptive search range. A toy dataset can illustrate the intuition of our algorithm to shed some light on the difference between DBSCAN and DBADV. As shown in Fig. 3.1, the toy dataset contains one upper half-moon cluster with high density, one random-shaped cluster with low density, and two distinct outliers marked as squares. x_i, x_j, and x_k represent boundary point and outliers, respectively. We set the minimum number of points $MinPts$ within the search range to 4 (including itself). DBSCAN and DBADV treat the points with at least $MinPts$ in their search range as *core points* (red). The non-core points within the search range of *core points* are called *border points* (orange). The remaining points after clustering are *outliers*[22] (black). The green arrow denotes the search path from the current *core point* to the next *core point*. The solid circle represents the search range of the *core point*, and the dashed circle represents the search range of the non-core point. For clarity, we only highlight partial circles of the representative points. Fig. 3.1(a) shows that DBSCAN, with a small global search range, can find the dense upper half-moon but assign all points in the sparse region as *outliers*. With a large global search range, DBSCAN treats the boundary point x_i as the next *core point* and further merges all points in the sparse region to one cluster, including the *outlier* x_j, as shown in Fig. 3.1(b). On the contrary, DBADV finds the adaptive search range of each point through the local density information collected from neighbors. The adaptive search range is small in the relatively dense region and large in the sparse region. Since each point has its own adaptive search range, DBADV correctly identifies two clusters with different densities and two outliers, as shown in Fig. 3.1(c).

The primary contributions include:

- We define the local density information and find the adaptive search range for each

[22] In this thesis, we distinguish between *noise* and *outliers*: *noise* refers to points within a cluster whose values follow a certain distribution, such as Gaussian noise; *outliers* refer to points that do not belong to any cluster.



(a) (b) (c)

Figure 3.1: The clustering process of DBSCAN and DBADV with different search ranges (partial). (a) DBSCAN with a small global search range; (b) DBSCAN with a large global search range; (c) DBADV with an adaptive search range.

object, enabling DBADV to discover clusters with varying densities.

- We design a new metric to search the mutual nearest neighbors of each object, which can detect the objects around the indistinguishable boundaries between clusters more efficiently.
- The experimental results show that DBADV can handle clusters of arbitrary shapes and sizes with varying densities and is robust to noise and outliers.

3.2 Related Work

This section introduces the development of density-based methods and some well-known clustering algorithms as baselines in our experiments.

Density-based methods [KKSZ11] can effectively exploit density information to find clusters of arbitrary sizes and shapes while remaining robust against noise and outliers. They do not require the user to specify the number of clusters but fail to identify clusters with varying densities due to the global search range, such as DBSCAN. Numerous improvements to DBSCAN have attempted to overcome this limitation. Ordering Points

To Identify the Clustering Structure (OPTICS) [ABKS99] defines the *reachability distance* and draws a reachability plot, such that all points are sorted in a special linear order, with spatially adjacent points following each other closely. OPTICS relies on order points to identify the clustering structure and expects density drops to detect the boundaries of the clusters. Shared Nearest Neighbors (SNN) clustering algorithm [ESK03, LWY18] uses SNN similarity instead of the distance measure. SNN similarity of any two points relies on the overlap between lists of their k-nearest neighbors. However, k-NN algorithm [AWY16] is highly sensitive to the parameter k. Density peaks (DP) clustering algorithm [RL14] characterizes cluster centers as a higher density than their neighbors and a relatively large distance from points with higher density. Whereas DP has a limited clustering effect on data with varying density distribution, equilibrium distribution, and multiple domain-density maximums, leading to problems such as sparse cluster loss and cluster fragmentation. Thus, Domain-Adaptive Density Clustering algorithm [CY21] is proposed to address these problems. However, the first step of the k-NN based domain-adaptive density measurement method leads to a heavy dependence on the parameter k for the whole clustering process. ReScale [ZTC16] is an adaptive scaling approach that operates as a preprocessing step to rescale a given dataset. It then applies the rescaled dataset to an existing density-based method, such as DBSCAN, OPTICS, and SNN. However, ReScale is a one-dimensional scaling method that is applied to each attribute independently. Therefore, DScale [ZTA18], a multidimensional scaling method, is proposed to consider all dimensions simultaneously. Using DScale as a preprocessing can improve accuracy but requires an extensive search of three parameters. Clustering with Robust Autocuts and Depth (CRAD) algorithm [HG17] is based on a new neighbor searching function using statistical data depth as the dissimilarity measure. CRAD is restricted by the design of its algorithm (e.g., requiring invertible matrices) and therefore cannot be applied to all data.

From another perspective, some algorithms combine the benefits of density-based methods and other categories of clustering methods to alleviate some of the inherent disadvantages of each method. The Hierarchical DBSCAN (HDBSCAN) [CMZS15] combines

density-based and hierarchical-based methods. HDBSCAN forms an MST to connect all points in the hyperspace and defines *mutual reachability distance* as its edge weight between two vertices in MST. However, processing boundary points for HDBSCAN is not ideal, and the minimum number of clusters needs to be defined. SpectACl [HDHM19] combines density-based and graph-based methods. SpectACl uses minimum cut from Spectral clustering [NJW01] and maximum density from DBSCAN to find clusters with a large average density. The appropriate density for each cluster is automatically determined through the spectrum of the weighted adjacency matrix. Nevertheless, SpectACl requires the user to specify the number of clusters and cannot handle outliers.

There are still many other categories of clustering methods that are widely used. It is worth comparing with them to verify the effectiveness of the proposed algorithm DBADV. k-means [Jai10, KR17] is one of the most famous partition-based methods, and its basic idea is to randomly initialize k cluster centers and assign points to the nearest cluster center. k-means has relatively low runtime complexity and high computing efficiency, but it is unsuitable for non-convex data, is relatively sensitive to outliers, and requires prior knowledge. Affinity Propagation clustering algorithm [FD07] can also be considered as one of the partition-based methods, which works based on similarities between pairs of points. Its core idea is to regard all points as the potential cluster centers, which are called *exemplars*, and the negative value of the Euclidean distance between two points as the affinity. Affinity Propagation has relatively high complexity, cannot handle outliers, and is only suitable for spherical clusters. Spectral clustering algorithm [NJW01, YGPB16] is evolved from graph theory. It treats the points in the dataset as vertices and constructs the similarity graph and similarity matrix. Then, the Laplacian matrix is constructed and normalized to perform the eigenvalue decomposition. Finally, k-means is applied to obtain the clusters by treating the first k eigenvectors as the center points. Spectral clustering has a highly complex complexity, cannot handle outliers, and requires the number of clusters. Self-tuning spectral clustering algorithm [ZMP04] is a variant of Spectral clustering, which automatically learns parameters in an unsupervised manner by combining a local scaling method for constructing the similarity graph and a

method for automatically selecting the number of clusters from the spectrum. Self-tuning spectral clustering has the same complexity as Spectral clustering, which cannot handle outliers. Additionally, it requires manually setting a specific range before automatically selecting the number of clusters. Synchronous clustering algorithm [BPSY10, STG+19] is a parameter-free method, which is based on the Kuramoto model [ABPV+05] to simulate the dynamics of each point during the process toward synchronization and discover cluster structure and outliers automatically in combination with the Minimum Description Length Principle [Grü04].

3.3 Methodology

3.3.1 Problem Definition

We aim to perform clustering in a multidimensional dataset while identifying potential outliers. The dataset is defined as $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \in \mathbb{R}^{n \times d}$, where each data point $\mathbf{x}_i \in \mathbb{R}^d$ is a vector in d-dimensional space, and n denotes the number of data points. The objective is to assign a cluster label c_i to each data point $\mathbf{x}_i$, forming the label set $C = \{c_1, c_2, \ldots, c_n\}$, where $c_i \in \{1, 2, \ldots, k, \textit{outlier}\}$, and k represents the number of clusters discovered by the algorithm. A label of *outlier* indicates that the point is not assigned to any cluster but is instead identified as an outlier.

3.3.2 Local Density Information

In information theory, perplexity is used to evaluate distribution or model performance, such as in language models [Sen12]. Recently, the notion of perplexity has been applied to other fields. The t-Distributed Stochastic Neighbor Embedding (t-SNE) [vdMH08], an effective technique for dimensionality reduction, uses perplexity to find the bandwidth of each object. Inspired by that, we aim to find the local density information reflecting individual property for each object through perplexity. To the best of our knowledge, this is the first time to apply the notion of perplexity to clustering algorithms.

The perplexity of a discrete probability distribution p is defined as follows:

$$\text{Perplexity}(p) = 2^{H(p)} \tag{3.1}$$

where $H(p)$ is the Shannon entropy of p measured in bits: $H(p) = -\sum_x p(x)\log_2 p(x)$.

For a dataset X, we define the local density information of a point $\mathbf{x}_i$ by collecting all the conditional probabilities $p_{j|i}$ with a neighbor point $\mathbf{x}_j$. This conditional probability shows the similarity between the two points. For a close point, $p_{j|i}$ is relatively high, whereas for a faraway point, $p_{j|i}$ is almost infinitesimal. Mathematically, we define this conditional probability $p_{j|i}$ as follows:

$$p_{j|i} = \frac{\exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma_i^2\right)}{\sum_{l \neq i} \exp\left(-\|\mathbf{x}_i - \mathbf{x}_l\|^2 / 2\sigma_i^2\right)} \tag{3.2}$$

where σ_i is the bandwidth of the Gaussian kernel centered on $\mathbf{x}_i$, and $\mathbf{x}_l$ is any point in the dataset except $\mathbf{x}_i$. The bandwidth of a point in the dense region is usually smaller than that in the sparse region. Therefore, we can consider the bandwidth of each point as the local density information. The local density information reflects the individual property of each point and describes the density distribution of clusters with fine granularity.

We set the same perplexity for the probability distribution of each point in the entire dataset, which means that the probability distribution of each point has the same Shannon entropy $H(p_i) = -\sum_j p_{j|i}\log_2 p_{j|i}$. Consequently, a binary search is performed to find the bandwidth of each point by approximating the Shannon entropy of the conditional probability p_i of $\mathbf{x}_i$ to the logarithm of the fixed perplexity, as described in Algorithm 1.

3.3.3 Adaptive Search Range

Unlike DBSCAN, which uses a global search range for all data points, our goal is to find the adaptive search range of each point based on the local density information. To achieve this, we leverage the quantile function [SS08] derived from the cumulative distribution function (CDF) [DFO20] of a Gaussian distribution. In probability theory,

Algorithm 1: Binary Search

Input: Dataset $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \in \mathbb{R}^{n \times d}$, Perplexity $perp$
Output: Bandwidths of all points $S_\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\} \in \mathbb{R}^n$

Initialize tolerance $tol = 1e-5$, iteration $iter = 50$, bandwidth set $S_\sigma = \emptyset$
foreach $\mathbf{x}_i \in X$ **do**
  Initialize loop $t = 0$, bandwidth $\sigma_i = 1$, maximum bandwidth $\sigma_{\max} = inf$, minimum bandwidth $\sigma_{\min} = -inf$
  repeat
    foreach $\mathbf{x}_j \in X$ **do**
      $p_{j|i} = \frac{\exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2{\sigma_i}^2\right)}{\sum_{l \neq i} \exp\left(-\|\mathbf{x}_i - \mathbf{x}_l\|^2 / 2{\sigma_i}^2\right)}$ `// Eq. (3.2)`
    $H(p_i) = -\sum_j p_{j|i} \log_2 p_{j|i}$
    $diff = H(p_i) - \log_2 perp$
    if $diff > 0$ **then**
      $\sigma_{\max} = \sigma_i$
      if $\sigma_{\min} == -inf$ **then**
        $\sigma_i = \sigma_i / 2$
      else
        $\sigma_i = (\sigma_i + \sigma_{\min}) / 2$
    else
      $\sigma_{\min} = \sigma_i$
      if $\sigma_{\max} == inf$ **then**
        $\sigma_i = \sigma_i \cdot 2$
      else
        $\sigma_i = (\sigma_i + \sigma_{\max}) / 2$
    $t = t + 1$
  until $|diff| < tol$ **or** $t > iter$
  $S_\sigma = S_\sigma \cup \{\sigma_i\}$
return S_σ

the CDF, denoted as $F(x)$, describes the cumulative probability that a random variable $\mathcal{X}$ takes a value less than or equal to x: $F(x) = P(\mathcal{X} \leq x)$. For a Gaussian distribution with mean μ and standard deviation σ, the CDF can be expressed as follows:

$$F(x) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)\right] \tag{3.3}$$

where $\operatorname{erf}(\cdot)$ is the error function [SE14].

The quantile function, denoted as $F^{-1}(p_{\text{cum}})$, is the inverse of the CDF. It specifies the value x such that the probability of the variable being less than or equal to x is equal to the given cumulative probability p_{cum}: $F^{-1}(p_{\text{cum}}) = x$, where $F(x) = p_{\text{cum}}$. For a Gaussian distribution, the quantile function can be written as follows:

$$F^{-1}(p_{\text{cum}}) = \mu + \sqrt{2}\sigma \ \operatorname{erf}^{-1}(2p_{\text{cum}} - 1) \tag{3.4}$$

where $\operatorname{erf}^{-1}(\cdot)$ is the inverse error function.

In our algorithm, the quantile function is used to adaptively determine the search range for each point. Specifically, for a given cumulative probability p_{cum}, the quantile function calculates the distance from the center of the Gaussian distribution (i.e., the mean of the Gaussian distribution) to the point where the cumulative probability equals p_{cum}. It is important to note that the Gaussian distribution here models the local density around each point $\mathbf{x}_i$, rather than assuming that the entire dataset follows a Gaussian distribution. For a Gaussian distribution centered at $\mathbf{x}_i$ with bandwidth σ_i, its adaptive search range ϵ_i is defined as follows:

$$\epsilon_i = \sqrt{2}\sigma_i \ \operatorname{erf}^{-1}(2p_{\text{cum}} - 1) \tag{3.5}$$

where $p_{\text{cum}} \in (0.5, 1)$ ensures $\epsilon_i > 0$ and increases monotonically with p_{cum}.

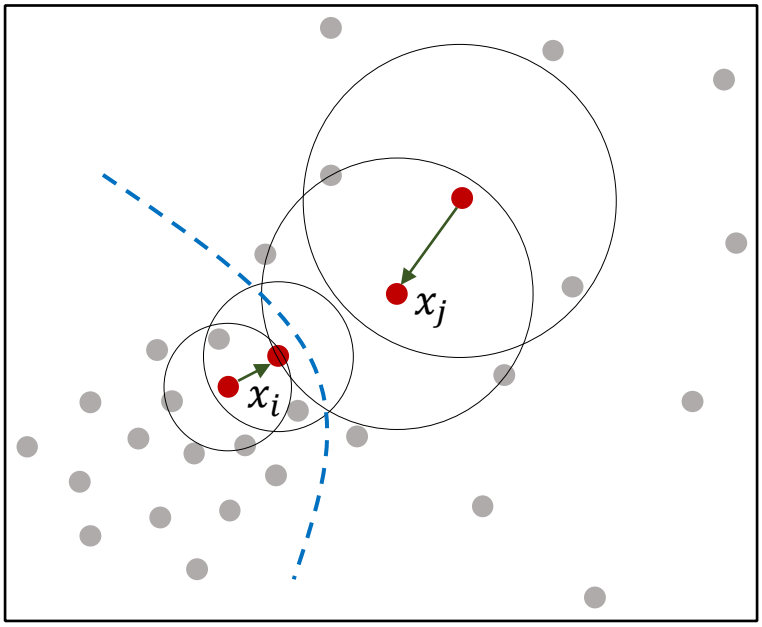


Figure 3.2: Assign points around the boundary between clusters with different densities.

3.3.4 Mutual Nearest Neighbors

Further, we try to detect the points around the indistinguishable boundaries between clusters with varying densities. As shown in Fig. 3.2, there are two clusters of different densities, and the blue dashed curve shows the boundary between dense and sparse regions, which is difficult to distinguish even by hand. We only highlight the representative points and mark the remaining points in gray. The adaptive search range of points in the dense region is generally small and vice versa. In the traditional nearest neighbor methods [AWY16], since the *core point* $\mathbf{x}_i$ is within the adaptive search range of the *core point* $\mathbf{x}_j$, $\mathbf{x}_j$ in the sparse region takes $\mathbf{x}_i$ in the dense region as the next search point. Then $\mathbf{x}_i$ and its neighbors are merged to the same cluster, though they are obviously in different clusters with varying densities.

To address this issue, we propose a mutual nearest neighbors metric as an effective tool to discover robust and reliable clusters. As shown in Fig. 3.3, the green point on the left is in the sparse region with a large bandwidth, while the red point on the right is in a dense region with a small bandwidth. Therefore, We define the mutual nearest

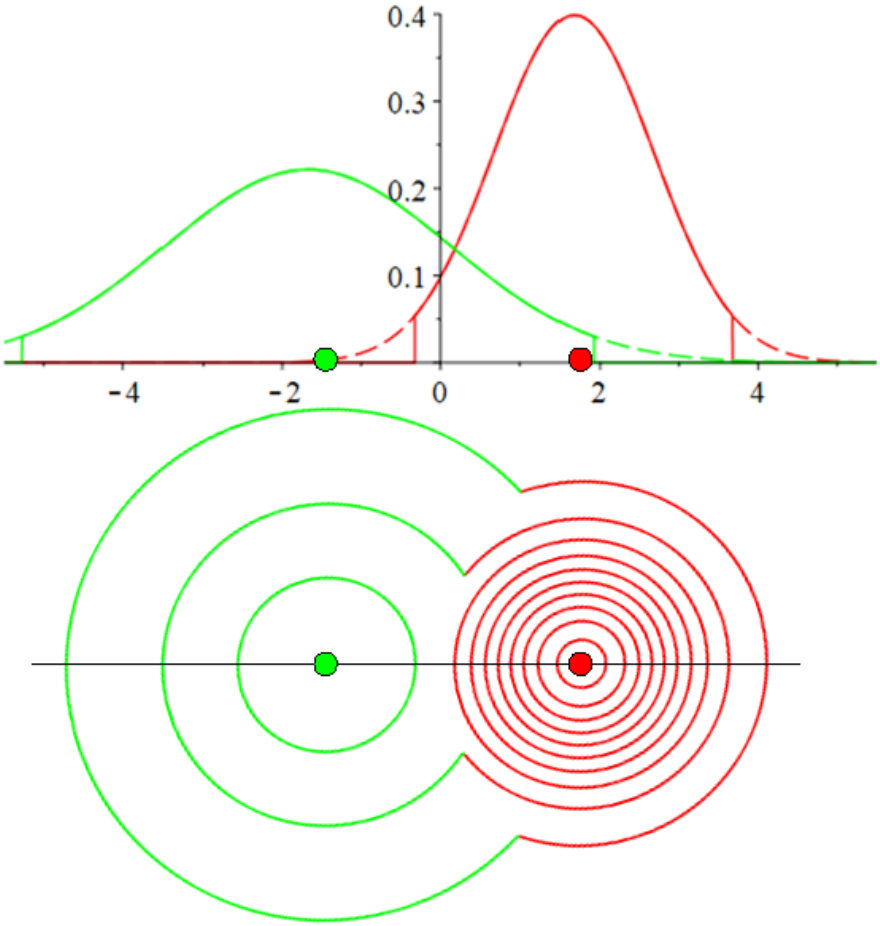


Figure 3.3: Effect of bandwidth on the probability density distribution and corresponding search ranges of points with different densities.

neighbors on the condition that both points are within the adaptive search range of each other. Since we only need to consider pairs of points in each step rather than the entire dataset, we can solely focus on the distance between these two points. The mutual nearest neighbors set MNN_i of the point $\mathbf{x}_i$ is defined as follows:

$$\text{MNN}_i = \{\mathbf{x}_j \in X \mid \text{distance}(\mathbf{x}_i, \mathbf{x}_j) \leq \min(\epsilon_i, \epsilon_j), j \neq i\} \tag{3.6}$$

where $\mathbf{x}_j$ is a neighbor point of $\mathbf{x}_i$ and $\text{distance}(\mathbf{x}_i, \mathbf{x}_j)$ denotes the Euclidean distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. Through this metric, boundary points in Fig. 3.2 can be easily identified, as points such as $\mathbf{x}_i$ and $\mathbf{x}_j$ are not mutual neighbors.

3.3.5 Algorithm Overview

We first briefly review some notions of DBSCAN. The relationship between a *core point* and the neighbors within its global search range is called *directly density-reachable*. If any of these neighbors is a *core point* again, its neighbors are also transitively included in the same cluster. The relationship between all these neighbors and a series of these *core points* is called *density-reachable*. All points within the same cluster are called *density-connected*.

The proposed algorithm DBADV defines *cluster* as a region of smoothly varying density separated from other clusters by a remarkable local density change. DBADV can be generally divided into three parts: Firstly, we find the bandwidth as the local density information of each point through Binary Search (Algorithm 1), which iteratively adjusts the bandwidth for a fixed perplexity $perp$ (Line 2). The quantile function is then used to obtain the adaptive search range for each point, determined by the cumulative probability p_{cum} and the bandwidth σ_i of the point $\mathbf{x}_i$ (Lines 3-5). Secondly, the notion of *directly density-reachable* is redefined by identifying mutual nearest neighbors using the adaptive search ranges of each point (Lines 6-11), and the $MinPts$ is reconsidered as the minimum number of mutual nearest neighbors. Moreover, the notions of *density-reachable* and *density-connected* also change according to $MinPts$ and *directly density-reachable*. Finally, clustering is performed the same as DBSCAN (Lines 12-29). The pseudocode of the DBADV algorithm is described in Algorithm 2.

3.3.6 Determining the Parameters

We give an effective heuristic to determine the primary parameters cumulative probability p_{cum}, perplexity $perp$, and the minimum number of mutual nearest neighbors $MinPts$ using the NMI score [VEB10], which is detailed in Chapter 2.3.1. We generate synthetic dataset *ThreeBlobs* containing three isotropic Gaussian blobs, each of which has the same number from 500 to 5,000 with the step size 500, and different densities with the ratio of their standard deviations given as 2:5:1. The search range of p_{cum} is

Algorithm 2: DBADV

Input: Dataset $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \in \mathbb{R}^{n \times d}$, Perplexity $perp$, Minimum number of mutual nearest neighbors $MinPts$, Cumulative probability p_{cum}

Output: Point labels $C = \{c_1, c_2, ..., c_n\} \in \mathbb{R}^n$

Initialize each point label $c_i \in C$ as $undefined$, number of clusters $k = 0$
$S_\sigma = \text{Binary Search}(X, perp)$ `// cf. Algorithm 1`
foreach $\mathbf{x}_i \in X$ **do**
    $\sigma_i = S_\sigma[i]$
    $\epsilon_i = \sqrt{2}\sigma_i \, \text{erf}^{-1}(2p_{\text{cum}} - 1)$ `// Eq. (3.5)`
foreach $\mathbf{x}_i \in X$ **do**
    Initialize $\text{MNN}_i = \emptyset$
    foreach $\mathbf{x}_j \in X$ **do**
        $\text{distance}(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\|\mathbf{x}_i - \mathbf{x}_j\|^2}$
        if *distance*$(\mathbf{x}_i, \mathbf{x}_j) \leq \min(\epsilon_i, \epsilon_j)$ **then**
            $\text{MNN}_i = \text{MNN}_i \cup \{\mathbf{x}_j\}$ `// Eq. (3.6)`
foreach $\mathbf{x}_i \in X$ **do**
    if $c_i \neq undefined$ **then**
        continue
    if $|MNN_i| < MinPts$ **then**
        $c_i = outlier$
        continue
    $k = k + 1$
    $c_i = k$
    $S_k = \text{MNN}_i \setminus \{\mathbf{x}_i\}$ `// Set of unprocessed points in cluster` k
    foreach $\mathbf{x}_q \in S_k$ **do**
        if $c_q == outlier$ **then**
            $c_q = k$
        if $c_q \neq undefined$ **then**
            continue
        $c_q = k$
        if $|MNN_q| < MinPts$ **then**
            continue
        $S_k = S_k \cup \text{MNN}_q$
return C

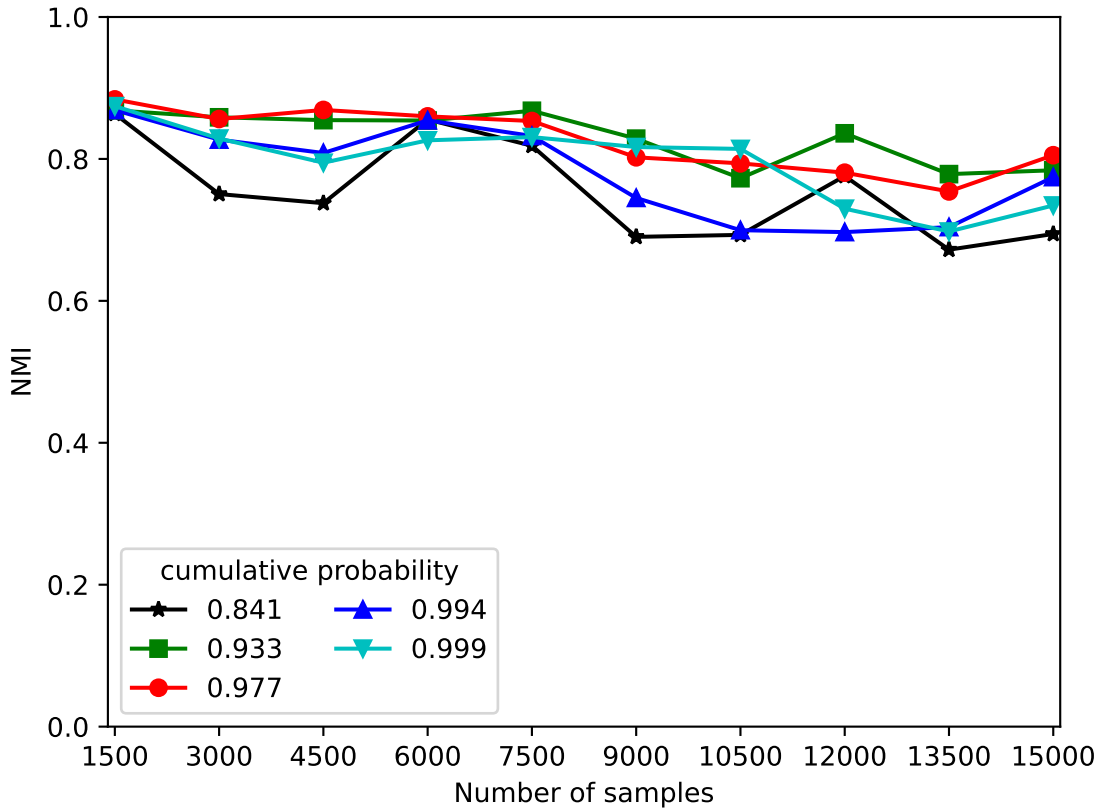


Figure 3.4: NMI score against different search ranges of parameter p_{cum} on synthetic *ThreeBlobs* dataset.

$\{0.841, 0.933, 0.977, 0.994, 0.999\}$, because it corresponds to approximately 1, 1.5, 2, 2.5 and 3 standard deviations [Dou16], respectively. $perp$ and $MinPts$ have the same large search range $[1, 100]$.

For p_{cum}, the feasible theoretical search range is $(0.5, 1)$. Since the quantile function of the Gaussian distribution is continuous and strictly monotonically increasing, the adaptive search range of each object increases as the cumulative probability increases once the bandwidth is obtained. As shown in Fig. 3.4, *ThreeBlobs* has the number of samples from 1,500 to 15,000, the search range of $perp$ and $MinPts$ is from 1 to 100, and the set of p_{cum} is $\{0.841, 0.933, 0.977, 0.994, 0.999\}$. We record the highest NMI score obtained by DBADV at different cumulative probabilities. Overall, DBADV can achieve good performance at different cumulative probabilities. The cumulative probability value is too

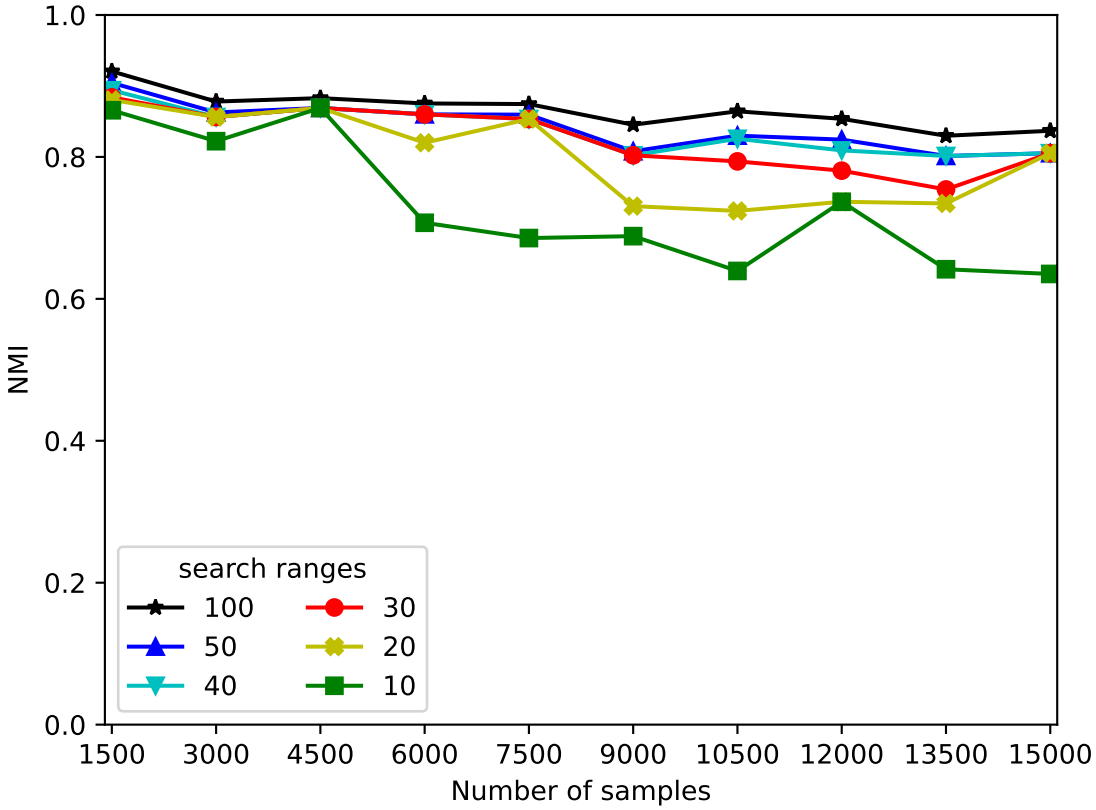


Figure 3.5: NMI score against different search ranges of parameters $perp$ and $MinPts$ on synthetic *ThreeBlobs* dataset.

high (0.994 and 0.999) or too low (0.841), causing slight performance degradation. The cumulative probability values of 0.933 and 0.977 have similar performances. However, the larger the value of cumulative probability p_{cum} is, the more perplexity tends to get high NMI score at a lower cost. Therefore, we can fix p_{cum} to 0.977 due to its quantile function is approximately 2 standard deviations. According to the empirical rule, 2 standard deviations can contain more than 95% information of a Gaussian distribution centered on the current object.

Since each probability distribution p_i has the same perplexity, the bandwidth σ_i increases monotonically with the perplexity. With the same cumulative probability, the higher the bandwidth, the more extensive the adaptive search range for each object, such that $MinPts$ is prone to an extensive search range correspondingly. We set the num-

Table 3.1: Computational complexity of DBADV and baseline methods.

Algorithm	Complexity
k-means	$O(n_{iter} \cdot n_c \cdot n)$
DBSCAN	$O(n^2)$
DBADV	$O(n^2)$
OPTICS	$O(n^2)$
HDBSCAN	$O(n^2)$
DBSCAN-Dscale	$O(n^2)$
CRAD	$O(n^2)$
Synchronous	$O(T \cdot n_l \cdot n^2)$
Affinity Propagation	$O(n^2 \log n)$
SpectACl	$O(n^3)$
Spectral clustering	$O(n^3)$
Self-tuning spectral clustering	$O(n^3)$

ber of samples for *ThreeBlobs* from 1,500 to 15,000 with the step size of 1,500. p_{cum} is fixed to 0.977, and $perp$ and $MinPts$ have the same search range with values ranging from 10 to 100, and the step size is 10. Fig. 3.5 shows the highest NMI score in the different search ranges of $perp$ and $MinPts$, ignoring the search range from 60 to 90 because their results are very similar and close. The principle of selecting an optimal search range is to get a higher NMI score in a relatively small search range. The search range 30 is the appropriate choice because both 10 and 20 have large drops and fluctuations, while 30 is consistent with the more extensive search range (40-100).

3.3.7 Complexity Analysis

The asymptotic computational complexity of DBADV is composed of three main parts as follows. The first part is to perform a binary search on the bandwidth and obtain the adaptive search range for each object with a complexity of $O(n^2)$; the second part is to find the mutual nearest neighbors for each object with a complexity of $O(n^2)$; finally, the complexity of discovering clusters is $O(n^2)$. Therefore, the overall asymptotic computational complexity of the proposed algorithm is $O(n^2)$, which is the same as DBSCAN.

In our experiments, we further compare the complexity of the proposed algorithm

with baselines. Considering that these clustering algorithms are implemented in different programming languages (Python, Java, Matlab), and some of them have already been optimized by the authors for efficiency, we only list the theoretical asymptotic computational complexity without any optimization, as shown in Table 3.1. k-mean has the lowest runtime complexity, where n_{iter} and n_c represent the number of iterations and the number of clusters, respectively. DBSCAN, along with its variants OPTICS, HDBSCAN, CRAD, and DBSCAN-DScale, have the same complexity $O(n^2)$. The complexity of the Synchronous is slightly higher, where T and n_l represent the time evolution and the number of different clustering models, respectively. Affinity Propagation has relatively high complexity. SpectACl, Spectral clustering, and its variant Self-tuning spectral clustering have the highest complexity $O(n^3)$. Compared with these clustering algorithms, DBADV is at a relatively efficient level.

3.4 Experiments

3.4.1 Experimental Setup

Experimental Environment

The proposed DBADV algorithm was implemented in Python. All experiments related to this algorithm were conducted on a machine equipped with an Intel Core i7-8700 CPU (six cores, 3.19 GHz) and 32GB of RAM. The code repository is available at `https://github.com/Qian-Lily/DBADV`.

Datasets

We evaluated DBADV and baselines on two synthetic datasets, *Shape3-Outlier* and *Shape5*, generated by us; eight common datasets with multivariate, including Seeds, Leaf, Dermatology, One-hundred plant species leaves shape (Plant Species), Image Segment, Page Blocks, Crowdsourced Mapping (Crowdsourced), Letter Recognition from

Table 3.2: Statistics of datasets used in DBADV.

Dataset	#Samples	#Dimensions	#Classes
Shape3	2250	2	3
Shape3-Outlier	2317	2	3
Shape5	3150	2	5
Seeds	210	7	3
warpPIE10P	210	2420	10
Leaf	340	15	30
Dermatology	358	34	6
COIL20	1440	1024	20
Plant Species	1600	64	100
Image Segment	2310	19	7
Page Blocks	5473	10	5
Crowdsourced	10545	28	6
Letter Recognition	20000	16	26

the UCI Machine Learning Repository [DG17]; and two face image datasets with high dimensionality, including COIL20 and warpPIE10P from the Feature Selection Repository[23]. These datasets have samples ranging from 210 to 20,000, dimensions ranging from 2 to 2,420, and classes ranging from 3 to 100. We preprocessed all datasets by scaling each attribute to the range $[0, 1]$ using the `MinMaxScaler`[24] class from the `sklearn.preprocessing` module. The statistics of these datasets are briefly summarized in Table 3.2.

Comparison Methods

To extensively evaluate the performance of DBADV, we compared it against 11 representative and state-of-the-art clustering algorithms. These included density-based methods such as DBSCAN[25] [EKSX96] and its variants OPTICS[26] [ABKS99],

[23] `https://jundongl.github.io/scikit-feature/datasets.html`

[24] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html`

[25] `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html`

[26] `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.OPTICS.html`

Table 3.3: Parameters and their search range for each algorithm. n_c denotes the number of clusters; $median$ represents the median of the similarity matrix.

Algorithm	Parameters with search ranges
DBADV	$MinPts \in \{1, 2, ..., 30\}$; $perp \in \{1, 2, ..., 30\}$; $p_{\text{cum}} = 0.977$
DBSCAN	$MinPts \in \{1, 2, ..., 30\}$; $eps \in \{0.01, 0.02, ..., 1\}$
OPTICS	$MinPts \in \{1, 2, ..., 30\}$; $\xi \in \{0.01, 0.02, ..., 0.99\}$
HDBSCAN	$minSample \in \{2, 3, ..., 30\}$; $minCluster \in \{2, 3, ..., 30\}$; $\alpha = 1$
CRAD	$StepSize = 1$; $Nbin \in (0.2 * n - 100, 0.2 * n + 100)$
DBSCAN-DScale	$MinPts \in \{1, 2, ..., 30\}$; $eps \in \{0.01, 0.02, ..., 1\}$; $\eta \in \{0.1, 0.2, ..., 0.5\}$
SpectACl	$eps \in \{0.01, 0.02, ..., 1\}$; $numCluster = n_c$; $d = 50$
k-means	$numCluster = n_c$
Spectral	$numCluster = n_c$; $\sigma = 0.04$
Self-tuning	$numCluster \in \{2, 3, ..., 100\}$; $K = 7$
Affinity Prop.	$lam \in \{0.5, 0.55, ..., 0.95\}$; $p = median$

HDBSCAN[27] [CMZS15], CRAD[28] [HG17], DBSCAN-DScale[29] [ZTA18], and SpectACl[30] [HDHM19]; other categories of clustering methods such as k-means[31] [Jai10], Spectral Clustering [NJW01], Self-tuning Spectral Clustering [ZMP04], Affinity Propagation[32] [FD07], and Synchronous [BPSY10]. The implementations for Spectral Clustering, Self-tuning Spectral Clustering, and Synchronous algorithms were obtained from the original authors of Self-tuning Spectral Clustering and Synchronous, respectively. For each comparison algorithm, we either searched for the best achievable clustering result within a reasonable parameter range or used the parameters recommended by the authors, as summarized in Table 3.3. For DBADV, we fixed p_{cum} at 0.977 and set the search range for $perp$ and $MinPts$ to $[1, 30]$, as described in Section 3.3.6. Synchronous is a parameter-free clustering algorithm. We ran all initialization-dependent algorithms (e.g., k-means, Spectral Clustering) 10 times and reported the best results.

[27] https://github.com/scikit-learn-contrib/hdbscan
[28] https://github.com/DataMining-ClusteringAnalysis/CRAD-Clustering/
[29] https://sourceforge.net/projects/distance-scaling/
[30] https://sfb876.tu-dortmund.de/spectacl/index.html
[31] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
[32] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.affinity_propagation.html

Evaluation Metrics

The evaluation metrics are performed through external measures NMI score [VEB10] and F-measure score [ZTC16], as described in Chapter 2.3.1 and Chapter 2.3.3, respectively. For clustering algorithms that can detect outliers, outliers are treated as an independent cluster when computing NMI score. AMI score [VEB10], introduced in Chapter 2.3.2, provides results similar to NMI score and thus is omitted here. F-measure score is particularly effective in handling outliers in clustering, as demonstrated in [ZTA18].

3.4.2 Parameter Sensitivity Analysis

In Section 3.3.6, we validated the effectiveness of setting the DBADV parameters p_{cum} to 0.977 and using the same search range of $[1, 30]$ for both $perp$ and $MinPts$ on the synthetic *ThreeBlobs* dataset. Based on this validation, the same parameter settings were applied to the real-world datasets, which also performed well. Fig. 3.6 shows NMI score against parameters $perp$ and $MinPts$ on four real-world datasets Dermatology, COIL20, Image Segment, and Letter Recognition. Overall, these recommended parameter settings are highly robust, provided that combinations of very high $MinPts$ and very low $perp$ are avoided.

3.4.3 Results on Synthetic Dataset

We generate a synthetic dataset *Shape3* with different densities, shapes, and sizes, which contains one lower half-moon with low density, one S-curve with high density, and one blob with low density that are all polluted by Gaussian noise. To evaluate the performance of clustering algorithms in the presence of outliers, we extend this dataset by adding 3% distinct global outliers evenly distributed across the space, ensuring they are outside the existing clusters. We refer to this variant as *Shape3-Outlier*. As shown in Fig. 3.7, DBADV is the only algorithm that can accurately detect three clusters and almost all outliers. In addition, it pulls away from the runner-up by a large margin due to

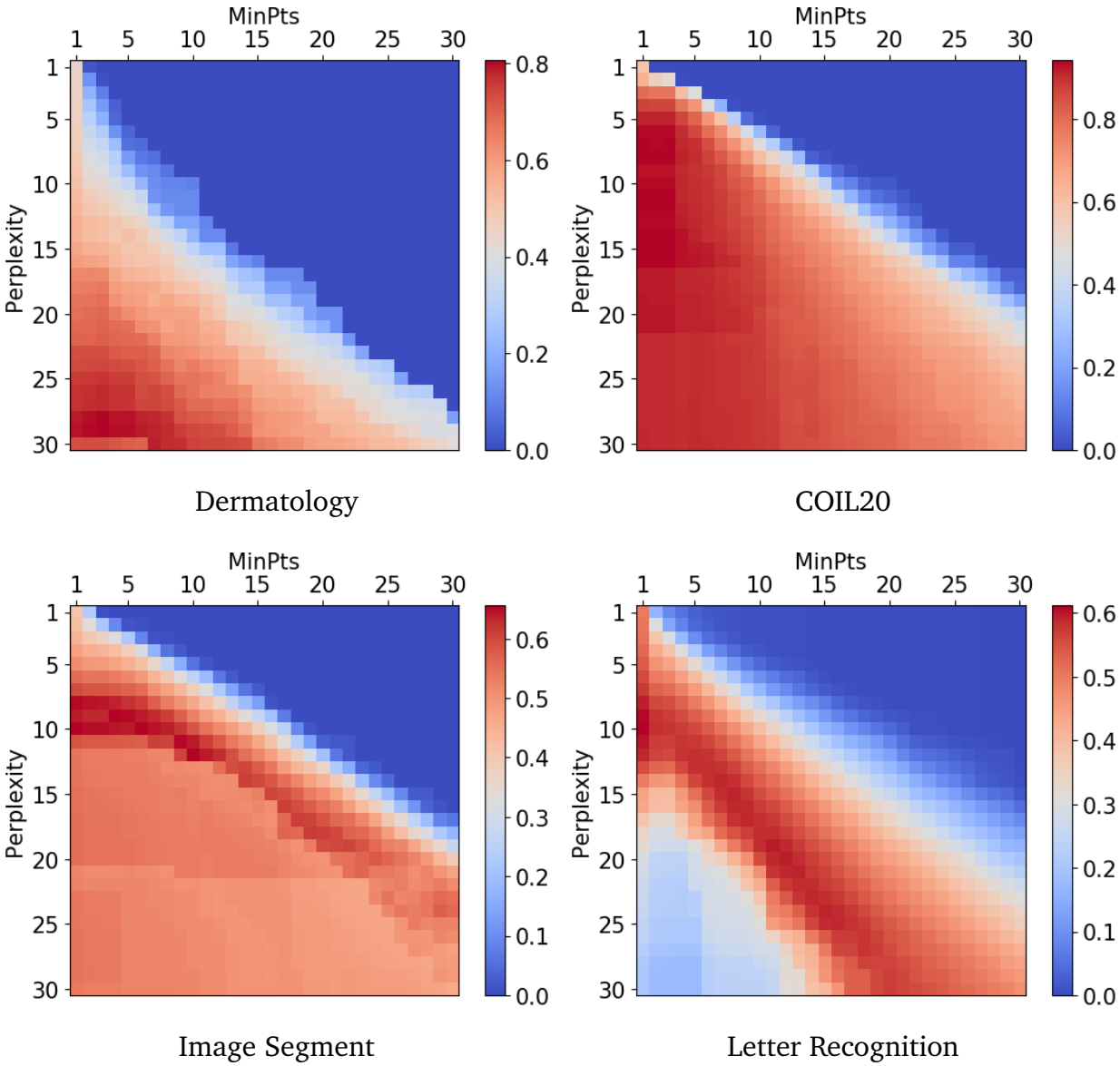


Figure 3.6: NMI score against different search ranges of parameters $perp$ and $MinPts$ on four real-world datasets.

its robustness to noise and outliers. CRAD, as the runner-up in terms of NMI score, can identify most of the major clusters. However, for the boundary points, especially between the high-density S-curve and the low-density lower half-moon, CRAD assigns them to the opposite cluster. In contrast, CRAD assigns outliers that fail to be identified to separate smaller clusters. DBSCAN, HDBSCAN, and DBSCAN-DScale can accurately identify three clusters and some outliers but show weaknesses when dealing with boundary points of clusters with Gaussian noise. SpectACl also identifies three clusters but cannot find outliers. Instead, it allocates most outliers and the boundary points of the blob to the same low-density cluster. k-means, Spectral clustering, and Self-tuning spectral clustering have similar results that cannot detect clusters with different shapes and assign outliers to the nearest cluster. OPTICS assigns both the lower half-moon and the blob with low density as outliers and divides the S-curve with high density into numerous clusters. Affinity Propagation and Synchronous erroneously discover too many clusters.

To verify that DBADV can handle clustering in more complex environments, such as indistinguishable boundaries and different data distribution, we generate synthetic dataset *Shape5*. It contains two concentric circles polluted by Gaussian noise and three homogeneous rectangles with different densities and sizes. As shown in Fig. 3.8, DBADV is the only algorithm that can accurately detect these five clusters and clearly distinguish the boundaries between clusters. DBSCAN-DScale is slightly inferior to DBADV, which can also recognize most regions of the clusters but fails to distinguish the boundary points, that is, between concentric circles and rectangles and between rectangles of different densities. For complex environments, many algorithms show vulnerability. Synchronous splits two concentric circles of different densities into three clusters. HDBSCAN treats points in low density as outliers or many small clusters. SpectACl, OPTICS, CRAD, Spectral clustering, and Self-tuning spectral clustering face failure in handling boundary points and are prone to join adjacent clusters into a large cluster. OPTICS and DBSCAN take rectangles of different densities as one cluster and consider all points of low density as outliers. CRAD, k-means, Spectral clustering, and Affinity Propagation fail to discover clusters of arbitrary shapes and sizes.

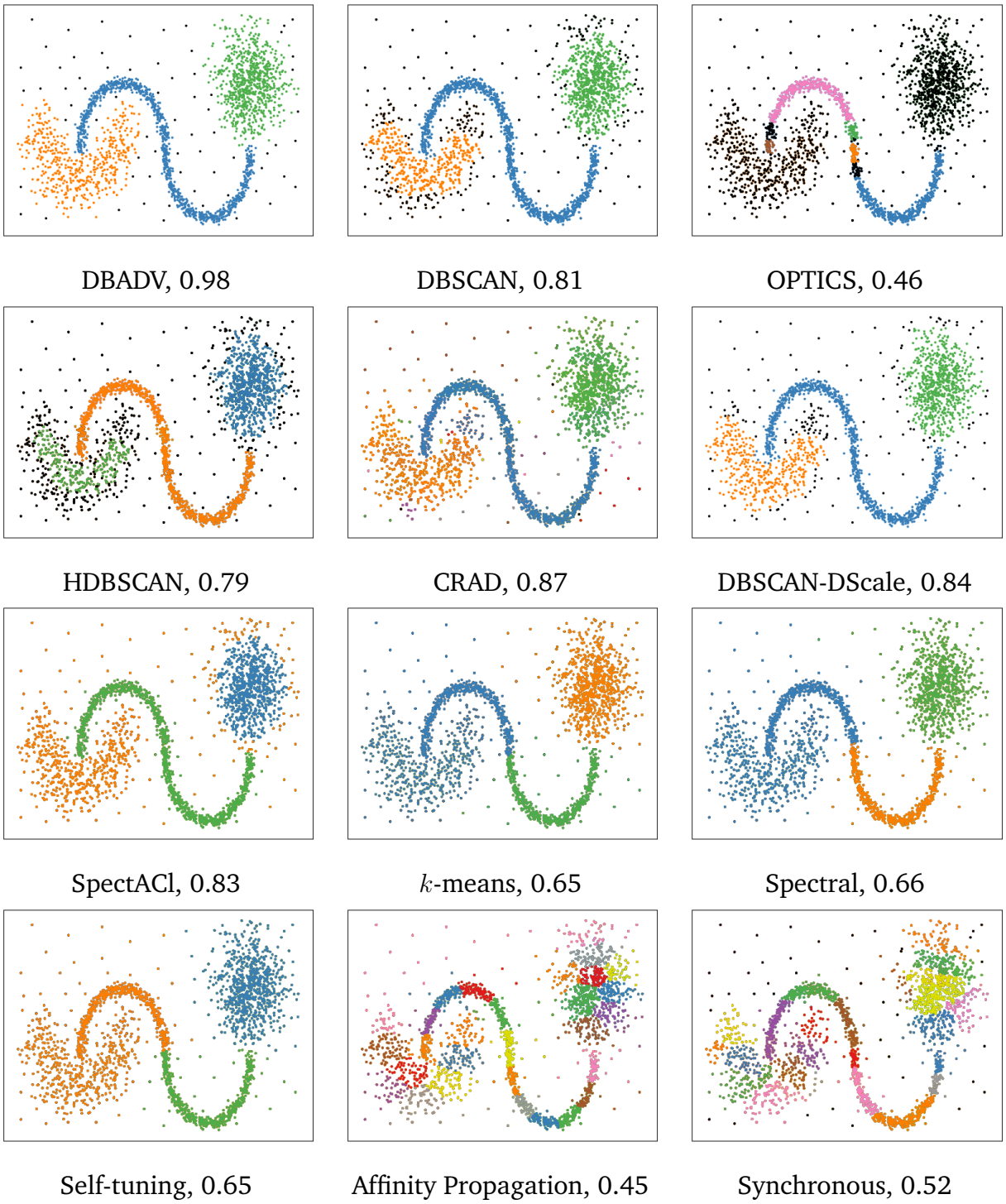


Figure 3.7: Clustering performance on synthetic *Shape3-Outlier* dataset. The first term indicates the clustering algorithm, and the second term represents the corresponding NMI score.

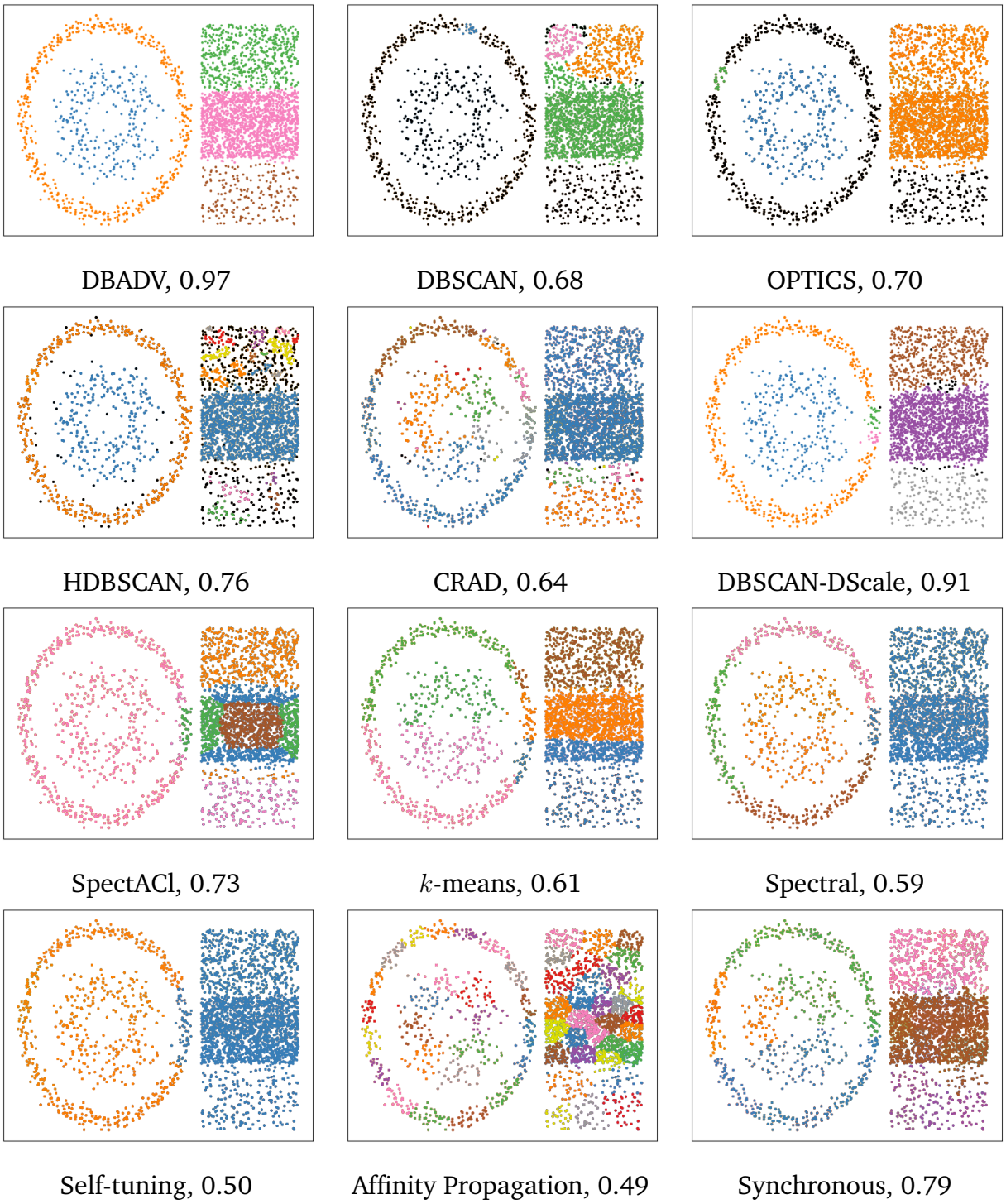


Figure 3.8: Clustering performance on synthetic *Shape5* dataset. The first term indicates the clustering algorithm, and the second term represents the corresponding NMI score.

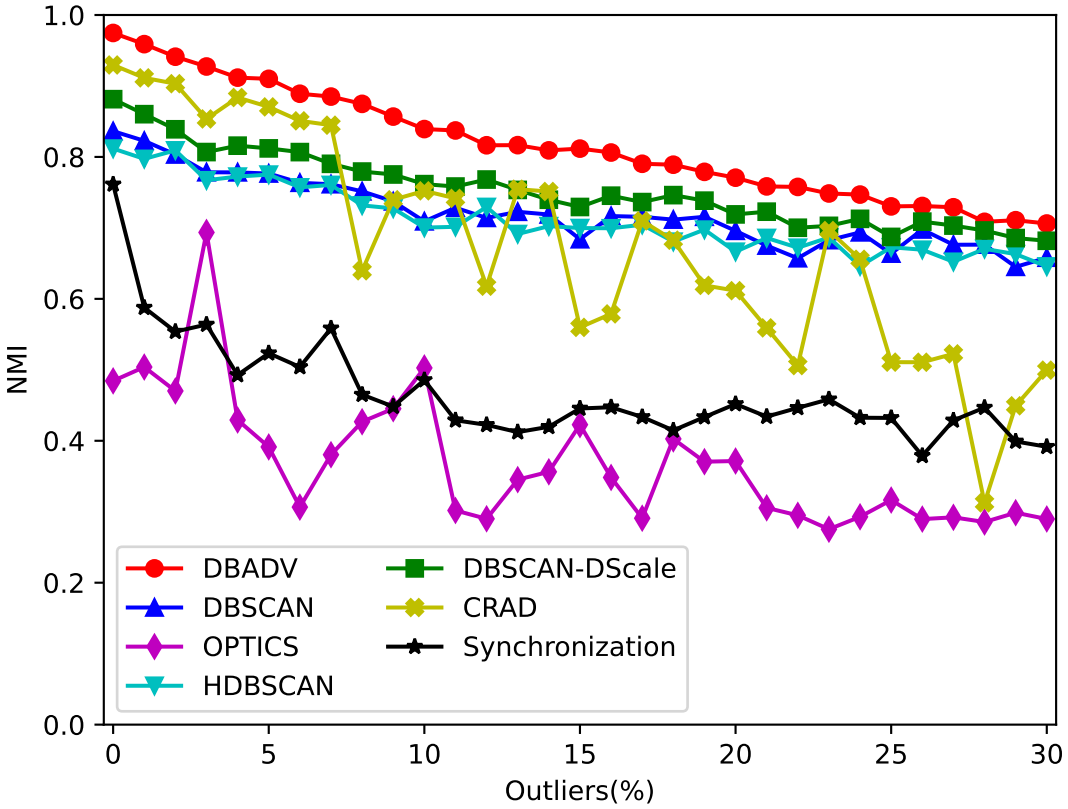


Figure 3.9: Clustering performance on synthetic *Shape3* dataset with outlier proportions ranging from 0% to 30%.

3.4.4 Robustness Analysis

Among all baselines, only DBADV, DBSCAN, OPTICS, HDBSCAN, CRAD, DBSCAN-DScale, and Synchronous can detect outliers. We explore the robustness of these algorithms against outliers by randomly adding outliers from 0% to 30% on synthetic dataset *Shape3*. As shown in Fig. 3.9, DBADV outperforms baselines in all cases where different proportions of outliers exist. DBSCAN-DScale, DBSCAN, and HDBSCAN have almost the same performance and trends and follow behind DBADV. It is worth noting that the performance of CRAD ranks second initially, but there is a significant decline and fluctuation when the proportion of outliers reaches 8%. The performance of Synchronous drops dramatically in the presence of outliers. OPTICS has the worst performance and a large fluctuation in the existence of outliers.

3.4.5 Results on Real-World Datasets

We chose ten real-world datasets from different domains, with dimensions ranging from 7 to 2,420, which are challenging due to their multivariate and high dimensionality. As shown in Table 3.4, DBADV outperforms the 11 baselines on all benchmarks in terms of NMI score, especially on warpPIE10P and Page Blocks datasets, with a substantial margin compared to the runner-up. DBADV achieves the best results for F-measure score in 7 out of 10 datasets and gets second place in the remaining datasets. In addition, DBADV also performs superior on high-dimensional datasets (e.g., COIL20 with 1,024 dimensions and warpPIE10P with 2,420 dimensions) and large datasets (e.g., Crowdsourced with 10,545 objects and Letter Recognition with 20,000 objects). With the extensive search range for three parameters, the overall performance of DBSCAN-DScale is closest to DBADV. HDBSCAN, SpectACl, and k-means follow closely in performance, earning runner-up on some datasets. DBSCAN, HDBSCAN, k-means, and Affinity Propagation have similar performances, showing moderate levels on all datasets. The performance of OPTICS, SpectACl, and Synchronous varies greatly depending on the properties of datasets, which means that they perform well on some datasets and poorly on others. CRAD, Spectral clustering, and Self-tuning spectral clustering perform exceptionally poorly on some datasets, such as warpPIE10P. Some algorithms cannot be applied to specific datasets due to the limitations of their algorithm design, such as CRAD, Self-tuning spectral clustering, and Synchronous. SpectACl fails to process warpPIE10P because its parameter eps requires a larger search range.

3.5 Conclusion

We propose a novel Density-Based clustering algorithm for Adaptive Density Variation (DBADV), which can handle clusters of arbitrary shapes and sizes with varying densities and is robust to noise and outliers. DBADV not only defines the local density information to find the adaptive search range of each object, but also designs a new metric to

Table 3.4: NMI and F-measure scores on real-world datasets (N/A: Not Applicable). The best value is in bold; the runner-up is underlined.

Method	Seeds		Leaf		Dermatology		Plant Species		Image Segment	
	NMI	F-measure	NMI	F-measure	NMI	F-measure	NMI	F-measure	NMI	F-measure
DBADV	**0.68**	**0.91**	**0.76**	**0.43**	**0.81**	**0.74**	**0.79**	0.39	**0.66**	**0.64**
DBSCAN	0.60	0.79	0.74	0.25	0.62	0.52	0.78	0.24	0.63	0.59
OPTICS	0.65	0.58	0.64	0.41	0.77	0.58	0.63	0.34	0.62	0.39
HDBSCAN	0.41	0.48	0.56	0.39	0.64	0.33	0.67	**0.40**	0.65	0.57
CRAD	0.42	0.65	0.62	**0.43**	0.46	0.33	0.29	0.03	N/A	N/A
DBSCAN-DScale	0.65	0.84	0.74	0.26	0.80	0.59	0.78	0.23	0.65	0.61
SpectACl	0.66	0.89	0.59	0.38	0.49	0.48	0.68	0.35	0.59	0.58
k-means	0.67	0.89	0.64	0.41	0.79	0.55	0.71	0.38	0.62	0.62
Spectral	0.02	0.22	0.43	0.13	0.03	0.11	0.50	0.09	0.40	0.38
Self-tuning	0.38	0.17	0.32	0.04	0.46	0.26	0.40	0.01	0.00	0.04
Affinity Propagation	0.51	0.50	0.66	**0.43**	0.67	0.59	0.86	0.25	0.59	0.33
Synchronous	0.61	0.54	0.27	0.04	0.74	0.41	0.31	0.02	N/A	N/A

Method	Page Blocks		Crowdsourced		Letter Recognition		warpPIE10P		COIL20	
	NMI	F-measure	NMI	F-measure	NMI	F-measure	NMI	F-measure	NMI	F-measure
DBADV	**0.36**	**0.44**	**0.45**	**0.56**	**0.61**	0.30	**0.78**	**0.64**	**0.94**	0.86
DBSCAN	0.24	0.26	0.38	0.23	0.58	0.29	0.60	0.09	0.62	0.19
OPTICS	0.22	0.17	0.18	0.05	0.44	0.02	0.68	0.30	0.87	0.67
HDBSCAN	0.19	0.29	0.33	0.16	0.48	0.24	0.69	0.32	0.91	**0.88**
CRAD	0.10	0.29	N/A	N/A	N/A	N/A	0.41	0.06	0.00	0.00
DBSCAN-DScale	0.27	0.32	0.38	0.23	0.60	**0.34**	0.68	0.40	0.93	0.80
SpectACl	0.20	0.41	0.41	0.48	0.31	0.24	N/A	N/A	0.24	0.10
k-means	0.13	0.26	0.26	0.32	0.35	0.26	0.30	0.30	0.77	0.57
Spectral	0.14	0.31	0.18	0.30	0.27	0.17	0.08	0.03	0.06	0.03
Self-tuning	N/A	N/A	0.32	0.27	N/A	N/A	0.62	0.29	0.49	0.26
Affinity Propagation	0.13	0.36	0.23	0.26	0.58	0.15	0.64	0.49	0.76	0.50
Synchronous	0.19	0.33	0.28	0.39	0.30	0.17	0.02	0.04	0.38	0.06

find the mutual nearest neighbors of each object to detect objects around boundaries between clusters more efficiently. Moreover, we give an effective heuristic to determine the primary parameters by fixing cumulative probability p_{cum} and only tuning $MinPts$ and $perp$, both of which share the same reasonable search range. Extensive experiments on challenging synthetic and real-world datasets have demonstrated that the proposed algorithm is effective and outperforms other state-of-the-art algorithms.

Chapter 4

Adaptive Density Outlier Detection

Outlier detection plays a dual role in data analysis: cleansing data to optimize the performance of downstream tasks and identifying potentially rare valuable events or patterns. Proximity-based methods, which are independent of data distribution assumptions, are plagued by parameter selection and performance challenges when handling datasets with varying densities. This chapter proposes a novel unsupervised algorithm named **A**daptive **D**ensity **O**utlier **D**etection (ADOD) to address these challenges. The core innovation of ADOD involves two main aspects: adaptive neighborhood boundaries and density consistency scoring. First, instead of relying on a predefined fixed radius, ADOD employs perplexity to calculate the local scale of each data point and dynamically adjusts the neighborhood boundaries according to this scale to adapt to data with varying densities. Second, ADOD estimates local density using a mutual nearest neighbor graph and combines the density differences between data points and their neighbors for outlier scores, effectively distinguishing outliers that significantly deviate from their surroundings. We evaluated ADOD on one synthetic and 32 real-world datasets and compared it with 14 classical and state-of-the-art algorithms from different categories. Extensive experimental results demonstrated the superior performance of ADOD, achieving the highest average ROC, P@N, and AP scores. This algorithm promotes the development of outlier detection techniques and expands their potential for real-time applications.

Parts of the material presented in this chapter have been published in [QQS+24]:

> *"Li Qian, Jing Qian, Xin Sun, Wengang Guo, Christian Böhm. ADOD: Adaptive Density Outlier Detection. 2024 IEEE International Conference on Data Mining (ICDM), pp. 400-409, 2024."*

where Li Qian was responsible for the development of the main concepts, the implementation of the algorithm, most of the experimental evaluation, and writing the majority of the paper; Jing Qian proposed the research idea of generalizing the algorithm to unknown data; Xin Sun and Wengang Guo were responsible for experiments on selected comparison algorithms and wrote the corresponding parts of the paper; Christian Böhm supervised and guided the research. All co-authors participated in the discussion of concept development and the revision of the manuscript.

4.1 Introduction

Outlier detection identifies rare and suspicious data points that significantly differ from the majority [ABC22]. On the one hand, for the majority fitting the expected pattern, outlier detection serves as data cleaning. Data quality can be improved by effectively identifying and removing outliers, thus ensuring the accuracy and reliability of downstream tasks such as predictive modeling and statistical analysis [RKV+21]. On the other hand, the few outliers often reveal valuable events or observations [AA17]. For example, in finance, outliers may indicate fraudulent transactions, leading to significant economic losses; in cybersecurity, outliers may imply network intrusion, resulting in private data leakage; in healthcare, outliers may indicate atypical diagnostic observations, resulting in missing the best time for treatment, or even endangering the patient's life [AA17]. Therefore, accurate and efficient outlier detection improves the quality of data analysis and aids industries in proactively responding to potential threats and seizing opportunities [SB21].

Outlier detection methods can generally be divided into five categories [AA17]: proximity-based, probabilistic-based, linear model-based, ensemble-based, and neural network-based methods [MWX+23]. Among them, proximity-based methods [ST23] have received considerable attention due to their high interpretability and independence from data distribution assumptions. Proximity-based methods offer several advantages [AASM21]. Firstly, these methods compute outlier scores through simple and intuitive proximity relationships, making them easier to understand and interpret. Secondly, unlike probabilistic-based methods, they do not rely on assumptions about statistical distributions, making them applicable to various data distributions. Additionally, compared to neural network-based and ensemble-based methods, proximity-based methods are less influenced by random values, resulting in more stable results. Proximity-based methods can be further divided into distance and density methods. Distance methods such as k-NN [RRS00] determine outliers by analyzing the distance to the k-th nearest neighbor, assuming that points farther away are more likely to be outliers. However, k-NN uses the same k for all points, which limits its adaptability to complex density variations. As shown in Fig. 4.1, k-NN incorrectly detects certain outliers as inliers, leading to more errors and irregular, scattered decision boundaries.

To alleviate this limitation, density methods such as Local Outlier Factor (LOF) [BKNS00] propose the concept of reachable distance, which is defined as the maximum distance between a point and its neighbor, and the k-distance of the neighbor. This distance metric adjusts for the difference in distances between points in regions with different densities and reduces the misclassification to a certain extent. LOF assesses outliers by calculating the local density of each point and its neighbors. However, its accuracy remains a concern in regions with significant density variations [AASM21]. As shown in Fig. 4.1, for boundary points with significant density variations, their nearest k neighbors are located in both high- and low-density regions. However, LOF incorrectly estimates their local densities, resulting in irregular decision boundaries. In addition, LOF and k-NN both rely on selecting the parameter k. Although k can be selected by methods such as greedy search, using the same k for each point cannot simultaneously

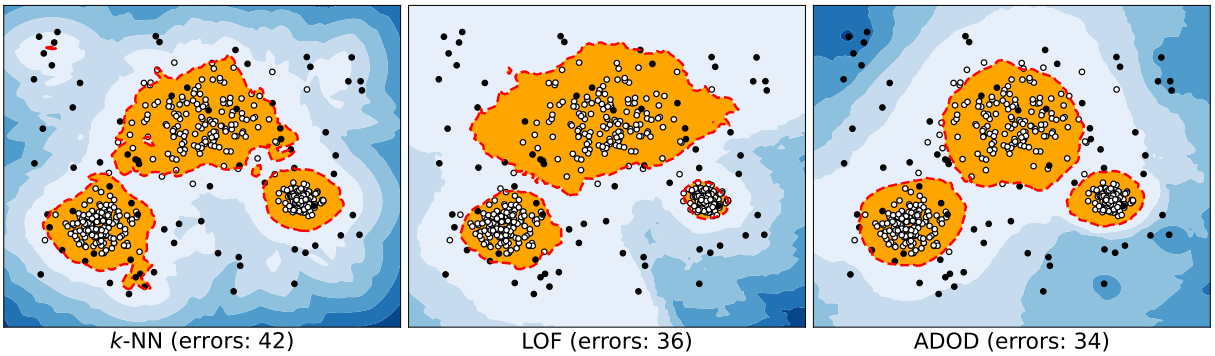


Figure 4.1: Decision boundaries comparison on synthetic *ThreeBlobs-Outlier* dataset using k-NN, LOF, and ADOD: Analysis on a synthetic dataset of 500 points from three Gaussian blobs ($\sigma = [0.6, 1.2, 0.3]$) with 15% uniformly distributed outliers, where white dots represent true inliers and black dots represent true outliers. Decision boundaries are depicted with predicted inliers (yellow), predicted outliers (blue), and boundaries (red dashed lines). Error counts are recorded for each algorithm.

adapt to various density variations in the data distribution. Therefore, these methods are usually ineffective in dealing with complex datasets and cannot meet the needs of practical applications.

We propose the Adaptive Density Outlier Detection (ADOD) algorithm to handle data with varying densities. Inspired by the concept of perplexity in t-SNE [vdMH08], ADOD adopts perplexity as a smoothing mechanism to measure the effective number of neighbors, thereby avoiding the constraints of predefined fixed k for each point. ADOD calculates the adaptive local scale of each data point by perplexity and dynamically adjusts the neighborhood boundary according to this scale. Based on these boundaries, it constructs a mutual nearest neighbor graph to estimate the local density and ensures inliers in regions with different densities have similar local densities. Finally, density consistency scoring is performed by combining the local densities of a data point and the density differences with its mutual neighbors, thus distinguishing the outliers that significantly deviate from their surroundings, as shown in Fig. 4.1. This simple and robust algorithm avoids tedious parameter tuning and exhibits excellent performance on data from various domains.

The primary contributions include:

1. We propose ADOD, one of the first outlier detection algorithms to leverage perplexity for density estimation, enabling adaptation to varying data densities.

2. We enhance the efficiency and scalability of ADOD by optimizing it with nearest neighbor search, ensuring robust performance on large datasets.

3. We extend ADOD to generalize to unknown data, enabling effective comparisons with known data and expanding its applicability to real-time scenarios.

4.2 Related Work

This section provides a concise overview of the five categories of outlier detection methods and discusses classic and state-of-the-art algorithms for each category.

Probabilistic-based methods identify outliers by building statistical models of data and then calculating the probabilistic deviation of data points [AA17]. For example, Angle-Based Outlier Detection (ABOD) [KSZ08] calculates the angular difference between each data point and all other points, whereas FastABOD [KSZ08] simplifies the computation through nearest neighbor search (NNS) [JDS11]. Stochastic Outlier Selection (SOS) [JHPvdH12] uses affinity and binding probabilities between data points to generate random neighbor graphs but relies on the parameter perplexity and has high memory requirements. Copula-based Outlier Detection (COPOD) [LZB+20] and Empirical-Cumulative-distribution-based Outlier Detection (ECOD) [LZH+23] assess the outlierness of data points using an empirical cumulative distribution function. COPOD models variable dependencies using Copula models, whereas ECOD focuses on the tail probabilities of each dimension. Both methods may be limited in real-time environments due to the lack of relationship exploitation between known and unknown data.

Linear model-based methods identify outliers by projecting data into a new space and using linear transformations [AA17]. For example, Principal Component Analysis

(PCA) [SCSC03] assumes linear relationships and projects data to a low-dimensional space by analyzing the principal and secondary components, identifying points that deviate from the principal components as outliers. Kernel PCA (KPCA) [Hof07] projects data to a high-dimensional feature space through kernel functions, extracting principal components for outlier detection. However, KPCA is computationally expensive and unsuitable for large data. One-Class Support Vector Machines (OCSVM) [SPST+01] construct a hyperplane in a high-dimensional space, assuming that inliers form a tight group and outliers lie outside this group. However, OCSVM is sensitive to parameter selection and has a high computational cost.

Ensemble-based methods improve robustness and accuracy by combining results from multiple base models [AA17]. For example, Feature Bagging (FB) [LK05] combines multiple detections based on random feature subsets but is sensitive to subset selection. Isolation Forest (IF) [LTZ08] constructs random tree structures to isolate data points using average path lengths. Deep Isolation Forest (DIF) [XPWW23] uses deep learning techniques to generate random representations and axis-parallel partitions. However, these representations may not efficiently capture outlier patterns in high-dimensional and complex data, resulting in unstable performance. Locally Selective Combination of Parallel Outlier Ensembles (LSCP) [ZNHL19] dynamically selects the optimal detector in a local region, with its effectiveness depending on the local region definition and detector selection.

Neural network-based methods employ deep learning models to capture complex nonlinear relationships and identify outliers [MWX+23]. For example, Single-Objective Generative Adversarial Active Learning (SO-GAAL) [LLZ+20] generates potential outliers through a single generator and improves data discrimination boundaries by iteratively optimizing a generative adversarial network (GAN). Still, it may encounter the mode collapse problem. Multiple-Objective GAAL (MO-GAAL) [LLZ+20] extends SO-GAAL by using multiple generators to enhance data diversity and reduce mode collapse risk, but at a higher computational cost. Adversarially Learned Anomaly Detection (ALAD) [ZRF+18] combines reconstruction errors with features obtained through adversarial learning us-

ing a bidirectional GAN. Unifying Local Outlier Detection Methods via Graph Neural Networks (LUNAR) [GHNN22] employs graph neural networks to generate learnable outlier scores for message passing and aggregation within local neighborhoods. However, its performance depends on hyperparameter selection.

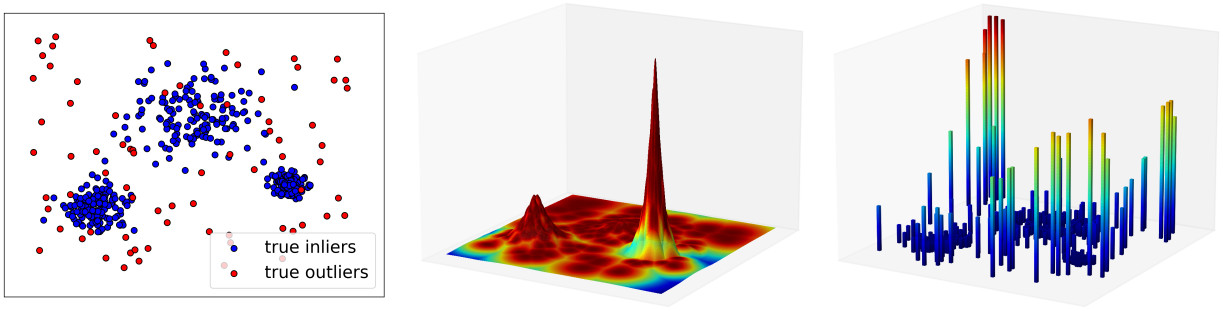
Proximity-based methods identify outliers by measuring the distance or density difference between a data point and its neighbors [AA17]. For example, k-NN [RRS00] and its variants AvgKNN and MedKNN [AP02] identify outliers by analyzing the k-th nearest distances: k-NN considers the exact k-th distance, AvgKNN averages these distances, and MedKNN takes the median. All methods are sensitive to the parameter k and are affected by density variations. LOF [BKNS00] computes the local density of each data point and compares it with its neighbors, which is also sensitive to parameter selection. Connectivity-based Outlier Factor (COF) [TCFC02], a variant of LOF, evaluates outlier scores by calculating the average chained distance between the data point and its neighbors but requires high memory.

The proposed ADOD algorithm is a proximity-based method. It determines the adaptive local density of each data point by perplexity and makes adjustments according to the local densities of its mutual neighbors. ADOD avoids dependence on a fixed k, improves robustness and applicability, performs well with varying densities and large data, and quickly adapts to unknown data for real-time processing.

4.3 Methodology

4.3.1 Problem Definition

We aim to perform unsupervised outlier detection in a multidimensional dataset. The dataset is defined as $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \in \mathbb{R}^{n \times d}$, where each data point $\mathbf{x}_i \in \mathbb{R}^d$ is a vector in d-dimensional space, and n denotes the number of data points. The objective is to compute an outlier score for each data point, denoted as s_i for $\mathbf{x}_i$, forming the outlier score set $S = \{s_1, s_2, \ldots, s_n\} \subseteq \mathbb{R}$. A higher score indicates a higher probability of being



(a) (b) (c)

Figure 4.2: Illustration of key ADOD steps on synthetic *ThreeBlobs-Outlier* dataset. (a) Ground truth; (b) Gaussian probability density via adaptive local scales (higher peaks indicate denser regions); (c) Outlier scores (higher bars indicate a greater likelihood of being an outlier).

an outlier.

4.3.2 Algorithm Description

Dissimilarity Matrix Calculation

The first step in ADOD is to calculate the dissimilarity matrix D, which is the basis for analyzing the interrelationships between data points in the dataset X. The dissimilarity matrix D is an $n \times n$ matrix, where each element D_{ij} represents the Euclidean distance between the i-th and j-th data points in X, formulated as follows:

$$D_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2 \tag{4.1}$$

where $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^d$ denote vectors in d-dimensional space. The diagonal element D_{ii} of the dissimilarity matrix is zero, and the symmetry of the matrix ($D_{ij} = D_{ji}$) ensures that the distance relationships are mutual.

Adaptive Local Scale Estimation

We use the probability density function of the Gaussian distribution [Mur22] to define the influence of the neighboring data point $\mathbf{x}_j$ on the central data point $\mathbf{x}_i$ and quantify

it using the conditional probability $p_{j|i}$ with the following expression:

$$p_{j|i} = \frac{\exp(-D_{ij}^2/2\sigma_i^2)}{\sum_{l\neq i}\exp(-D_{il}^2/2\sigma_i^2)} \tag{4.2}$$

where σ_i represents the local scale of $\mathbf{x}_i$. Based on σ_i, we can adaptively adjust the neighborhood boundary of $\mathbf{x}_i$ to reflect changes in local density. The bell-shaped curve of the Gaussian distribution provides a natural weight decay mechanism [Mur22]. This causes data points further from the centroid to exhibit a gradually decreasing effect, which is suitable for quantifying the strength of interactions between data points, as shown in Fig. 4.2(b).

In data analysis, perplexity is commonly used as a smoothing mechanism to measure the effective number of neighbors, excelling in density estimation. It has been widely applied in techniques such as t-SNE [vdMH08] for dimensionality reduction, DBADV [QPB21] for clustering (described in Chapter 3), and SOS [JHPvdH12] for outlier detection. In contrast to SOS, which uses perplexity to assess the affinity between data points and requires careful tuning, ADOD uses perplexity to obtain the adaptive local scale σ_i of $\mathbf{x}_i$, improving the adaptability to datasets with varying densities. Perplexity is defined as follows:

$$\text{Perplexity}(p_i) = 2^{H(p_i)} \tag{4.3}$$

where, $H(p_i)$ is the entropy of the conditional probability distribution centered on $\mathbf{x}_i$, which is used to quantify the uncertainty of the probability distribution. A higher $H(p_i)$ indicates a more uniform probability distribution within the neighborhood of $\mathbf{x}_i$, implying minimal density differences. Conversely, a lower entropy signifies significant density differences within the neighborhood. Its formula is as follows:

$$H(p_i) = -\sum_j p_{j|i}\log_2 p_{j|i} \tag{4.4}$$

The local scale σ_i is continuously adjusted by fixing the perplexity and using a binary

search [vdMH08] to ensure that the neighborhood complexity (i.e., the number of effective neighbors) is balanced for each data point. Thus, it adaptively reflects the true relationships and density variations of the data points in their local surroundings. Although no explicit guidance is provided for setting the perplexity, inspired by the rule of thumb in k-NN [DHS12], it is often recommended to set k proportional to $\sqrt{n}$ [İnk23]. Similarly, we set the perplexity proportional to $\sqrt{n}$ to ensure reliable density estimation over sufficient neighbors while avoiding unnecessary computational costs associated with excessively large neighborhoods. Experiments on parameter sensitivity analysis in Section 4.4 verified the effectiveness of this setting, simplifying parameter selection and enhancing adaptability to data size.

Adaptive Neighborhood Boundary Determination

ADOD employs the quantile function [Gil00] to determine an adaptive neighborhood boundary for each data point. By dynamically adjusting the boundary based on the local scale of each point, the method ensures that neighborhood sizes adapt to the data distribution, capturing points with relatively consistent densities around the given point. The quantile function, as the inverse of the CDF [DFO20], calculates a value such that the probability of a random variable being less than or equal to this value matches the specified cumulative probability. For a Gaussian distribution with mean μ and standard deviation σ, the quantile function is expressed as follows:

$$F^{-1}(p_{\text{cum}}) = \mu + \sqrt{2}\sigma \ \text{erf}^{-1}(2p_{\text{cum}} - 1) \tag{4.5}$$

where $\text{erf}^{-1}(\cdot)$ is the inverse of the error function [SE14], and p_{cum} is the cumulative probability.

In our algorithm, the neighborhood boundary of each data point $\mathbf{x}_i$ is determined using the quantile function with a local Gaussian distribution centered at $\mathbf{x}_i$. Specifically, the boundary ϵ_i is defined as follows:

$$\epsilon_i = \sqrt{2}\sigma_i \, \mathrm{erf}^{-1}(2p_{\text{cum}} - 1) \tag{4.6}$$

where σ_i represents the local scale of $\mathbf{x}_i$, and p_{cum} is the specified cumulative probability that determines the boundary. To ensure meaningful neighborhood boundaries, p_{cum} is constrained within the range $(0.5, 1)$. Here, p_{cum} is set to 0.999, corresponding to the three-sigma rule [Gil00]. This setting ensures that almost all data points are included in the neighborhood of each data point while effectively identifying and isolating potential outliers that deviate significantly. Experiments on parameter sensitivity analysis in Section 4.4.2 verified the effectiveness of this configuration.

Mutual Nearest Neighbor Graph Construction

In the ADOD algorithm, the mutual nearest neighbor graph (MNNG) depicts the adjacency between data points. This mutual connection reflects the interactions between data points and helps identify potential outliers. The MNNG G is an undirected graph consisting of a node set V and an edge set E. The node set V contains all nodes corresponding to data points in the dataset, where each node $v_i \in V$ uniquely represents a data point $\mathbf{x}_i \in X$. The edge set E includes all pairs of nodes (v_i, v_j) that satisfy the mutual nearest neighbor condition: the node v_j is within the neighborhood boundary ϵ_i of $\mathbf{x}_i$, and simultaneously, v_i is within the neighborhood boundary ϵ_j of $\mathbf{x}_j$. To ensure that each node pair is considered only once and to avoid self-loops, we define the set of mutual neighborhood edges as follows, where $i < j$ ensures that each node pair is unique:

$$E = \{(v_i, v_j) \mid D_{ij} \leq \min(\epsilon_i, \epsilon_j), i < j\} \tag{4.7}$$

The MNNG depicts the adjacency relationships between data points. This mutual relationship reflects the interactions between data points and helps identify potential outliers.

The neighborhood of a node v_i, denoted as N_i, is obtained from the edge set E of the

MNNG as follows:

$$N_i = \{v_j \mid (v_i, v_j) \in E\} \tag{4.8}$$

Local Density Estimation

The local density d_i of $\mathbf{x}_i$ quantifies the number of neighbors within its adaptive neighborhood boundary ϵ_i, reflecting the density distribution in that region. It is defined as follows:

$$d_i = \frac{\deg(v_i) + 1}{\epsilon_i} \tag{4.9}$$

where ϵ_i is the adaptive neighborhood boundary of $\mathbf{x}_i$, and $\deg(v_i)$ denotes the degree of the corresponding node v_i in G. The "+1" ensures that $\mathbf{x}_i$ itself is included in the density estimation, providing a baseline density for isolated points. In dense regions, inliers have a relatively small neighborhood boundary with a lower degree, whereas in sparse regions, they exhibit a larger neighborhood boundary with a higher degree. Therefore, even if these inliers are in regions with different densities, their local density results remain comparable, effectively preventing misclassification due to density differences. For outliers, which typically have larger neighborhood boundaries but lower degrees, the local density is significantly lower than that of inliers.

Density Consistency Outlier Scoring

The outlier score is calculated based on the combination of the local density of the data point and the local density differences with its mutual neighbors to identify potential outliers. The outlier score s_i of $\mathbf{x}_i$ is defined as follows:

$$s_i = d_i^{-1} + \sum_{j \in N_i} w'_{ij}(d_j^{-1} - d_i^{-1}) \tag{4.10}$$

where d_i^{-1} and d_j^{-1} denote the inverse of the local density of $\mathbf{x}_i$ and its neighbor $\mathbf{x}_j$, re-

spectively. The inverse of local density reflects the relative sparsity of a data point, where a higher value implies lower density. Consequently, such points typically gain higher outlier scores than inliers. Furthermore, by calculating $d_j^{-1} - d_i^{-1}$, ADOD quantifies the density difference between each data point and its mutual neighbors, thereby identifying those points that deviate from their surroundings. If the neighbor of $\mathbf{x}_i$ is in a sparser surrounding compared to $\mathbf{x}_i$, $d_j^{-1} - d_i^{-1} > 0$, then its outlier score further increases. Whereas, if it is in a denser surrounding, $d_j^{-1} - d_i^{-1} < 0$, the outlier score further decreases. Finally, if $\mathbf{x}_i$ and its neighbors are in regions of similar density, the outlier score remains stable.

The weights w_{ij} are calculated based on the Euclidean distance D_{ij} between $\mathbf{x}_i$ and $\mathbf{x}_j$, defined by the formula $w_{ij} = \frac{1}{D_{ij}}$. These weights are then normalized to ensure that the total sum of all neighboring weights equals 1, using the formula $w'_{ij} = \frac{w_{ij}}{\sum_{l \in N_i} w_{il}}$. This process reflects the distance relationship between each neighbor point and the current point (i.e., the closer the distance, the greater the influence) and balances the contribution of each neighbor to the outlier score, as shown in Fig. 4.2(c). Through this density-consistent scoring mechanism, the ADOD algorithm not only takes into account the local density of each data point but also integrates the information of its mutual neighbors, providing an effective way to detect outlier patterns in the data.

4.3.3 Algorithm Overview

The ADOD algorithm is designed to identify potential outliers in the dataset, and its pseudocode is presented in Algorithm 3. The algorithm accepts a dataset X and outputs outlier scores S for all data points. It begins by calculating the Euclidean distances between all pairs of data points to form a dissimilarity matrix D, where D_i is a vector containing the distances between the data point $\mathbf{x}_i$ and all other points in X (line 2). For $\mathbf{x}_i$, an adaptive local scale σ_i is determined by a binary search based on perplexity $perp$ and D_i. The adaptive neighborhood boundary ϵ_i is calculated based on σ_i and the predetermined cumulative probability p_{cum} (lines 3-5). An MNNG G is constructed, where the edge set E includes only pairs of data points that lie within the neighborhood boundary

Algorithm 3: ADOD

Input: Dataset $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathbb{R}^{n \times d}$
Output: Outlier scores $S = \{s_1, \ldots, s_n\} \in \mathbb{R}^n$

1 Initialize perplexity $perp = 2 \cdot \lfloor \sqrt{n} \rfloor$, cumulative probability $p_{\text{cum}} = 0.999$
2 Calculate dissimilarity matrix D, where $D_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2$
3 **foreach** $\mathbf{x}_i \in X$ **do**
4 Calculate adaptive local scale σ_i: $\sigma_i \leftarrow \text{binary\_search}(D_i, perp)$
5 Determine adaptive neighborhood boundary ϵ_i: $\epsilon_i = \sqrt{2}\sigma_i \, \text{erf}^{-1}(2p_{\text{cum}} - 1)$
6 Initialize $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ and $E = \emptyset$
7 **for** $i, j \in \{1, \ldots, n\}, i < j$ **do**
8 **if** $D_{ij} \leq \min(\epsilon_i, \epsilon_j)$ **then**
9 $E = E \cup \{(v_i, v_j)\}$
10 **foreach** $v_i \in V$ **do**
11 Estimate local density $d_i = \frac{\deg(v_i)+1}{\epsilon_i}$
12 **foreach** $v_i \in V$ **do**
13 $N_i = \{v_j \mid (v_i, v_j) \in E\}$
14 **foreach** $v_j \in N_i$ **do**
15 $w_{ij} = \frac{1}{D_{ij}}$
16 Normalize weights: $w'_{ij} = \frac{w_{ij}}{\sum_{v_l \in N_i} w_{il}}$
17 $s_i = d_i^{-1} + \sum_{v_j \in N_i} w'_{ij}(d_j^{-1} - d_i^{-1})$
18 **return** S

of each other (lines 6-9). The local density d_i of $\mathbf{x}_i$ is estimated based on the degree of the corresponding node v_i in the graph, which reflects the number of connections to neighbors within the neighborhood boundary ϵ_i (lines 10-11). Finally, the outlier score for each data point is calculated based on the local density and normalized weights of the neighbors w'_{ij} (lines 12-17). ADOD can effectively identify and quantify outlier patterns in data through this density-consistent scoring mechanism, thus rendering it adaptive and robust across various data properties.

4.3.4 Generalization to Unknown Data

In data analysis, unknown data refers to data points added after the initial training or model construction, which may differ from the known dataset observed during model

initialization. To effectively detect outliers among these unknown data points, an outlier detection algorithm should accurately identify known data patterns and flexibly adapt to unknown patterns [AA17]. An efficient outlier detection algorithm should leverage the known model and data structure to comprehensively evaluate the degree of outlierness, including comparing the unknown data points with known ones regarding distance or dissimilarity and computing the corresponding outlier scores.

Algorithm 4 details how ADOD evaluates potential outliers in an unknown dataset. It receives an unknown dataset $X^{\text{unk}} = \{\mathbf{x}_1^{\text{unk}}, \ldots, \mathbf{x}_m^{\text{unk}}\} \in \mathbb{R}^{m \times d}$ with m data points, and a known dataset $X^{\text{kn}} = \{\mathbf{x}_1^{\text{kn}}, \ldots, \mathbf{x}_n^{\text{kn}}\} \in \mathbb{R}^{n \times d}$ with n data points. For each unknown data point $\mathbf{x}_i^{\text{unk}}$, the algorithm first computes the Euclidean distance between $\mathbf{x}_i^{\text{unk}}$ and all points in the known dataset, forming the dissimilarity vector D_i^{unk} (line 3). Next, based on D_i^{unk} and the same perplexity $perp$ as the known dataset, the adaptive local scale σ_i^{unk} for $\mathbf{x}_i^{\text{unk}}$ is determined by a binary search. Then adaptive neighborhood boundary ϵ_i^{unk} is calculated based on σ_i^{unk} and the predetermined cumulative probability p_{cum} (lines 4-5). Subsequently, a set of mutual neighbors N_i^{unk} of $\mathbf{x}_i^{\text{unk}}$ is constructed to include the known data points that satisfy the mutual nearest neighbor conditions (lines 6-9). The local density d_i^{unk} of $\mathbf{x}_i^{\text{unk}}$ is determined by the number of neighbors within its adaptive neighborhood boundary ϵ_i^{unk} (line 10). The outlier score s_i^{unk} is computed using the normalized neighbor weights and density differences, which quantify the outlierness of the unknown data points with the known data points (lines 11-14).

4.3.5 Complexity Analysis

The time complexity of the ADOD algorithm without optimization is analyzed as follows: First, the adaptive local scale estimation is mainly affected by the dissimilarity matrix computation, calculating the Euclidean distance between all pairs of data points. The time complexity is $O(n^2)$. Next, the time complexity of the adaptive neighborhood boundary determination step is $O(n)$. Mutual nearest neighbor graph construction necessitates the comparison of every pair of data points with a time complexity of $O(n^2)$. The

Algorithm 4: ADOD for Unknown Data

Input: Unknown dataset $X^{\text{unk}} = \{\mathbf{x}_1^{\text{unk}}, \ldots, \mathbf{x}_m^{\text{unk}}\} \in \mathbb{R}^{m \times d}$, Known dataset $X^{\text{kn}} = \{\mathbf{x}_1^{\text{kn}}, \ldots, \mathbf{x}_n^{\text{kn}}\} \in \mathbb{R}^{n \times d}$

Output: Outlier scores $S^{\text{unk}} = \{s_1^{\text{unk}}, \ldots, s_m^{\text{unk}}\} \in \mathbb{R}^m$

Initialize perplexity $perp = 2 \cdot \lfloor \sqrt{n} \rfloor$, cumulative probability $p_{\text{cum}} = 0.999$

foreach $\mathbf{x}_i^{unk} \in X^{unk}$ **do**

  Calculate dissimilarity vector D_i^{unk} for $\mathbf{x}_i^{\text{unk}}$ with each $\mathbf{x}_j^{\text{kn}}$ in X^{kn}: $D_{ij}^{\text{unk}} = \|\mathbf{x}_i^{\text{unk}} - \mathbf{x}_j^{\text{kn}}\|_2$ for $j = 1, \ldots, n$

  Calculate adaptive local scale σ_i^{unk}: $\sigma_i^{\text{unk}} = \text{binary\_search}(D_i^{\text{unk}}, perplexity)$

  Determine adaptive neighborhood boundary ϵ_i^{unk}: $\epsilon_i^{\text{unk}} = \sqrt{2}\sigma_i^{\text{unk}} \, \text{erf}^{-1}(2p_{\text{cum}} - 1)$

  Initialize neighbor set $N_i^{\text{unk}} = \emptyset$

  for $j \in \{1, \ldots, n\}$ **do**

    if $D_{ij}^{unk} \leq \min(\epsilon_i^{unk}, \epsilon_j^{kn})$ **then**

      $N_i^{\text{unk}} = N_i^{\text{unk}} \cup \{\mathbf{x}_j^{\text{kn}}\}$

  Estimate local density $d_i^{\text{unk}} = \frac{|N_i^{\text{unk}}|+1}{\epsilon_i^{\text{unk}}}$

  foreach $\mathbf{x}_j^{kn} \in N_i^{unk}$ **do**

    $w_{ij}^{\text{unk}} = \frac{1}{D_{ij}^{\text{unk}}}$

  Normalize weights: $w_{ij}^{\text{unk}\prime} = \frac{w_{ij}^{\text{unk}}}{\sum_{\mathbf{x}_l^{\text{kn}} \in N_i^{\text{unk}}} w_{il}^{\text{unk}}}$

  $s_i^{\text{unk}} = \frac{1}{d_i^{\text{unk}}} + \sum_{\mathbf{x}_j^{\text{kn}} \in N_i^{\text{unk}}} w_{ij}^{\text{unk}\prime} \left(\frac{1}{d_j^{\text{kn}}} - \frac{1}{d_i^{\text{unk}}} \right)$

return S^{unk}

local density estimation iterates over all data points with a time complexity of $O(n)$. Finally, the density consistency outlier scoring step exhibits a time complexity of $O(n \cdot k_{avg})$, where k_{avg} is the average number of mutual neighbors used to compute the density difference between each data point and its mutual neighbors.

In summary, the overall time complexity of the ADOD algorithm without optimization is $O(n^2)$. For the evaluation of unknown data, the time complexity is similar to that of the above steps, and the overall time complexity is $O(m \cdot n)$, where m denotes the number of unknown data points.

4.3.6 Efficiency Optimization

We optimize the efficiency of the ADOD algorithm using NNS [JDS11]. This method enables the algorithm to focus on the primary neighbors of each data point rather than calculating all pairwise distances, significantly increasing processing speed and reducing memory usage. As previously discussed, the number of neighbors is typically set proportional to $\sqrt{n}$ [DHS12]. Following the setting used in t-SNE [vdMH08], where the perplexity, a smooth measure of the effective number of neighbors, is multiplied by 3 to determine the neighbor number, we set the number of neighbors for NNS to $3\sqrt{n}$. This setting ensures that sufficient local information is captured while maintaining high computational efficiency. When evaluating unknown data points, NNS simplifies the querying of neighbors between the unknown and known data points. This strategy accelerates the outlier detection process and enables the algorithm to adapt to real-time applications.

4.4 Experiments

4.4.1 Experimental Setup

Experimental Environment

The proposed DynoGraph algorithm was implemented in Python, utilizing `faiss-gpu`, the GPU implementation of the Faiss library [JDJ19], for nearest neighbor search. All experiments related to this algorithm were conducted on a machine equipped with an Intel Core i9-12900H CPU (14 cores, 2.50 GHz), 64 GB of RAM, and an NVIDIA GeForce RTX 3080 Ti GPU. The code repository is available at `https://github.com/Qian-Lily/ADOD`.

Datasets

Table 4.1 summarizes the 32 commonly used real-world datasets for outlier detection. These datasets are sourced from two repositories: 20 from ODDS [Ray16] and 12 from

ADBench [HHH+22]. They cover various domains, including healthcare, chemistry, physics, linguistics, finance, astronomy, web, image processing, and sociology. The total number of samples ranges from 80 to 567,498, the number of samples after deduplication ranges from 80 to 286,048, the number of dimensions ranges from 3 to 500, and the percentage of outliers after deduplication ranges from 0.03% to 34.90%. To ensure data uniqueness, we removed duplicate entries before conducting the experiments. Since the outlier detection algorithms in this chapter are unsupervised, we used the entire dataset without splitting. We preprocessed each dataset by applying the `StandardScaler`[33] from the `scikit-learn` library, which standardizes each attribute by centering it to a mean of 0 and scaling it to a unit standard deviation of 1. In these datasets, the label '1' denotes outliers, while the label '0' denotes inliers.

Comparison Methods

To comprehensively evaluate the performance of ADOD, we selected 14 representative and state-of-the-art outlier detection algorithms for comparison. These included probabilistic-based methods such as ECOD [LZH+23], FastABOD [KSZ08], and SOS [JHPvdH12]; linear model-based methods such as KPCA [Hof07] and OCSVM [SPST+01]; proximity-based methods such as LOF [BKNS00], COF [TCFC02], and k-NN [RRS00]; ensemble-based methods such as DIF [XPWW23], FB [LK05], and LSCP [ZNHL19]; and neural network-based methods such as MO-GAAL [LLZ+20], ALAD [ZRF+18], and LUNAR [GHNN22]. The implementation code for these algorithms is available in the PyOD library[34] [ZNL19], a Python toolbox specifically designed for outlier detection. All comparison methods were configured with the default parameters provided in PyOD, except for LSCP, which was configured using the PyOD example settings[35], where LOF was employed as the `detector_list` with `n_neighbors` set to $\{15, 20, 25, 30\}$. For ADOD, we fixed the perplexity $perp$ to $2\sqrt{n}$ and the cumulative

[33] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`

[34] `https://github.com/yzhao062/pyod`

[35] `https://github.com/yzhao062/pyod/blob/master/examples/lscp_example.py`

Table 4.1: Statistics of the 32 real-world datasets used in ADOD. Datasets with "_OD" and "_AD" suffixes are from the ODDS and ADBench repositories, respectively.

Dataset	#Total	#Unique	#Dimensions	%Outliers	Category
Hepatitis_AD	80	80	19	16.25	Healthcare
wine_OD	129	129	13	7.75	Chemistry
lympho_OD	148	148	18	4.05	Healthcare
WPBC_AD	198	198	33	23.74	Healthcare
Stamps_AD	340	340	9	9.12	Document
WDBC_AD	367	367	30	2.72	Healthcare
wbc_OD	378	377	30	5.31	Healthcare
arrhythmia_OD	452	452	274	14.60	Healthcare
pima_OD	768	768	8	34.90	Healthcare
vowels_OD	1456	1452	12	3.17	Linguistics
cardio_OD	1831	1822	21	9.60	Healthcare
musk_OD	3062	3062	166	3.17	Chemistry
Waveform_AD	3443	3443	21	2.90	Physics
speech_OD	3686	3686	400	1.65	Linguistics
thyroid_OD	3772	3656	6	2.54	Healthcare
PageBlocks_AD	5393	5393	10	9.46	Document
satimage-2_OD	5803	5801	36	1.19	Astronautics
satellite_OD	6435	6435	36	31.64	Astronautics
pendigits_OD	6870	6870	16	2.27	Image
annthyroid_OD	7200	7062	6	7.56	Healthcare
mnist_OD	7603	7603	100	9.21	Image
mammography_OD	11183	7848	6	3.22	Healthcare
magic_gamma_AD	19020	18905	10	34.77	Physics
campaign_AD	41188	41176	62	11.27	Finance
shuttle_OD	49097	49097	9	7.15	Astronautics
smtp_OD	95156	71230	3	0.03	Web
backdoor_AD	95329	87020	196	2.16	Network
celeba_AD	202599	113983	39	2.55	Image
fraud_AD	284807	275661	29	0.17	Finance
cover_OD	286048	286048	10	0.96	Botany
census_AD	299285	223223	500	8.25	Sociology
http_OD	567498	221900	3	0.03	Web

probability p_{cum} to 0.999.

Evaluation Metric

We evaluated the performance by averaging the scores from 10 independent trials with different random values, using the ROC score, described in Chapter 2.3.5, P@N score, described in Chapter 2.3.3, and AP score, described in Chapter 2.3.6. These metrics, which compare the ground truth with the predicted scores, indicate better performance with higher values. Furthermore, we used the critical difference (CD) diagram [IFFW+19] to illustrate the statistical differences. This diagram visualizes the statistical comparisons using the Wilcoxon signed-rank test with Holm's correction, with a default p-value of 0.05.

4.4.2 Parameter Sensitivity Analysis

To evaluate the performance of ADOD under different parameter settings, we performed a sensitivity analysis on two primary parameters. Cumulative probability p_{cum} in the Quantile function is set as a multiple of sigma based on the three-sigma rule, with the set of candidate values $\{0.841, 0.933, 0.977, 0.994, 0.999\}$ corresponding to cumulative probability values of 1, 1.5, 2, 2.5 and 3 sigma, respectively. Perplexity $perp$ is used to measure the number of effective neighbors and is set as a multiple of $\sqrt{n}$ based on the scale of the dataset with a set of candidate values of $\{1, 2, 3, 4, 5\}$.

Fig. 4.3 shows the average ROC score performance of ADOD on 32 real-world datasets with different cumulative probability p_{cum} and perplexity $perp$ settings. The performance was optimal for $p_{\text{cum}} = 0.999$, with the ROC score peaking at 0.852 (marked with a yellow star) when perplexity was $2\sqrt{n}$, before gradually decreasing. Fig. 4.4 and Fig. 4.5 show the average P@N and AP score performances, respectively. Both metrics achieved optimal performance at $p_{\text{cum}} = 0.999$. For P@N, the highest score of 0.467 was observed at $3\sqrt{n}$ perplexity, with a slightly lower by 0.01 at $2\sqrt{n}$. Similarly, for AP, the highest score of 0.448 was observed at $3\sqrt{n}$ perplexity, with a slightly lower of 0.006 at $2\sqrt{n}$.

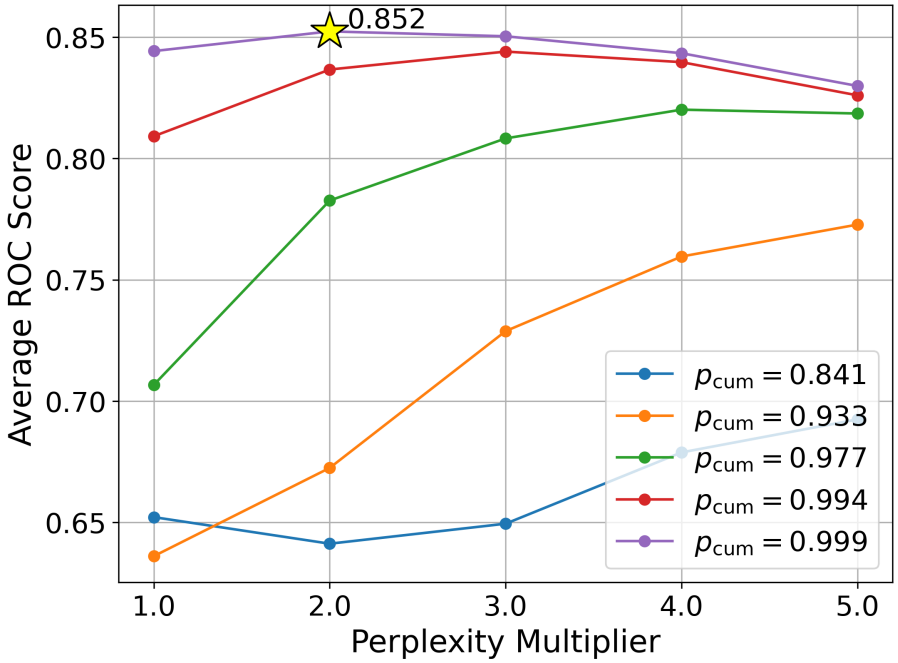


Figure 4.3: Average ROC score against different search ranges of parameters p_{cum} and $perp$ on real-world datasets.

Both metrics remained relatively stable between $2\sqrt{n}$ and $4\sqrt{n}$, with slightly inferior performance at $3\sqrt{n}$.

Overall, the cumulative probability setting significantly impacted performance. With $p_{\text{cum}} = 0.999$, the ROC score changed by 0.022, the P@N score changed by 0.059, and the AP score changed by 0.048. These results indicate that perplexity variations have a smaller impact on performance, demonstrating high robustness. Based on this analysis, we recommend setting the perplexity $perp$ to $2\sqrt{n}$ and the cumulative probability p_{cum} to 0.999. Throughout the experiments, we adopted these parameter settings as they consistently yielded competitive performance across different datasets.

4.4.3 Decision Boundaries Comparison

We generate a synthetic dataset *ThreeBlobs-Outlier* containing 500 data points, 15% of which are outliers, as shown in Fig. 4.6. The dataset consists of three Gaussian clusters (white dots) of different densities but the same number of points with standard devi-

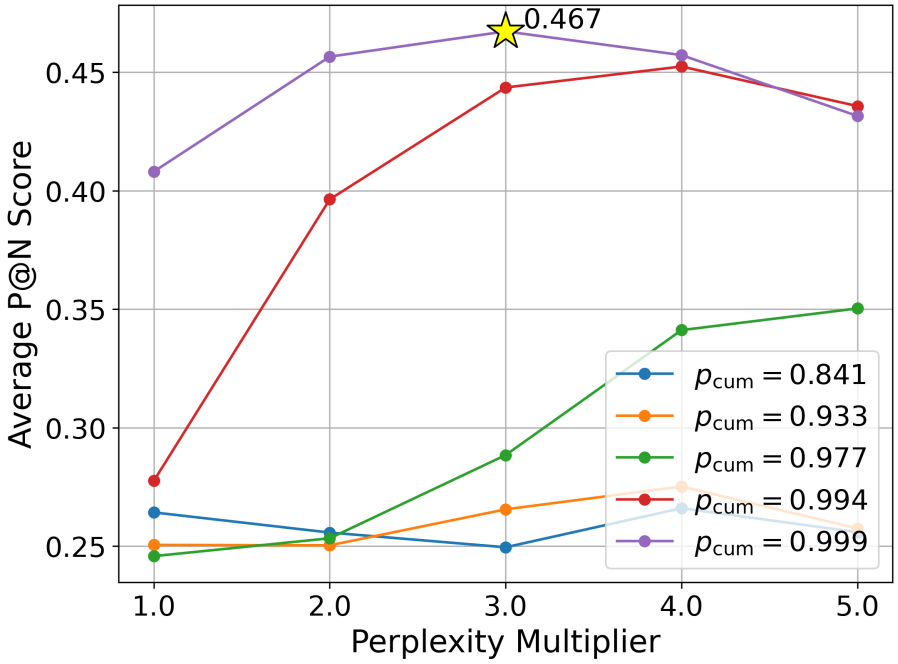


Figure 4.4: Average P@N score against different search ranges of parameters p_{cum} and $perp$ on real-world datasets.

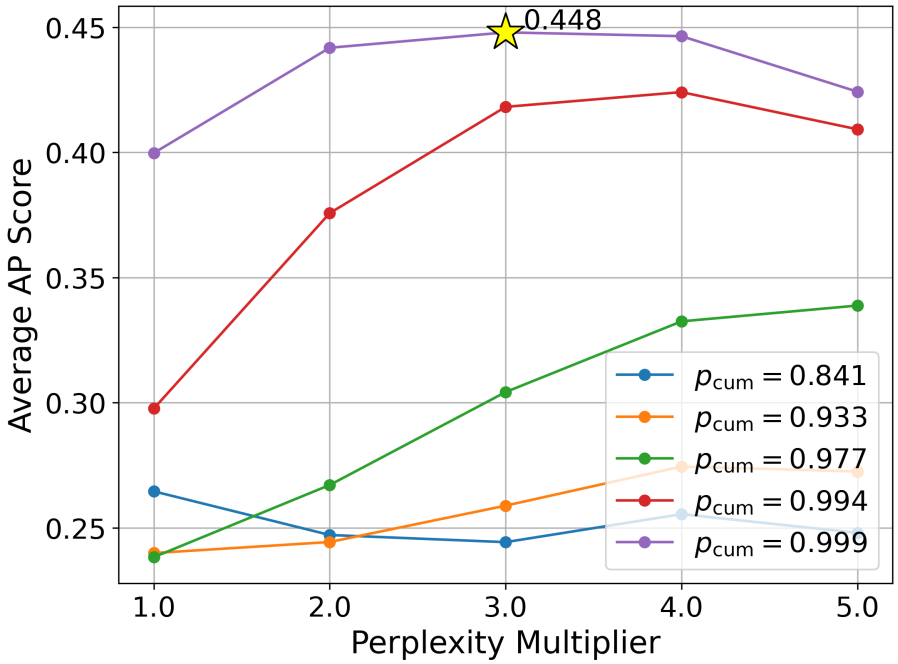


Figure 4.5: Average AP score against different search ranges of parameters p_{cum} and $perp$ on real-world datasets.

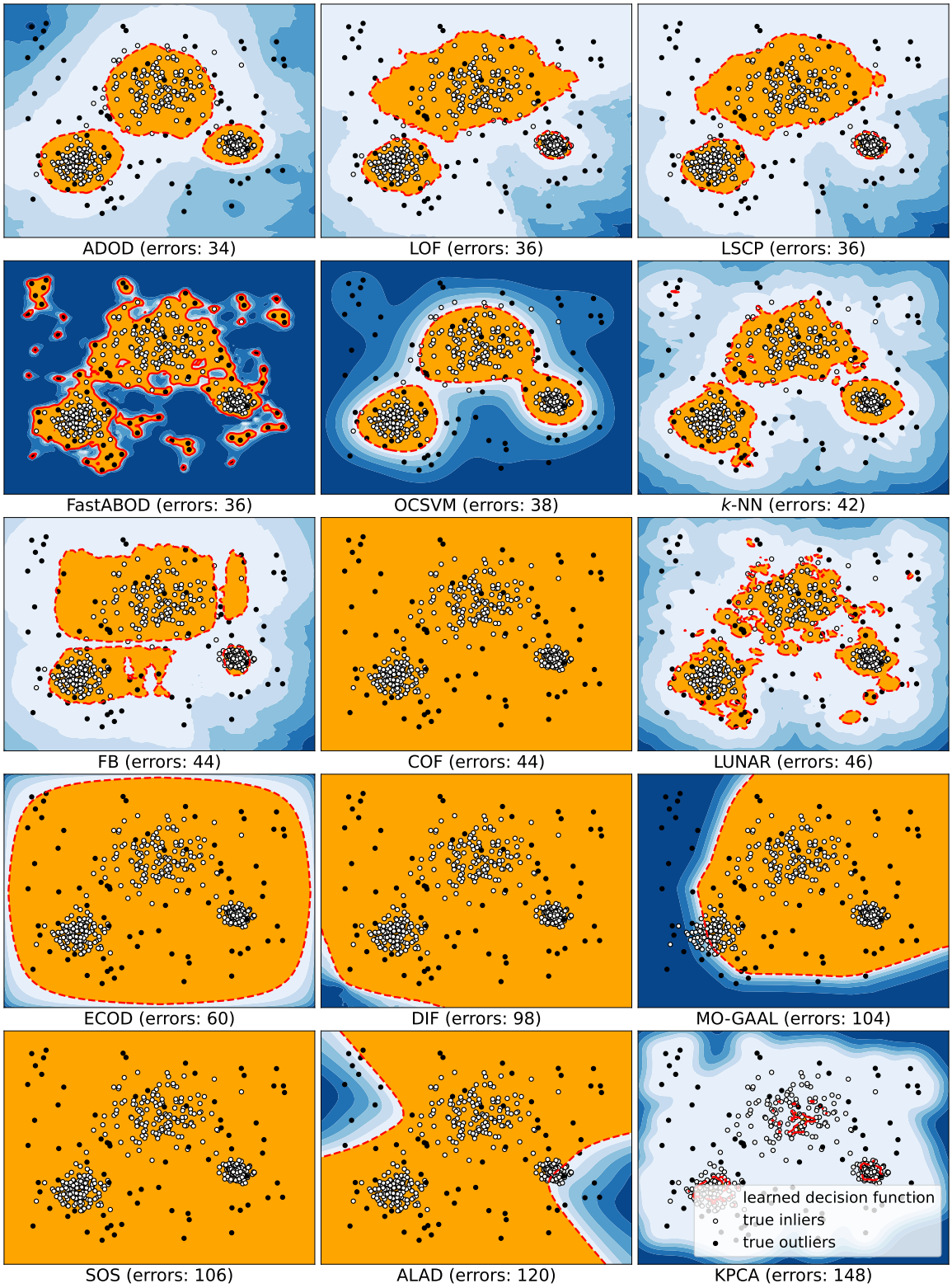


Figure 4.6: Decision boundaries comparison on synthetic *ThreeBlobs-Outlier* dataset using various outlier detection algorithms, sorted by error counts.

ations of $[0.6, 1.2, 0.3]$ and randomly distributed outliers (black dots). We trained the detector using the entire dataset and then computed decision function values on the entire 2D plane to draw decision boundaries. Decision function values are mapped to predicted inlier regions (yellow) and outlier regions (blue), and red dashed lines indicate decision boundaries. We also calculated the number of misclassifications (the number of predicted labels that do not match the true labels) for each algorithm, shown after the algorithm.

Fig. 4.6 shows the decision boundary and number of errors for various algorithms on *ThreeBlobs-Outlier* dataset. This experiment used ADOD for unknown data (Algorithm 4) to generate decision boundaries, where the dataset represents known data, and the entire 2D plane represents unknown data. The decision boundaries are depicted with yellow regions for inliers and blue regions for outliers, with the gradient of blue indicating the outlier score, where darker blue represents a higher outlier score. In summary, ADOD exhibited the lowest number of errors (34) and outperformed all baselines by accurately identifying three Gaussian clusters of different densities with clear decision boundaries. LOF, LSCP, and FastABOD exhibited relatively few errors (36) but featured irregular decision boundaries. LSCP presented almost the same results as LOF because LSCP is an ensemble-based method based on LOF detector. OCSVM, with 38 errors, displayed smooth decision boundaries but merged blobs of different densities into a single region. COF, ECOD, DIF, MO-GAAL, SOS, and ALAD failed to obtain reasonable decision boundaries. SOS, ALAD, and KPCA reported error numbers 106, 120, and 148, respectively. By calculating the number of errors, we demonstrated the performance of the algorithms on known data. Additionally, by drawing decision boundaries on a two-dimensional plane, we effectively highlighted the generalization ability of the algorithms on unknown data points within this plane.

Table 4.2: ROC score on real-world datasets: highest scores in bold; ranking in parentheses (lower is better). Notations: N/A (No Results), O/M (Out-of-Memory, >64GB), O/T (Out-of-Time, >12h).

Dataset	ECOD	FastABOD	SOS	KPCA	OCSVM	LOF	COF	k-NN	DIF	FB	LSCP	MO-GAAL	ALAD	LUNAR	ADOD
Hepatitis	0.739(4)	0.620(10)	0.480(12)	0.431(15)	0.721(5)	0.710(7)	0.479(13)	0.740(3)	0.720(6)	0.709(8)	0.762(2)	0.700(9)	0.479(13)	0.606(11)	**0.824(1)**
wine	0.733(5)	0.433(12)	0.477(9)	0.291(14)	0.696(6)	0.886(3)	0.302(13)	0.546(8)	0.634(7)	0.883(4)	0.898(2)	0.077(15)	0.476(10)	0.475(11)	**0.967(1)**
lympho	**0.997(1)**	0.876(9)	0.617(14)	0.816(11)	0.975(3)	0.970(4)	0.864(10)	0.967(7)	0.793(12)	0.969(6)	0.970(4)	0.583(15)	0.696(13)	0.921(8)	0.981(2)
WPBC	0.481(9)	0.466(13)	0.463(15)	0.471(12)	0.485(8)	**0.519(1)**	0.474(11)	0.501(7)	0.504(6)	0.514(3)	0.517(2)	0.478(10)	0.465(14)	0.514(3)	0.507(5)
Stamps	0.876(2)	0.731(6)	0.455(15)	0.571(12)	0.872(3)	0.604(11)	0.540(14)	0.833(5)	0.869(4)	0.635(9)	0.698(7)	0.628(10)	0.558(13)	0.684(8)	**0.912(1)**
WDBC	0.971(7)	0.921(11)	0.501(14)	0.722(12)	0.983(3)	0.982(4)	0.947(8)	0.977(6)	0.922(10)	0.984(2)	0.981(5)	0.068(15)	0.704(13)	0.937(9)	**0.992(1)**
wbc	0.915(8)	0.915(8)	0.622(13)	0.466(14)	0.949(2)	0.945(5)	0.898(10)	**0.951(1)**	0.877(11)	0.949(2)	0.945(5)	0.070(15)	0.659(12)	0.939(7)	0.946(4)
arrhythmia	**0.805(1)**	0.748(11)	0.628(12)	0.589(14)	0.774(3)	0.760(9)	0.772(4)	0.768(6)	0.786(2)	0.757(10)	0.762(7)	0.612(13)	0.533(15)	0.772(4)	0.761(8)
pima	0.594(10)	0.675(4)	0.531(13)	0.492(14)	0.624(6)	0.603(7)	0.591(11)	0.714(2)	0.673(5)	0.602(8)	0.602(8)	0.306(15)	0.537(12)	0.687(3)	**0.735(1)**
vowels	0.605(11)	**0.985(1)**	0.652(10)	0.562(12)	0.793(9)	0.943(4)	0.975(2)	0.975(2)	0.536(13)	0.942(5)	0.942(5)	0.100(15)	0.536(13)	0.942(5)	0.860(8)
cardio	0.938(2)	0.548(10)	0.538(13)	0.537(14)	0.935(3)	0.544(12)	0.567(9)	0.710(5)	0.905(4)	0.578(8)	0.548(10)	0.379(15)	0.600(7)	0.618(6)	**0.946(1)**
musk	0.956(3)	0.054(15)	0.454(13)	0.498(12)	**1.000(1)**	0.637(6)	0.560(9)	0.759(4)	0.540(11)	0.619(7)	0.612(8)	0.674(5)	0.550(10)	0.236(14)	**1.000(1)**
Waveform	0.604(11)	0.654(10)	0.595(12)	0.507(14)	0.672(9)	0.708(6)	0.697(7)	0.734(2)	0.693(8)	0.731(3)	0.718(4)	0.436(15)	0.509(13)	0.710(5)	**0.786(1)**
speech	0.470(12)	**0.751(1)**	0.648(2)	0.442(15)	0.466(13)	0.508(7)	0.532(5)	0.485(10)	0.472(11)	0.507(8)	0.500(9)	0.455(14)	0.517(6)	0.536(4)	0.620(3)
thyroid	**0.976(1)**	0.943(5)	0.529(14)	0.060(15)	0.957(4)	0.704(12)	0.609(13)	0.959(3)	0.712(11)	0.750(9)	0.751(8)	0.869(7)	0.717(10)	0.935(6)	0.970(2)
PageBlocks	0.914(3)	0.750(8)	0.512(14)	0.357(15)	0.915(2)	0.725(10)	0.623(13)	0.855(4)	0.814(5)	0.782(6)	0.740(9)	0.719(11)	0.704(12)	0.761(7)	**0.937(1)**
satimage-2	0.965(4)	0.812(8)	0.493(15)	0.561(11)	0.997(2)	0.534(13)	0.565(10)	0.936(5)	0.995(3)	0.528(14)	0.545(12)	0.899(6)	0.694(9)	0.878(7)	**0.998(1)**
satellite	0.583(7)	0.555(9)	0.466(15)	0.572(8)	0.664(4)	0.546(11)	0.536(13)	0.672(2)	0.650(5)	0.542(12)	0.554(10)	0.669(3)	0.532(14)	0.639(6)	**0.745(1)**
pendigits	0.927(3)	0.673(8)	0.505(12)	0.603(9)	0.931(2)	0.499(13)	0.522(11)	0.744(5)	0.912(4)	0.489(15)	0.497(14)	0.713(6)	0.523(10)	0.688(7)	**0.940(1)**
annthyroid	0.784(4)	**0.830(1)**	0.624(11)	0.318(15)	0.675(10)	0.743(6)	0.727(8)	0.803(3)	0.546(13)	0.804(2)	0.750(5)	0.624(11)	0.536(14)	0.729(7)	0.697(9)
mnist	0.746(6)	0.774(4)	N/A	0.517(14)	0.850(2)	0.679(10)	0.635(11)	0.841(3)	0.730(7)	0.681(9)	0.696(8)	0.559(13)	0.560(12)	0.766(5)	**0.877(1)**
mammography	**0.886(1)**	0.756(6)	0.535(14)	0.405(15)	0.844(2)	0.659(10)	0.604(12)	0.797(4)	0.542(13)	0.669(9)	0.677(8)	0.712(7)	0.606(11)	0.788(5)	0.807(3)
magic_gamma	0.639(11)	0.798(3)	0.575(12)	0.222(15)	0.675(9)	0.697(8)	0.648(10)	0.815(2)	0.703(6)	0.715(5)	0.700(7)	0.531(14)	0.557(13)	**0.819(1)**	0.728(4)
campaign	**0.770(1)**	0.732(4)	0.579(12)	0.529(14)	0.737(3)	0.621(9)	0.595(11)	0.741(2)	0.559(13)	0.605(10)	0.631(7)	0.622(8)	0.529(14)	0.660(6)	0.705(5)
shuttle	**0.993(1)**	0.617(10)	0.498(15)	0.843(6)	0.992(2)	0.523(12)	0.522(13)	0.640(8)	0.984(3)	0.509(14)	0.525(11)	0.733(7)	0.847(5)	0.623(9)	0.949(4)
smtp	0.907(7)	0.945(3)	0.510(14)	0.181(15)	0.923(5)	0.902(8)	0.730(12)	0.951(2)	0.737(11)	0.894(9)	0.919(6)	0.615(13)	0.787(10)	0.939(4)	**0.961(1)**
backdoor	0.886(2)	0.717(10)	O/M	O/M	0.886(2)	0.787(6)	0.739(9)	0.783(7)	0.745(8)	0.796(5)	0.807(4)	O/T	0.702(11)	0.566(12)	**0.912(1)**
celeba	0.701(3)	0.415(11)	O/M	O/M	0.719(2)	0.489(9)	O/M	0.630(4)	0.491(8)	0.507(7)	0.485(10)	O/T	0.528(5)	0.527(6)	**0.756(1)**
fraud	0.947(3)	0.886(7)	O/M	O/M	0.952(2)	0.506(11)	O/M	0.945(4)	0.912(6)	0.508(10)	0.534(9)	O/T	0.679(8)	0.930(5)	**0.955(1)**
cover	0.920(3)	0.750(5)	O/M	O/M	0.952(2)	0.521(11)	O/M	0.779(4)	0.589(7)	0.538(9)	0.527(10)	O/T	0.589(7)	0.713(6)	**0.970(1)**
census	0.488(9)	**0.565(1)**	O/M	O/M	0.542(3)	0.532(5)	O/M	0.551(2)	0.335(10)	0.518(6)	O/T	O/T	0.489(8)	0.517(7)	0.535(4)
http	0.995(4)	0.981(6)	O/M	O/M	**1.000(1)**	0.638(11)	O/M	0.996(3)	0.928(7)	0.806(9)	0.709(10)	O/T	0.888(8)	0.985(5)	**1.000(1)**
average	0.803(3)	0.715(6)	0.540(13)	0.483(15)	0.817(2)	0.676(10)	0.639(11)	0.784(4)	0.713(7)	0.688(9)	0.694(8)	0.532(14)	0.603(12)	0.720(5)	**0.852(1)**

Table 4.3: P@N score on real-world datasets: highest scores in bold; ranking in parentheses (lower is better). Notations: N/A (No Results), O/M (Out-of-Memory, >64GB), O/T (Out-of-Time, >12h).

Dataset	ECOD	FastABOD	SOS	KPCA	OCSVM	LOF	COF	k-NN	DIF	FB	LSCP	MO-GAAL	ALAD	LUNAR	ADOD
Hepatitis	0.308(4)	0.000(14)	0.077(12)	0.083(11)	0.231(7)	0.231(7)	0.000(14)	0.308(4)	0.215(9)	0.254(6)	0.354(2)	0.338(3)	0.154(10)	0.069(13)	**0.385(1)**
wine	0.100(5)	0.000(8)	0.000(8)	0.000(8)	0.000(8)	0.100(5)	0.000(8)	0.000(8)	0.130(4)	0.150(3)	0.200(2)	0.000(8)	0.060(7)	0.000(8)	**0.600(1)**
lympho	**0.833(1)**	0.500(8)	0.000(15)	0.167(14)	0.667(2)	0.667(2)	0.500(8)	0.667(2)	0.433(11)	0.667(2)	0.667(2)	0.267(12)	0.217(13)	0.500(8)	0.667(2)
WPBC	0.128(14)	0.128(14)	0.234(2)	**0.238(1)**	0.170(7)	0.170(7)	0.170(7)	0.170(7)	0.162(12)	0.179(6)	0.181(5)	0.223(3)	0.217(4)	0.160(13)	0.170(7)
Stamps	0.290(2)	0.161(13)	0.032(15)	0.241(3)	0.194(6)	0.194(6)	0.194(6)	0.194(6)	**0.374(1)**	0.190(12)	0.194(6)	0.229(4)	0.097(14)	0.203(5)	0.194(6)
WDBC	0.400(5)	0.200(10)	0.100(14)	0.400(5)	0.500(2)	0.400(5)	0.200(10)	0.400(5)	0.120(13)	0.470(3)	0.470(3)	0.000(15)	0.180(12)	0.320(9)	**0.800(1)**
wbc	0.450(8)	0.300(11)	0.100(14)	0.200(12)	0.500(5)	0.550(2)	0.350(10)	0.500(5)	0.360(9)	**0.570(1)**	0.500(5)	0.000(15)	0.145(13)	0.535(4)	0.550(2)
arrhythmia	**0.485(1)**	0.379(9)	0.288(13)	0.390(8)	0.424(3)	0.379(9)	0.409(5)	0.409(5)	0.468(2)	0.370(12)	0.379(9)	0.280(14)	0.194(15)	0.417(4)	0.394(7)
pima	0.455(7)	0.530(3)	0.347(13)	0.329(14)	0.478(6)	0.440(8)	0.422(11)	0.545(2)	0.485(5)	0.439(9)	0.434(10)	0.195(15)	0.385(12)	0.526(4)	**0.556(1)**
vowels	0.174(11)	**0.804(1)**	0.283(9)	0.100(12)	0.261(10)	0.304(8)	0.609(2)	0.478(4)	0.024(14)	0.322(6)	0.311(7)	0.000(15)	0.035(13)	0.480(3)	0.370(5)
cardio	0.531(3)	0.229(7)	0.109(14)	0.179(10)	0.503(4)	0.160(12)	0.217(8)	0.337(5)	0.535(2)	0.157(13)	0.167(11)	0.082(15)	0.200(9)	0.265(6)	**0.651(1)**
musk	0.495(3)	0.000(15)	0.062(11)	0.302(4)	**1.000(1)**	0.258(6)	0.206(9)	0.237(8)	0.012(14)	0.243(7)	0.268(5)	0.068(10)	0.044(12)	0.037(13)	**1.000(1)**
Waveform	0.040(13)	0.050(10)	0.050(10)	0.138(3)	0.090(9)	0.100(7)	0.100(7)	0.150(2)	0.042(12)	0.128(5)	0.118(6)	0.035(14)	0.032(15)	0.131(4)	**0.320(1)**
speech	0.033(6)	**0.147(1)**	0.098(2)	0.049(4)	0.033(6)	0.033(6)	0.016(13)	0.016(13)	0.028(11)	0.061(3)	0.033(6)	0.011(15)	0.021(12)	0.038(5)	0.033(6)
thyroid	0.548(2)	0.237(6)	0.021(14)	0.000(15)	0.387(3)	0.118(9)	0.065(13)	0.269(5)	0.065(12)	0.073(11)	0.139(8)	0.362(4)	0.117(10)	0.222(7)	**0.570(1)**
PageBlocks	0.431(4)	0.400(5)	0.090(14)	0.071(15)	0.494(2)	0.349(11)	0.304(12)	0.486(3)	0.350(10)	0.374(8)	0.352(9)	0.391(6)	0.303(13)	0.376(7)	**0.612(1)**
satimage-2	0.609(4)	0.174(7)	0.029(14)	0.129(10)	**0.927(1)**	0.087(13)	0.159(8)	0.377(5)	0.912(3)	0.096(12)	0.101(11)	0.000(15)	0.201(6)	0.151(9)	0.913(2)
satellite	0.449(7)	0.372(13)	0.262(15)	0.384(8)	0.538(2)	0.376(11)	0.380(9)	0.492(4)	0.482(5)	0.374(12)	0.377(10)	0.493(3)	0.349(14)	0.463(6)	**0.604(1)**
pendigits	**0.359(1)**	0.077(11)	0.019(15)	0.136(5)	0.340(2)	0.083(8)	0.070(13)	0.103(6)	0.153(3)	0.085(7)	0.083(8)	0.071(12)	0.053(14)	0.078(10)	0.147(4)
annthyroid	0.305(3)	**0.313(1)**	0.219(11)	0.073(15)	0.245(7)	0.242(9)	0.229(10)	**0.313(1)**	0.092(14)	0.254(5)	0.247(6)	0.113(12)	0.107(13)	0.243(8)	0.298(4)
mnist	0.180(11)	0.350(5)	N/A	0.167(12)	0.387(3)	0.294(8)	0.264(9)	0.424(2)	0.232(10)	0.297(7)	0.315(6)	0.154(14)	0.155(13)	0.374(4)	**0.449(1)**
mammography	**0.371(1)**	0.190(7)	0.047(12)	0.043(13)	0.269(3)	0.182(8)	0.130(10)	0.221(4)	0.003(14)	0.136(9)	0.202(5)	0.000(15)	0.110(11)	0.198(6)	0.285(2)
magic_gamma	0.460(11)	0.638(3)	0.414(12)	0.122(15)	0.525(8)	0.523(9)	0.483(10)	0.648(2)	0.550(4)	0.547(6)	0.527(7)	0.409(13)	0.404(14)	**0.659(1)**	0.549(5)
campaign	**0.393(1)**	0.296(5)	0.192(10)	0.016(15)	0.367(2)	0.209(9)	0.182(11)	0.328(3)	0.145(14)	0.156(12)	0.222(8)	0.270(6)	0.147(13)	0.255(7)	0.328(3)
shuttle	0.868(3)	0.193(9)	0.076(15)	0.336(6)	**0.956(1)**	0.125(11)	0.112(13)	0.214(8)	0.952(2)	0.097(14)	0.125(11)	0.280(7)	0.503(4)	0.192(9)	0.408(5)
smtp	**0.714(1)**	0.095(7)	0.000(10)	0.000(10)	0.571(3)	0.000(10)	0.000(10)	0.286(4)	0.133(6)	0.000(10)	0.143(5)	0.000(10)	0.038(8)	0.029(9)	0.667(2)
backdoor	0.137(11)	0.318(8)	O/M	O/M	0.548(2)	0.475(3)	0.456(5)	0.436(6)	0.030(12)	0.374(7)	0.475(3)	O/T	0.227(9)	0.185(10)	**0.549(1)**
celeba	0.134(2)	0.000(11)	O/M	O/M	**0.138(1)**	0.021(9)	O/M	0.051(4)	0.028(6)	0.028(6)	0.021(9)	O/T	0.041(5)	0.028(6)	0.069(3)
fraud	0.294(2)	0.046(7)	O/M	O/M	0.076(6)	0.000(9)	O/M	0.093(5)	**0.316(1)**	0.000(9)	0.000(9)	O/T	0.035(8)	0.197(4)	0.271(3)
cover	0.164(2)	0.053(6)	O/M	O/M	0.076(4)	0.034(8)	O/M	0.085(3)	0.013(10)	0.030(9)	0.037(7)	O/T	0.005(11)	0.060(5)	**0.277(1)**
census	0.046(9)	**0.097(1)**	O/M	O/M	0.064(6)	0.087(3)	O/M	0.095(2)	0.025(10)	0.053(8)	O/T	O/T	0.081(4)	0.070(5)	0.054(7)
http	0.278(4)	0.250(5)	O/M	O/M	0.361(2)	0.139(7)	O/M	0.292(3)	0.006(11)	0.019(10)	0.111(8)	O/T	0.156(6)	0.060(9)	**0.875(1)**
average	0.358(3)	0.235(7)	0.126(15)	0.165(12)	0.385(2)	0.229(10)	0.231(9)	0.301(4)	0.246(6)	0.225(11)	0.250(5)	0.164(13)	0.157(14)	0.235(7)	**0.457(1)**

Table 4.4: AP score on real-world datasets: highest scores in bold; ranking in parentheses (lower is better). Notations: N/A (No Results), O/M (Out-of-Memory, >64GB), O/T (Out-of-Time, >12h).

Dataset	ECOD	FastABOD	SOS	KPCA	OCSVM	LOF	COF	k-NN	DIF	FB	LSCP	MO-GAAL	ALAD	LUNAR	ADOD
Hepatitis	0.292(4)	0.198(12)	0.175(13)	0.156(14)	0.278(7)	0.253(9)	0.153(15)	0.281(6)	0.287(5)	0.263(8)	0.315(3)	0.362(2)	0.204(10)	0.203(11)	**0.417(1)**
wine	0.191(5)	0.070(12)	0.076(11)	0.061(13)	0.142(7)	0.270(3)	0.058(14)	0.089(9)	0.161(6)	0.266(4)	0.293(2)	0.046(15)	0.116(8)	0.082(10)	**0.566(1)**
lympho	**0.915(1)**	0.341(11)	0.069(15)	0.274(12)	0.782(3)	0.716(6)	0.628(8)	0.760(4)	0.469(10)	0.725(5)	0.716(6)	0.270(13)	0.234(14)	0.624(9)	0.803(2)
WPBC	0.218(14)	0.226(12)	**0.244(1)**	0.239(4)	0.223(13)	0.232(7)	0.217(15)	0.227(11)	0.243(2)	0.230(10)	0.233(6)	0.236(5)	0.241(3)	0.232(7)	0.232(7)
Stamps	0.314(3)	0.184(9)	0.111(15)	0.164(13)	0.294(4)	0.175(12)	0.183(10)	0.266(5)	0.338(2)	0.178(11)	0.195(8)	0.236(6)	0.129(14)	0.198(7)	**0.378(1)**
WDBC	0.505(5)	0.208(11)	0.068(14)	0.370(9)	0.494(6)	0.535(4)	0.231(10)	0.438(7)	0.195(12)	0.550(2)	0.544(3)	0.016(15)	0.168(13)	0.375(8)	**0.637(1)**
wbc	0.474(8)	0.312(10)	0.113(14)	0.150(12)	0.540(5)	0.574(4)	0.309(11)	0.527(6)	0.372(9)	0.579(3)	0.587(2)	0.029(15)	0.143(13)	0.511(7)	**0.610(1)**
arrhythmia	**0.466(1)**	0.330(10)	0.239(14)	0.293(12)	0.391(4)	0.330(10)	0.459(2)	0.382(6)	0.424(3)	0.332(9)	0.338(8)	0.258(13)	0.196(15)	0.391(4)	0.356(7)
pima	0.464(6)	0.503(4)	0.381(13)	0.366(14)	0.464(6)	0.418(9)	0.436(8)	**0.530(1)**	0.475(5)	0.415(10)	0.414(11)	0.262(15)	0.410(12)	0.514(3)	0.522(2)
vowels	0.075(11)	**0.837(1)**	0.229(9)	0.072(12)	0.191(10)	0.321(8)	0.618(2)	0.524(3)	0.038(14)	0.338(7)	0.361(5)	0.017(15)	0.042(13)	0.470(4)	0.341(6)
cardio	0.570(2)	0.161(8)	0.112(14)	0.151(13)	0.534(3)	0.154(11)	0.160(9)	0.321(5)	0.501(4)	0.159(10)	0.153(12)	0.091(15)	0.186(7)	0.221(6)	**0.585(1)**
musk	0.492(3)	0.017(15)	0.031(14)	0.261(4)	**1.000(1)**	0.205(7)	0.162(9)	0.240(5)	0.036(12)	0.203(8)	0.215(6)	0.083(10)	0.055(11)	0.032(13)	**1.000(1)**
Waveform	0.041(12)	0.054(9)	0.040(12)	0.065(8)	0.053(10)	0.076(6)	0.073(7)	0.106(2)	0.051(11)	0.088(4)	0.085(5)	0.030(15)	0.032(14)	0.104(3)	**0.315(1)**
speech	0.020(9)	**0.099(1)**	0.048(2)	0.017(14)	0.018(12)	0.025(4)	0.023(7)	0.019(11)	0.018(12)	0.024(5)	0.023(7)	0.016(15)	0.020(9)	0.024(5)	0.029(3)
thyroid	0.467(2)	0.209(7)	0.030(14)	0.014(15)	0.320(3)	0.092(10)	0.040(13)	0.284(4)	0.053(12)	0.070(11)	0.109(8)	0.280(5)	0.107(9)	0.220(6)	**0.486(1)**
PageBlocks	0.520(3)	0.390(5)	0.103(14)	0.077(15)	0.533(2)	0.300(11)	0.257(13)	0.484(4)	0.313(9)	0.352(7)	0.322(8)	0.311(10)	0.282(12)	0.375(6)	**0.629(1)**
satimage-2	0.668(4)	0.117(7)	0.015(15)	0.052(11)	**0.964(1)**	0.033(14)	0.094(10)	0.353(5)	0.959(2)	0.035(12)	0.034(13)	0.113(8)	0.178(6)	0.111(9)	0.958(3)
satellite	0.526(6)	0.386(10)	0.286(15)	0.425(8)	**0.655(1)**	0.376(13)	0.381(12)	0.527(5)	0.627(2)	0.372(14)	0.383(11)	0.570(3)	0.390(9)	0.480(7)	0.536(4)
pendigits	**0.266(1)**	0.051(10)	0.028(15)	0.056(7)	0.228(2)	0.043(12)	0.040(14)	0.076(5)	0.164(3)	0.050(11)	0.041(13)	0.056(7)	0.065(6)	0.053(9)	0.160(4)
annthyroid	**0.269(1)**	0.255(2)	0.150(11)	0.058(15)	0.186(8)	0.189(7)	0.182(10)	0.228(4)	0.097(14)	0.211(5)	0.195(6)	0.133(12)	0.107(13)	0.185(9)	0.241(3)
mnist	0.178(11)	0.302(5)	N/A	0.140(12)	0.385(3)	0.250(8)	0.204(9)	0.391(2)	0.199(10)	0.253(7)	0.266(6)	0.130(14)	0.133(13)	0.316(4)	**0.440(1)**
mammography	**0.363(1)**	0.138(6)	0.044(13)	0.033(14)	0.185(3)	0.101(9)	0.069(12)	0.161(4)	0.032(15)	0.077(11)	0.107(7)	0.090(10)	0.102(8)	0.146(5)	0.190(2)
magic_gamma	0.530(10)	0.718(3)	0.431(13)	0.229(15)	0.624(5)	0.531(9)	0.498(11)	0.741(2)	0.589(6)	0.550(7)	0.535(8)	0.383(14)	0.436(12)	**0.745(1)**	0.660(4)
campaign	**0.355(1)**	0.243(5)	0.163(10)	0.112(15)	0.284(2)	0.166(9)	0.156(11)	0.276(3)	0.133(13)	0.150(12)	0.174(8)	0.206(7)	0.133(13)	0.210(6)	0.246(4)
shuttle	0.904(3)	0.142(10)	0.072(15)	0.314(6)	0.906(2)	0.110(12)	0.091(13)	0.166(9)	**0.966(1)**	0.084(14)	0.112(11)	0.247(7)	0.523(4)	0.169(8)	0.431(5)
smtp	**0.572(1)**	0.070(7)	0.001(13)	0.000(13)	0.428(2)	0.013(10)	0.002(11)	0.266(4)	0.137(5)	0.002(11)	0.078(6)	0.000(13)	0.023(9)	0.026(8)	0.384(3)
backdoor	0.156(10)	0.195(8)	O/M	O/M	0.566(2)	0.413(4)	0.381(5)	0.373(6)	0.054(12)	0.302(7)	0.423(3)	O/T	0.160(9)	0.078(11)	**0.623(1)**
celeba	0.082(2)	0.020(11)	O/M	O/M	**0.086(1)**	0.025(9)	O/M	0.040(4)	0.026(7)	0.026(7)	0.025(9)	O/T	0.032(5)	0.028(6)	0.070(3)
fraud	0.198(3)	0.032(7)	O/M	O/M	0.094(5)	0.002(9)	O/M	0.060(6)	**0.239(1)**	0.002(9)	0.002(9)	O/T	0.014(8)	0.116(4)	0.213(2)
cover	0.107(2)	0.028(6)	O/M	O/M	0.096(3)	0.012(10)	O/M	0.036(4)	0.012(10)	0.013(8)	0.013(8)	O/T	0.016(7)	0.030(5)	**0.239(1)**
census	0.074(9)	**0.094(1)**	O/M	O/M	0.085(4)	0.088(3)	O/M	0.091(2)	0.057(10)	0.082(6)	O/T	O/T	0.081(8)	0.082(6)	0.084(5)
http	0.115(6)	0.126(5)	O/M	O/M	0.472(2)	0.071(7)	O/M	0.215(3)	0.005(10)	0.004(11)	0.067(8)	O/T	0.137(4)	0.042(9)	**0.757(1)**
average	0.356(3)	0.220(10)	0.130(15)	0.160(13)	0.391(2)	0.222(9)	0.226(8)	0.296(4)	0.258(5)	0.218(11)	0.237(6)	0.172(12)	0.159(14)	0.231(7)	**0.442(1)**

Table 4.5: Run time (in seconds) on real-world datasets (Part 1): fastest in bold; ranking in parentheses (lower is better). Notations: N/A (No Results), O/M (Out-of-Memory, >64GB), O/T (Out-of-Time, >12h).

Dataset	ECOD	FastABOD	SOS	KPCA	OCSVM	LOF	COF	k-NN
Hepatitis	0.001(2)	0.016(6)	0.016(6)	0.016(6)	0.002(3)	0.002(3)	0.007(5)	0.016(6)
wine	0.001(2)	0.011(7)	0.021(8)	0.026(9)	0.002(4)	0.002(4)	0.009(6)	0.001(2)
lympho	0.001(2)	0.018(7)	0.024(8)	0.030(9)	0.002(3)	0.008(5)	0.012(6)	0.006(4)
WPBC	0.002(2)	0.022(7)	0.034(8)	0.041(9)	0.003(3)	0.009(5)	0.017(6)	0.006(4)
Stamps	0.001(2)	0.031(8)	0.064(9)	0.078(10)	0.005(5)	0.003(4)	0.029(7)	0.002(3)
WDBC	0.002(2)	0.040(7)	0.071(8)	0.091(9)	0.005(3)	0.010(4)	0.034(6)	0.010(4)
wbc	0.001(2)	0.042(7)	0.073(8)	0.085(9)	0.006(3)	0.011(5)	0.034(6)	0.010(4)
arrhythmia	**0.011(1)**	0.058(6)	0.090(7)	0.120(9)	0.018(5)	0.013(2)	0.139(10)	0.015(3)
pima	**0.001(1)**	0.065(6)	0.198(10)	0.211(11)	0.022(5)	0.012(3)	0.091(8)	0.005(2)
vowels	**0.003(1)**	0.130(7)	0.461(10)	0.485(11)	0.078(5)	0.028(3)	0.236(9)	0.027(2)
cardio	**0.005(1)**	0.175(7)	0.658(10)	0.685(11)	0.133(6)	0.045(3)	0.391(8)	0.043(2)
musk	**0.081(1)**	0.343(6)	1.465(10)	2.082(11)	0.578(7)	0.109(2)	10.450(13)	0.118(3)
Waveform	**0.010(1)**	0.307(6)	1.977(11)	2.947(12)	0.465(7)	0.023(3)	1.610(10)	0.015(2)
speech	0.277(4)	0.517(5)	2.173(10)	3.638(11)	1.605(9)	**0.080(1)**	35.354(14)	0.110(2)
thyroid	**0.003(1)**	0.245(5)	2.211(11)	2.762(12)	0.482(8)	0.087(3)	1.317(10)	0.051(2)
PageBlocks	**0.007(1)**	0.383(5)	4.753(11)	7.146(13)	1.116(8)	0.153(3)	3.086(10)	0.112(2)
satimage-2	**0.025(1)**	0.413(5)	5.177(10)	11.196(13)	1.388(8)	0.048(3)	10.988(12)	0.039(2)
satellite	**0.034(1)**	0.503(5)	6.424(10)	14.751(13)	1.692(8)	0.057(3)	12.868(12)	0.046(2)
pendigits	**0.015(1)**	0.486(5)	6.882(11)	16.735(13)	1.770(9)	0.053(3)	6.050(10)	0.039(2)
annthyroid	**0.005(1)**	0.460(5)	7.535(11)	15.813(14)	1.810(9)	0.131(3)	5.442(10)	0.094(2)
mnist	**0.085(1)**	0.729(5)	10.674(10)	200.506(15)	3.051(9)	0.101(2)	41.867(13)	0.118(3)
mammography	**0.007(1)**	0.497(5)	9.909(12)	20.043(14)	2.340(9)	0.129(3)	6.488(10)	0.080(2)
magic_gamma	**0.037(1)**	1.645(5)	49.115(12)	276.881(15)	15.027(9)	0.903(3)	56.354(14)	0.561(2)
campaign	**0.307(1)**	3.854(4)	212.156(12)	2605.953(15)	107.709(11)	1.147(2)	857.158(13)	1.227(3)
shuttle	**0.059(1)**	5.527(3)	746.870(14)	3930.835(15)	104.717(11)	5.552(4)	478.822(13)	2.700(2)
smtp	**0.040(1)**	4.263(4)	3851.268(14)	12978.204(15)	220.319(12)	0.531(3)	1103.847(13)	0.306(2)
backdoor	**2.115(1)**	17.893(4)	O/M	O/M	1215.598(11)	9.674(2)	11305.249(13)	10.953(3)
celeba	**0.575(1)**	14.060(4)	O/M	O/M	850.937(11)	6.254(2)	O/M	6.642(3)
fraud	**2.086(1)**	51.627(4)	O/M	O/M	4901.288(10)	33.245(2)	O/M	34.182(3)
cover	**0.501(1)**	25.353(4)	O/M	O/M	3674.421(11)	17.667(3)	O/M	8.497(2)
census	**15.256(1)**	197.360(5)	O/M	O/M	16197.552(10)	156.431(3)	O/M	167.366(4)
http	**0.120(1)**	17.166(4)	O/M	O/M	2357.628(11)	2.114(3)	O/M	1.525(2)
average	**0.677(1)**	10.758(4)	189.242(11)	772.745(13)	926.930(14)	7.332(2)	516.220(12)	7.341(3)

Table 4.6: Run time (in seconds) on real-world datasets (Part 2): fastest in bold; ranking in parentheses (lower is better). Notations: N/A (No Results), O/M (Out-of-Memory, >64GB), O/T (Out-of-Time, >12h).

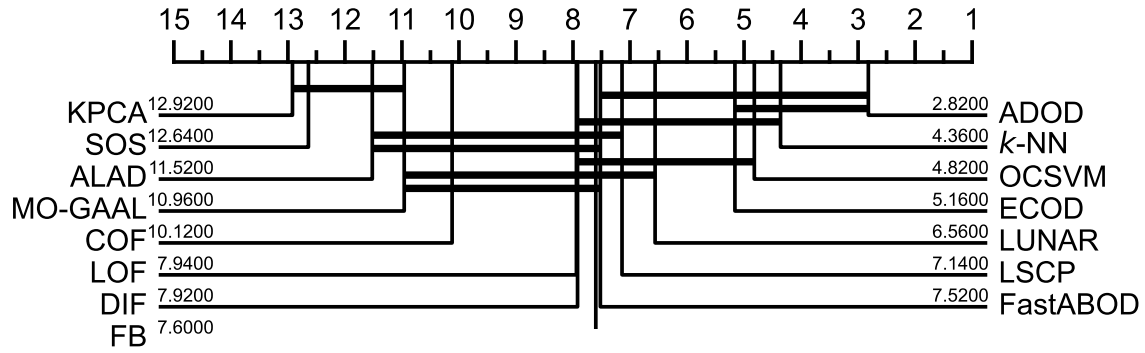
Dataset	DIF	FB	LSCP	MO-GAAL	ALAD	LUNAR	ADOD*	ADOD
Hepatitis	0.450(13)	0.031(10)	0.268(12)	40.307(16)	31.830(15)	0.770(14)	**0.000(1)**	0.100(11)
wine	0.496(13)	0.026(9)	0.284(12)	41.805(16)	30.880(15)	0.809(14)	**0.000(1)**	0.110(11)
lympho	0.517(13)	0.032(10)	0.415(12)	41.446(16)	32.343(15)	0.879(14)	**0.000(1)**	0.112(11)
WPBC	0.555(13)	0.064(10)	0.533(12)	43.161(16)	33.192(15)	0.931(14)	**0.000(1)**	0.114(11)
Stamps	0.663(12)	0.026(6)	0.704(13)	45.519(16)	32.971(15)	0.898(14)	**0.000(1)**	0.105(11)
WDBC	0.847(12)	0.126(11)	0.942(13)	46.946(16)	32.846(15)	1.020(14)	**0.000(1)**	0.107(10)
wbc	0.829(12)	0.116(11)	0.962(14)	47.518(16)	32.748(15)	0.932(13)	**0.000(1)**	0.107(10)
arrhythmia	1.038(13)	0.146(11)	1.542(14)	50.633(16)	31.917(15)	1.014(12)	0.015(4)	0.107(8)
pima	1.542(13)	0.073(7)	1.634(14)	49.385(16)	30.929(15)	0.944(12)	0.016(4)	0.108(9)
vowels	2.532(13)	0.169(8)	3.292(14)	98.108(16)	32.575(15)	1.026(12)	0.031(4)	0.115(6)
cardio	2.921(13)	0.393(9)	4.888(14)	150.362(16)	31.347(15)	1.214(12)	0.048(4)	0.118(5)
musk	4.910(12)	1.010(8)	21.969(14)	710.906(16)	30.347(15)	1.276(9)	0.331(5)	0.169(4)
Waveform	4.991(13)	1.186(9)	10.090(14)	674.723(16)	32.970(15)	1.178(8)	0.220(5)	0.146(4)
speech	6.301(12)	0.692(6)	56.534(15)	1135.564(16)	34.936(13)	1.573(8)	0.980(7)	0.146(3)
thyroid	4.981(13)	0.385(7)	8.100(14)	872.353(16)	32.479(15)	1.216(9)	0.332(6)	0.178(4)
PageBlocks	6.971(12)	0.797(7)	13.392(14)	1410.542(16)	32.681(15)	1.583(9)	0.550(6)	0.258(4)
satimage-2	8.083(11)	0.430(6)	18.466(14)	1820.492(16)	34.501(15)	1.681(9)	0.566(7)	0.222(4)
satellite	8.869(11)	0.510(6)	20.052(14)	2196.644(16)	34.062(15)	1.750(9)	0.707(7)	0.253(4)
pendigits	9.803(12)	1.589(7)	17.857(14)	2238.766(16)	33.137(15)	1.708(8)	0.716(6)	0.292(4)
annthyroid	9.128(12)	0.667(6)	15.543(13)	2495.206(16)	33.654(15)	1.759(8)	0.816(7)	0.309(4)
mnist	11.322(11)	0.983(6)	59.895(14)	3089.309(16)	33.500(12)	2.059(8)	1.283(7)	0.335(4)
mammography	9.898(11)	0.568(6)	15.840(13)	2134.093(16)	32.123(15)	1.852(8)	1.162(7)	0.401(4)
magic_gamma	25.340(10)	5.808(7)	55.932(13)	2823.087(16)	34.496(11)	6.585(8)	5.592(6)	1.349(4)
campaign	59.581(10)	9.401(6)	1087.362(14)	9057.621(16)	40.160(9)	13.467(7)	36.794(8)	6.495(5)
shuttle	60.473(9)	24.739(7)	190.964(12)	11990.691(16)	40.160(8)	15.812(6)	63.752(10)	8.105(5)
smtp	83.347(9)	13.880(5)	105.221(10)	25022.970(16)	40.135(8)	15.220(7)	172.821(11)	14.418(6)
backdoor	122.164(9)	78.607(7)	6938.666(12)	O/T	67.415(6)	79.842(8)	277.521(10)	23.109(5)
celeba	162.863(9)	57.624(8)	3894.577(12)	O/T	53.090(6)	55.181(7)	629.568(10)	49.700(5)
fraud	374.678(7)	1143.984(9)	25887.675(11)	O/T	84.175(5)	234.578(6)	O/M	468.122(8)
cover	397.624(9)	102.576(6)	962.728(10)	O/T	76.589(5)	167.252(7)	O/M	294.397(8)
census	360.443(7)	1130.943(9)	O/T	O/T	296.550(6)	927.143(8)	O/M	103.457(2)
http	261.809(8)	391.694(9)	548.678(10)	O/T	67.167(6)	40.341(5)	O/M	110.156(7)
average	62.687(9)	92.790(10)	1288.549(15)	2628.006(16)	47.435(7)	49.484(8)	42.636(6)	33.851(5)

4.4.4 Results on Real-World Datasets

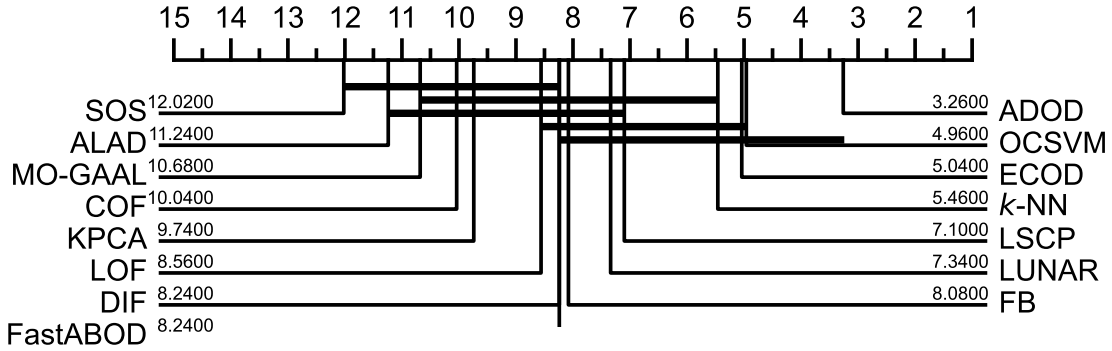
Table 4.2 shows the ROC score on real-world datasets. ADOD ranked first on 19 out of 32 datasets, achieving an overall average score of 0.852 and outperforming all baselines. ADOD and OCSVM both achieved a perfect score of 1.0 on musk dataset, while their scores on http dataset were rounded to 1.0. Despite leading only two datasets, OCSVM ranked second overall with an average score of 0.817. ECOD led in six datasets, ranking third overall with a score of 0.803. In contrast, SOS and MO-GAAL underperformed, with average scores of 0.540 and 0.532, respectively. KPCA performed the worst, with an average score of 0.483, ranking 15th. It is worth noting that certain algorithms failed to complete the computation because of out-of-memory (O/M) or out-of-time (O/T) issues on specific large or high-dimensional datasets, with a single runtime limit of 12 hours. For instance, SOS, KPCA, and COF encountered out-of-memory issues on particular datasets, whereas LSCP and MO-GAAL were plagued by timeouts.

Table 4.3 and Table 4.4 present the P@N and AP scores on 32 real-world datasets. ADOD consistently ranked first on 14 out of 32 datasets across both metrics, achieving an overall average score of 0.457 for P@N and 0.442 for AP, significantly outperforming all baselines. Notably, ADOD achieved a perfect score of 1.0 on musk dataset for both metrics. OCSVM followed in terms of overall performance, ranking first on four datasets, with average scores of 0.385 for P@N and 0.391 for AP. Compared to OCSVM, ADOD exhibits improvements of approximately 18.70% in P@N and 13.04% in AP. In contrast, SOS performs the worst, with average scores of 0.126 for P@N and 0.130 for AP, ranking 15th. Notably, several algorithms achieve a score of 0 on certain datasets, such as wine, smtp, and fraud, in terms of P@N score, suggesting that some algorithms (including FastABOD, SOS, KPCA, COF, and MO-GAAL) struggle to handle the challenging characteristics of these datasets. Moreover, KPCA and MO-GAAL yield a score of 0 on SMTP dataset for AP.

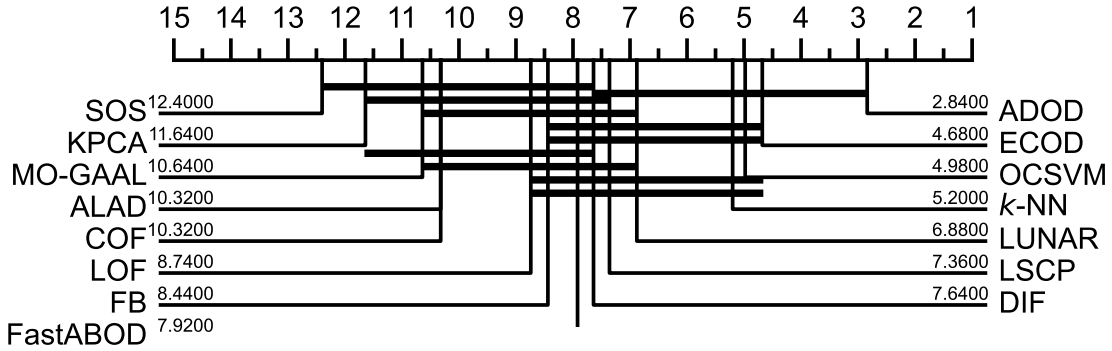
Fig. 4.7 shows the CD between 15 outlier detection algorithms in terms of ROC, P@N, and AP scores on 25 datasets after removing those containing N/A, O/T, and O/M.



(a) CD diagram of ROC score



(b) CD diagram of P@N score



(c) CD diagram of AP score

Figure 4.7: CD diagram illustrating pairwise statistical difference comparison: ADOD outperforms baselines on ROC, P@N, and AP scores, with statistically significant results on P@N score.

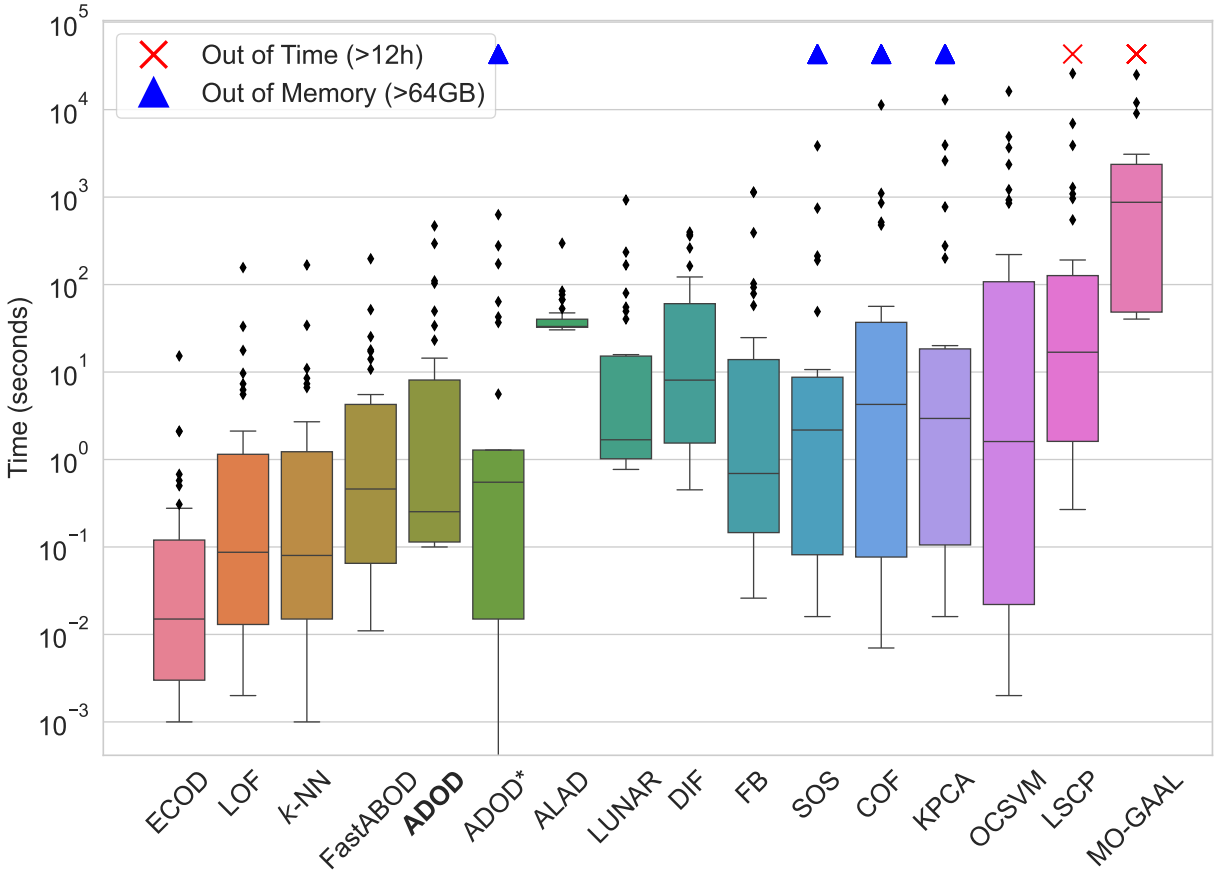


Figure 4.8: Run time comparison of 16 algorithms on 32 real-world datasets. Boxplot illustrates the median, quartiles, and outliers of log-scaled runtime for each algorithm, sorted by their average runtime.

ADOD ranked first on 13 out of 25 datasets based on ROC score and on 10 out of 25 datasets according to P@N and AP scores, achieving the highest average rank across these metrics. ADOD demonstrated statistically significant differences in P@N scores compared to other algorithms, indicated by the lack of connections between ADOD and the other algorithms in the CD diagram. k-NN, OCSVM, and ECOD also performed well, usually ranking high. In contrast, algorithms such as KPCA, COF, MO-GAAL, ALAD, and SOS ranked low across these metrics.

4.4.5 Runtime Analysis

Fig.4.8 provides a boxplot illustration based on the runtime in Table4.5 and Table 4.6. As shown in this figure, ADOD refers to the optimized algorithm with NNS, while ADOD* refers to the original algorithm without NNS optimization. ADOD* ranked 6th with an average time of 42s but suffered from out-of-memory errors on datasets exceeding 200k samples. In contrast, the optimized ADOD ranked 5th with an average time of 33s, maintaining a relatively low and stable runtime across most datasets and demonstrating its efficiency and reliability in handling large datasets. For example, ADOD (49s) was approximately 12.8 times faster than ADOD* (629s) on celeba dataset (114k). SOS, COF, and KPCA encountered out-of-memory errors when processing backdoor (87k) and celeba (114k) datasets. ECOD, with an average runtime of less than 1s, exhibited the shortest runtime among all algorithms. Meanwhile, LSCP (average time: 1,288s) and MO-GAAL (average time: 2,628s) ran over 12 hours on large or high-dimensional datasets, highlighting their computational inefficiencies in these cases.

Notably, the performance difference between ADOD and ADOD* was negligible. Comparing their performance on each dataset (ADOD - ADOD*), we obtained an average difference of 0.0035 in ROC score with a standard deviation of 0.0171, an average difference of -0.0054 in P@N score with a standard deviation of 0.0394, and an average difference of 0.0038 in AP score with a standard deviation of 0.0183. Thus, NNS optimization improves efficiency and reduces memory overhead while remaining consistent in terms of performance compared with the original algorithm ADOD*.

4.4.6 Visualization on Real-World Datasets

The first column of Fig. 4.9 shows the visualization of musk dataset. ADOD achieved ROC, P@N, and AP scores of 1.0, indicating that it accurately identified all outliers. The visualization results show that the outliers detected by ADOD perfectly matched the distribution of the true labels and were located in separate clusters far from the main clusters. The second column of Fig. 4.9 shows the visualization of magic_gamma dataset.

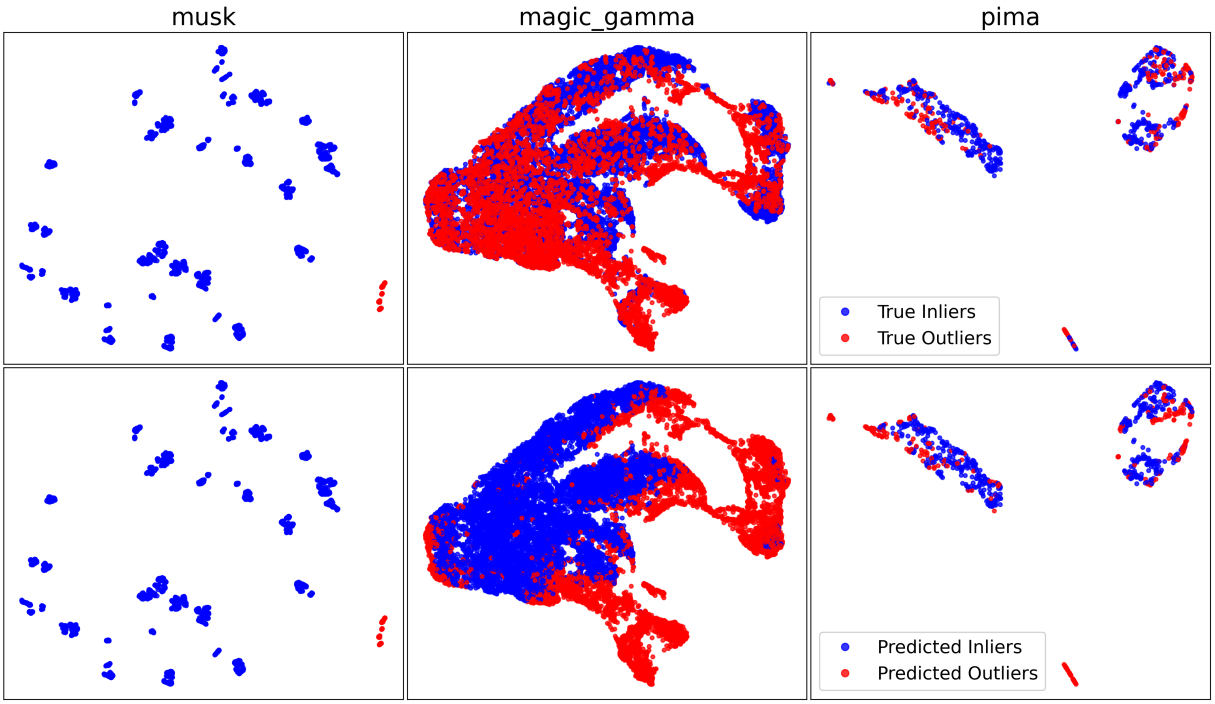


Figure 4.9: Visualization of musk, magic_gamma, and pima datasets using UMAP [MHM18], comparing ground truth labels (first row) and ADOD predictions (second row) with inliers in blue and outliers in red.

ADOD achieved ROC, P@N, and AP scores of 0.728, 0.549, and 0.66, respectively. In the ground truth, outliers were scattered throughout the data distribution, whereas the outliers detected by ADOD were primarily located at the edges of the data distribution. Compared with the outliers in the ground truth, predicted outliers appeared to better represent the characteristics of outliers. The third column of Fig. 4.9 shows the visualization of pima dataset. ADOD achieved ROC, P@N, and AP scores of 0.735, 0.556, and 0.522, respectively. Compared with the outliers in the ground truth, the outliers predicted by ADOD better reflected the separation from the inliers, as indicated by two completely red clusters of outliers.

This discrepancy between evaluation metrics and human intuition is an interesting discovery because we often rely on evaluation metrics to assess the performance of outlier detection algorithms. The visualization revealed certain cases where the results were more valuable, even if the metric scores were not high. Through visualization, we can in-

tuitively observe that the outliers detected by ADOD on specific datasets are more consistent with human intuition and are typically located at the edges of the data distributions or form separate small clusters.

4.5 Conclusion

This chapter proposes a novel unsupervised Adaptive Density Outlier Detection (ADOD) algorithm for handling data with varying densities. The primary innovations of ADOD included adaptive neighborhood boundaries and density consistency scoring. In addition, we optimized ADOD using nearest neighbor search to improve its efficiency and reduce memory usage. The experimental results demonstrated that ADOD significantly outperformed 14 other outlier detection algorithms from different categories in key performance metrics, including ROC, P@N, AP scores, and execution time. In the parameter sensitivity analysis, we fixed the values of the primary parameters. We demonstrated their robust performance, which enabled ADOD to be regarded as a parameter-free method in practical applications, eliminating the need for users to tune the parameters manually. ADOD is adaptable and scalable while maintaining high accuracy, providing an effective outlier detection solution, and expanding its application potential in real-time data processing.

Chapter 5

Dynamic Graph Construction for Nonlinear Dimensionality Reduction

Most well-known graph-based dimensionality reduction algorithms, such as t-SNE and UMAP, use a two-step approach: first, to construct a graph out of the high-dimensional data and then to embed the graph into the low-dimensional space. The main challenges of these algorithms include how to construct a good graph and how to maintain the similarity structure of the high-dimensional data in the low-dimensional space. This chapter proposes DynoGraph, a novel algorithm called **Dy**namic **Graph** construction for **No**nlinear dimensionality reduction, to address these two challenges. First, we develop an adaptive neighborhood graph construction method that accurately captures the intrinsic geometry of the high-dimensional data. Second, for the first time, we introduce a dynamic graph modification process during dimensionality reduction, ensuring that the data structure in the low-dimensional space faithfully reflects the high-dimensional data. For vertex pairs connected by edges in high-dimensional space exhibit far apart in low-dimensional space, additional edges are inserted to strengthen the connection between them. Conversely, for vertex pairs not connected in high-dimensional space exhibit close together in the low-dimensional space, edges are deleted to reduce the connection between them. These adjustments help to update their positions in subsequent embed-

dings, aligning them toward the high-dimensional data. Extensive experiments have demonstrated the superiority of DynoGraph against various comparative algorithms in tasks such as visualization, classification, and clustering.

Parts of the material presented in this chapter have been published in [QPQ+24]:

> *"Li Qian, Claudia Plant, Yalan Qin, Jing Qian, Christian Böhm. DynoGraph: Dynamic Graph Construction for Nonlinear Dimensionality Reduction. 2024 IEEE International Conference on Data Mining (ICDM), pp. 827-832, 2024."*

where Li Qian was responsible for the development of the main concepts, the implementation of the algorithm, most of the experimental evaluation, and writing the majority of the paper; Claudia Plant proposed the initial idea of applying graph data compression to dimensionality reduction and dynamically modifying the graph structure; Yalan Qin was responsible for experiments on selected comparison algorithms and wrote the corresponding parts of the paper; Jing Qian proposed the research idea of adaptive threshold setting when dynamically modifying the graph structure during dimensionality reduction. Christian Böhm and Claudia Plant supervised and guided the research. All coauthors participated in the discussion of concept development and the revision of the manuscript.

5.1 Introduction

Nonlinear dimensionality reduction is an unsupervised learning task that targets transforming high-dimensional data to a low-dimensional space while preserving the intrinsic structure of the original data [LV07]. It has a wide range of applications, covering many fields like image processing, computer vision, bioinformatics, genomics, and financial analysis [EMK+21]. A practical nonlinear dimensionality reduction method not only helps visualize high-dimensional data, making it more accessible, but also promotes interpretability in downstream tasks such as clustering, classification, and outlier detection [AHT20].

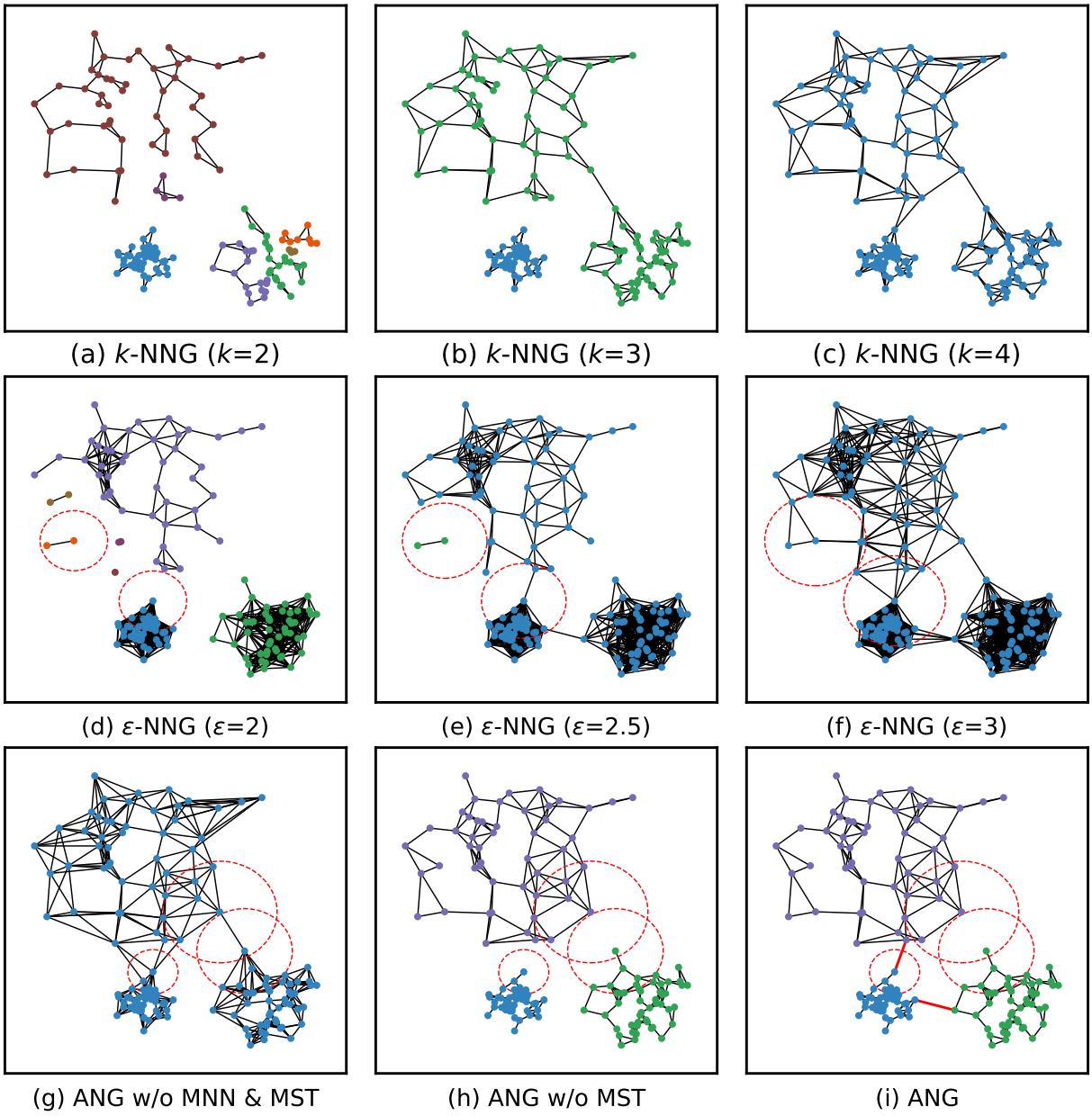


Figure 5.1: Graph construction techniques on the toy dataset. Showcases three classes, each with 50 nodes, derived from a Gaussian distribution with a variance ratio of 1:2:4. Different colors indicate different components; the black line represents the normal edge, and the red line represents the edge where ANG connects the different components through MST. The red dashed circles represent the ϵ-range for nodes in ϵ-NNG and the ϵ_i-range in ANG. For a fair comparison with k-NNG and ϵ-NNG, the minimum five neighbors condition in ANG is omitted.

In nonlinear dimensionality reduction, graph-based methods, such as Isometric feature mapping (Isomap) [TdSL00], t-SNE [vdMH08], and Uniform Manifold approximation and Projection (UMAP) [MHM18], involve two main steps: graph construction and node embedding. First, high-dimensional data points are treated as nodes in the graph, and edges are formed based on the proximity between these points. Second, nodes are embedded into a low-dimensional space based on this graph. The effectiveness of these methods depends heavily on graph quality. In dimensionality reduction, the commonly used graph construction methods are k-NNG and ϵ-NNG [QZCS18]. The k-NNG is constructed by connecting each node to its k nearest neighbors. However, a fixed k may not reflect the actual proximity. For example, nodes are connected even if they are far away because they belong to the nearest k neighbors of a node. The ϵ-NNG is constructed by connecting all node pairs whose distance is less than a fixed threshold ϵ. However, a fixed ϵ can result in overly sparse or dense regions in the graph when handling data with varying densities. Fig. 5.1(a)-(g) illustrate that, even with the extensive tuning of the parameters k or ϵ, it is still challenging to construct a graph that accurately represents the three distinct classes in a toy dataset with varying densities.

Maintaining the similarity structure of the high-dimensional data in the low-dimensional space is another challenge, leading to distortion. Distortion refers to the alteration or loss of relationships and properties of the original data during the transformation from high-dimensional to low-dimensional space [CVLD19]. This distortion occurs when close node pairs in high-dimensional space become distant in low-dimensional space or vice versa. Distortion is inevitable in graph-based methods because they rely heavily on the graph structure to update the embedding in the low-dimensional space. However, this graph structure does not accurately reflect the actual distances between high-dimensional data points [CPA$^+$20]. Specifically, densely connected regions in the graph may appear very close in the embedding, while sparsely connected regions may end up further apart.

We propose a novel algorithm, Dynamic Graph construction for Nonlinear dimensionality reduction (DynoGraph), to address these challenges. The primary contributions

include:

1. We propose an adaptive neighborhood graph construction method that captures the intrinsic geometric structure of the data, avoids global fixed parameters, and adapts to data with varying densities and sizes.

2. We introduce for the first time a dynamic graph modification process during dimensionality reduction to mitigate distortion and maintain consistency with high-dimensional data.

3. We design an objective function called the Graph Structure Representation Measure, which applies attractive forces to vertex pairs with edges and repulsive forces to vertex pairs without edges.

5.2 Related Work

We categorize dimensionality reduction methods into graph-based and non-graph-based methods. For non-graph-based methods, the classical algorithm Principal Component Analysis (PCA) [Smi02] converts potentially correlated variables into a set of linearly uncorrelated variables, known as principal components, by orthogonal transformation, and the core idea is to find the direction in which the data change the most. Multidimensional Scaling (MDS) [BG05] preserves the distances between pairs of data points as close as possible to their distances in the high-dimensional space during dimensionality reduction. However, when dealing with the reconstruction and unfolding of nonlinear data structures such as manifolds, they can only show projections from specific viewpoints, not fully unfolding them.

For graph-based methods, Locally-linear Embedding (LLE) [RS00] approximates each data point with a linear combination of its nearest neighbor points and thus fails to capture the overall structure of the data accurately. Laplacian Eigenmaps [BN03] constructs the Laplacian matrix, performs spectral decomposition, and selects the eigenvectors corresponding to the smallest non-zero eigenvalues as the embedding vectors. This method

effectively preserves the local structure but fails to unfold the global manifold structure. t-SNE [vdMH08] achieves dimensionality reduction by minimizing the probability distribution between data points in low- and high-dimensional spaces. LargeVis [TLZM16], a variant of t-SNE, is optimized for large datasets. UMAP [MHM18] assumes that the data is uniformly distributed on a locally connected Riemannian manifold and maintains the topology of the data for the neighborhoods of each data point. Space-based Manifold Approximation and Projection (SpaceMAP) [ZT22] is a variant of UMAP that solves the crowding problem by transforming the distance metric of the space. LargeVis, UMAP, and SpaceMAP emphasize maintaining compactness within classes while ensuring clear separation between different classes. Unlike the previously mentioned methods that only consider pairs of data points, TriMap [AW19] utilizes the relative distances of triples to maintain the local and global structure of the data.

The above graph-based methods all rely on the quality of the graph and suffer from distortion problems in embedding updates, resulting in embedding representations deviating from the original data. In contrast, DynoGraph uses an adaptive neighborhood graph that adjusts to the density and size of the dataset and, for the first time, introduces a dynamic graph modification process during dimensionality reduction, ensuring that the structure remains aligned with the original data.

5.3 Methodology

Fig. 5.2 provides the overall framework for DynoGraph, starting with the original data X. First, DynoGraph constructs an adaptive neighborhood graph (ANG) labeled G to describe the proximity between data points in X. Next, DynoGraph utilizes G to update the embedding Z in the low-dimensional space. As Z evolves, if node pairs exhibit deviations from the high-dimensional data structure in the low-dimensional space, DynoGraph constructs a modified graph G_{mod} using X, Z, and G to adjust the connections between these node pairs. This feedback loop establishes a mechanism where each new round of updates to Z is based on G_{mod}, which in turn is adjusted according to changes in Z.

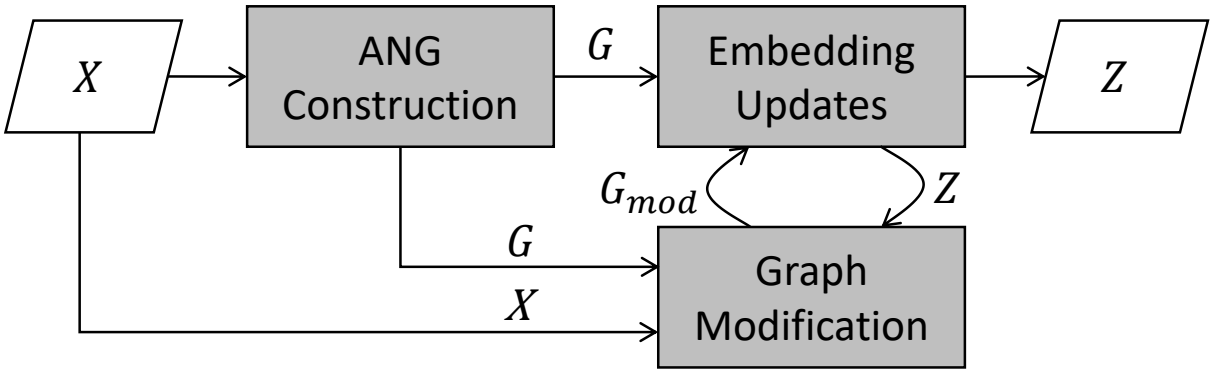


Figure 5.2: Overview of the DynoGraph algorithm.

Through this dynamic graph modification process, Z is continuously optimized and accurately aligned to high-dimensional data. To the best of our knowledge, this is the first time that dynamic graph modification techniques have been introduced into dimensionality reduction to mitigate distortion and maintain data fidelity.

5.3.1 Problem Definition

We aim to perform dimensionality reduction on a high-dimensional dataset to obtain a low-dimensional embedding while preserving the intrinsic structure of the data. The dataset is defined as $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \in \mathbb{R}^{n \times d}$, where each data point $\mathbf{x}_i \in \mathbb{R}^d$ is a vector in d-dimensional space, and n denotes the number of data points. The objective is to compute a corresponding low-dimensional embedding $Z = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n\} \in \mathbb{R}^{n \times m}$, where each embedded point $\mathbf{z}_i \in \mathbb{R}^m$ is a vector in m-dimensional space.

5.3.2 Adaptive Neighborhood Graph

Adaptive Neighborhood Graph (ANG) constructs a graph $G(V, E)$ from a dataset X. Each vertex in the vertex set $V = \{v_1, \ldots, v_n\}$ corresponds to a data point in X, and the edge set E represents proximity between data points. The following outlines the main steps of ANG, with details as shown in Algorithm 5:

Adaptive ϵ_i Calculation

Traditional ϵ-NNG relies on a globally fixed ϵ or the heuristic rule based on the average distance to its k-th nearest neighbor [İnk23]. However, choosing the appropriate k remains a challenge. Empirical evidence suggests that adapting the number of nearest neighbors to $\lfloor\sqrt{n}\rfloor$ can adjust to the size of the dataset, thus avoiding the limitation of fixed k [DHS12].

Inspired by these studies, we propose a core innovation of ANG: an adaptive ϵ_i is calculated for each point $\mathbf{x}_i$. ANG determines ϵ_i based on the average distance to its $\lfloor\sqrt{n}\rfloor$-th nearest neighbor. This approach adapts the neighborhood range according to the size of the dataset and also enhances flexibility and accuracy when handling datasets with varying densities.

Additionally, we employ nearest neighbor search [JDS11] to efficiently construct the graph. For each data point $\mathbf{x}_i$, the nearest neighbor search returns a list of neighbors $\mathcal{N}_i$ and their corresponding distances D_i in ascending order of distance. To ensure sufficient candidate neighbors to calculate ϵ_i and subsequently determine mutual neighbors, we set the target number of neighbors for the nearest neighbor search to $3\lfloor\sqrt{n}\rfloor$, as this multiplier of 3 is an empirical choice [vdMH08].

Edge Formation

In the ANG algorithm, edge formation is based on two criteria. Firstly, the algorithm ensures that each node is connected to at least five nearest neighbors. This criterion is designed to avoid numerous isolated or fragmented nodes, especially in sparser regions, and to ensure a certain extent of basic graph connectivity [TdSL00]. Additional connections are established based on the mutual nearest neighbors criterion, where an edge is formed between nodes v_i and v_j only if their distance is less than the minimum of ϵ_i and ϵ_j. This selective criterion ensures that only the most relevant and meaningful relationships are retained.

Algorithm 5: Adaptive Neighborhood Graph

Input: Dataset $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathbb{R}^{n \times d}$
Output: Graph $G(V, E)$

Initialize $G(V, E)$ with $V = \{v_1, \ldots, v_n\}$ and $E = \emptyset$
for $i = 1$ *to* n **do**
    `// Nearest neighbor search`
    $(\mathcal{N}_i, D_i) = \text{NearestNeighbors}(\mathbf{x}_i, \lfloor 3\sqrt{n} \rfloor)$
    `// Adaptive` ϵ `calculation`
    $\epsilon_i = \text{Mean}(D_i[: \lfloor \sqrt{n} \rfloor])$
    `// Edge formation`
    for $k = 1$ *to* $|\mathcal{N}_i|$ **do**
        $j = \mathcal{N}_i[k]$
        `// Ensure minimum five neighbors`
        if $k \leq 5$ **then**
            $E = E \cup \{(v_i, v_j)\}$
        `// Mutual nearest neighbors`
        else if $D_i[k] \leq \min(\epsilon_i, \epsilon_j)$ **then**
            $E = E \cup \{(v_i, v_j)\}$
`// Ensure graph connectivity`
Identify disconnected components $\{C_1, \ldots, C_g\}$ in G
if $g > 1$ **then**
    $E_{\text{candidate}} = \{(v_r, v_s, \text{distance}(v_r, v_s)) \mid v_r \in C_p, v_s \in C_q, p \neq q\}$
    $E_{\text{mst}} = \text{Kruskal}(E_{\text{candidate}})$
    $E = E \cup E_{\text{mst}}$
return $G(V, E)$

Graph Connectivity

In the final stage, ANG identifies disconnected components of the graph. If multiple components exist, it forms a set $E_{\text{candidate}}$ of candidate edges weighted by the minimum distance between nodes v_r and v_s across different components. Kruskal's algorithm [CLRS09] is then applied to $E_{\text{candidate}}$ to select a subset of these edges, which forms the MST, denoted as E_{mst}. This subset connects all disconnected components with minimal total weight and is integrated into the existing set of edges E, ensuring connectivity of the graph G. This crucial step captures the complete topology of the data and preserves global structure, which is often neglected in dimensionality reduction techniques.

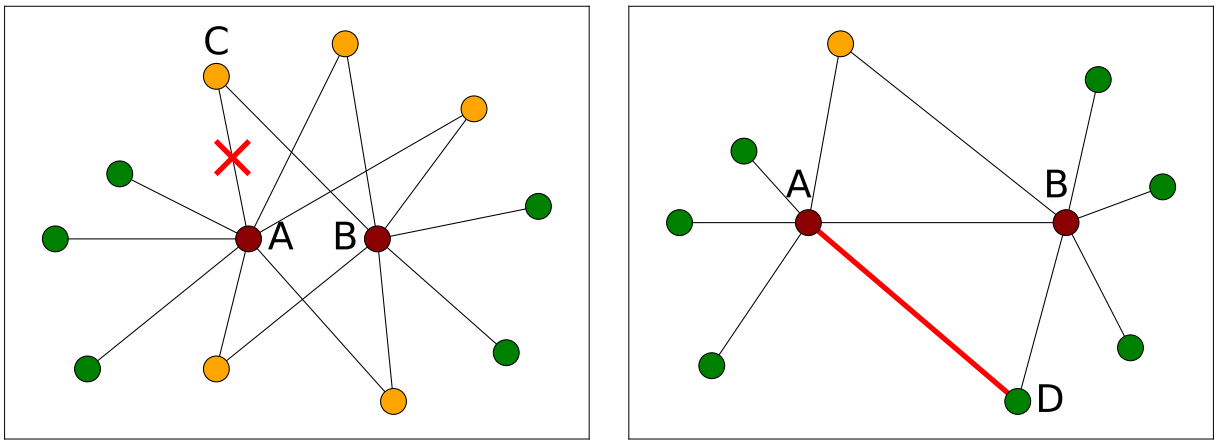
Fig. 5.1 illustrates the ANG construction process when the condition requiring each point to connect to at least five nearest neighbors is omitted. Fig. 5.1(g) shows the graph where each point connects to its neighbors within the adaptive ϵ_i-range, ensuring a relatively balanced number of neighbors across points, regardless of density variations. Fig. 5.1(h) builds on Fig. 5.1(g) by imposing the mutual nearest neighbors constraint, which accurately identifies the three components. Finally, Fig. 5.1(i) demonstrates the use of MST to connect the distinct components with the minimum and closest edges required to ensure graph connectivity.

5.3.3 Graph Modification

Determining Adaptive Threshold

In graph modification, it is crucial to accurately identify the vertex pairs that need to be adjusted, particularly to distinguish between *far* and *near* vertex pairs based on proximity and connection in the low-dimensional space. It is common practice to compute the distances between all vertex pairs in the low-dimensional space, flatten these distances into an array, sort them in ascending order, and then manually set predefined thresholds. For example, the first 5% of the minimum distances can be defined as *near*, while the last 10% of the maximum distances can be defined as *far*. However, different datasets can vary greatly in density. For example, some datasets may be suitable for determining the first 1% of distance as *near*, while others may need to consider a larger threshold of 10% due to different densities. Fixed thresholds may not accurately capture proximity in all cases. In addition, this approach does not take into account the inherent structure of the original data and the graph constructed from it.

To address this issue, we propose a data-driven strategy for deriving adaptive thresholds from the data distribution of the high-dimensional space and graph structure. Instead of defining the threshold directly in the low-dimensional space, we returned to the high-dimensional data to analyze it and then applied it to the low-dimensional space. It maintains the consistency of spatial proximity in both high- and low-dimensional spaces



(a)Edge Deletion (b)Edge Insertion

Figure 5.3: Edge modification strategies.

during the dimensionality reduction process. Specifically, we define the maximum distance for vertex pairs with edges in the high-dimensional space as *far*, labeled $dist_{edge}$, i.e., vertex pairs beyond this distance are not connected. The minimum distance for vertex pairs without edges is *near*, labeled $dist_{noedge}$, i.e., vertex pairs less than this distance are all connected. We record the location of these two distances in the flattened, sorted high-dimensional distance array, labeled P_{far} and P_{near}, respectively. Finally, we use the same locations in the flattened, sorted low-dimensional distance array to find T_{far} and T_{near}, corresponding to the adaptive distance thresholds in the low-dimensional space. The T_{far} is a lower bound on the distance in the low-dimensional space that identifies vertex pairs that should not be connected, while T_{near} is an upper bound on the distance that identifies vertex pairs that should be connected.

Reducing Connections of Vertex Pairs without Edges

The distortion problem is serious when vertex pairs that are not connected by edges in high-dimensional space become *near* ($< T_{near}$) in the low-dimensional space. This phenomenon contradicts the original data, as vertex pairs not connected in high-dimensional spaces are typically due to their greater distances. As shown in Fig. 5.3(a), the vertex pair

A and B (denoted in red) are *near* in the low-dimensional space, but they are not directly connected. To mitigate such distortion, the connections that lead to improper closeness need to be reduced. This reduction can be achieved by identifying their common neighbors (denoted in orange) and randomly removing one connection. For example, the connection between common neighbor C and vertex A is highlighted in a red cross mark, indicating that the edge should be deleted to reduce the connection between A and B. Reduce connections only if there are multiple common neighbors to keep the graph connectivity.

Strengthening Connections of Vertex Pairs with Edges

Distortions also occur when vertex pairs that are connected by edges in high-dimensional space become *far* ($> T_{far}$) in low-dimensional space. Fig. 5.3(b) illustrates such a scenario where the vertex pairs A and B are *far* in the low-dimensional space and are directly connected. To mitigate this distortion, we strengthen the connections by inserting an edge between a vertex and a non-common neighbor vertex (denoted in green) of its counterpart. For example, D is a non-common neighbor vertex of B. As indicated in the red line, we insert a new edge between D and A.

Algorithm 6 describes in detail the graph modification process, which adjusts the connections between vertex pairs based on their relative distances in the high- and low-dimensional spaces and the original graph G constructed by Algorithm 5. First, the algorithm calculates the flattened distance array D_X and D_Z for all vertex pairs in the high- and low-dimensional spaces (line 2). Relying on D_X, two positions P_{far} and P_{near} are found, which represent the maximum distances for vertex pairs with edges $dist_{edge}$ and the minimum distances for vertex pairs without edges $dist_{noedge}$, respectively (lines 3-6). These positions are then applied in D_Z to determine the adaptive thresholds T_{far} and T_{near} by using the QuickSelect algorithm [CFNTV16] in the low-dimensional space, providing a guide for the adjustment of the connections (lines 7-8). For each vertex pair, the algorithm decides whether to strengthen or reduce the connection between

Algorithm 6: Graph Modification

Input: Dataset $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathbb{R}^{n \times d}$, Graph $G(V, E)$, Embedding $Z = \{\mathbf{z}_1, \ldots, \mathbf{z}_n\} \in \mathbb{R}^{n \times m}$

Output: Modified graph $G_{mod}(V, E_{mod})$

Initialize $G_{mod}(V, E_{mod})$ with $E_{mod} = E$

`// Determining adaptive threshold`

Compute flattened distance array of all vertex pairs in X and Z: D_X, D_Z

$dist_{edge} = \max(\{d | (v_i, v_j) \in E, d = D_X[i, j]\})$

$dist_{noedge} = \min(\{d | (v_i, v_j) \notin E, d = D_X[i, j]\})$

$P_{far} = |\{(i, j) | D_X[i, j] \leq dist_{edge}\}|$

$P_{near} = |\{(i, j) | D_X[i, j] \leq dist_{noedge}\}|$

$T_{far} =$ QuickSelect(D_Z, P_{far})

$T_{near} =$ QuickSelect(D_Z, P_{near})

for *each pair* (v_i, v_j) **do**

  $N_i = \{v_k \mid (v_i, v_k) \in E_{mod}\}$

  $N_j = \{v_k \mid (v_j, v_k) \in E_{mod}\}$

  $N_{common} = N_i \cap N_j$

  $N_{nc_i} = N_i \setminus N_{common}$

  $N_{nc_j} = N_j \setminus N_{common}$

  `// Strengthening connections`

  if $D_Z[i, j] > T_{far}$ ***and*** $(v_i, v_j) \in E$ **then**

    $Candidate_{ins} = \{(v_i, v_k) | v_k \in N_{nc_j}\} \cup \{(v_j, v_k) | v_k \in N_{nc_i}\}$

    randomly select (v, v_k) from $Candidate_{ins}$

    $E_{mod} = E_{mod} \cup \{(v, v_k)\}$

  `// Reducing connections`

  else if $D_Z[i, j] < T_{near}$ ***and*** $(v_i, v_j) \notin E$ ***and*** $|N_{common}| > 1$ **then**

    $Candidate_{del} = \{(v_i, v_k), (v_j, v_k) | v_k \in N_{common}\}$

    randomly select (v, v_k) from $Candidate_{del}$

    $E_{mod} = E_{mod} \setminus \{(v, v_k)\}$

Return $G_{mod}(V, E_{mod})$

them based on these two thresholds. If the distance between a vertex pair with an edge in the low-dimensional space exceeds T_{far}, an edge will be randomly inserted between one of their non-common neighbors (lines 9-18). Conversely, if the distance between a vertex pair without an edge is less than T_{near} and more than one common neighbor exists, the connection with one common neighbor will be randomly deleted (lines 19-22). Eventually, the algorithm returns the modified graph $G_{mod}(V, E_{mod})$, where E_{mod} denotes the adjusted set of edges.

5.3.4 Objective Function for Embedding

In this section, we integrate three critical components: Vertex Pair Probability Function (VPPF), Graph Structure Representation Measure (GSRM), and Weighted Majorization (WM). The VPPF establishes a probabilistic basis for modeling the proximity of vertex pairs in the embedding space, reflecting their connectivity in the original graph. The GSRM, as an objective function, then utilizes the VPPF to attract vertex pairs with edges and repel vertex pairs without edges, evaluating the overall representation quality of the graph structure. Further, WM is employed to approximate the solution of the GSRM, providing the necessary parameters for effective embedding.

Vertex Pair Probability Function

The Vertex Pair Probability Function (VPPF) mathematically models the probability based on the distance between vertex pairs in a low-dimensional space, considering whether an edge exists between them in the original graph. The VPPF is denoted by $P_{vp}(||\mathbf{z}_i - \mathbf{z}_j||)$, and its expression is:

$$P_{vp}(||\mathbf{z}_i - \mathbf{z}_j||) = \begin{cases} \frac{1}{2} - \frac{1}{2}\,\mathrm{erf}\left(\frac{||\mathbf{z}_i - \mathbf{z}_j|| - \mu}{\sqrt{2}\sigma}\right) \text{ if } (v_i, v_j) \in E \\ \frac{1}{2} + \frac{1}{2}\,\mathrm{erf}\left(\frac{||\mathbf{z}_i - \mathbf{z}_j|| - \mu}{\sqrt{2}\sigma}\right) \text{ if } (v_i, v_j) \notin E \end{cases} \tag{5.1}$$

where $\mathbf{z}_i$ and $\mathbf{z}_j$ denote the embedding vectors of the vertex pair v_i and v_j, respectively. $||\mathbf{z}_i - \mathbf{z}_j||$ denotes the Euclidean distance between these two vertices in the low-

dimensional space, quantifying their proximity. By setting μ to 0 and σ to 1, we exploit the properties of the standard Gaussian distribution. This choice simplifies the model computation and ensures consistency of probability measures and standardization of the low-dimensional space. By introducing the error function (erf), the VPPF transforms the Euclidean distance between embedding vectors into a probability value between 0 and 1. For the vertex pair with an edge $(v_i, v_j) \in E$, the probability monotonically decreases as the distance increases, ranging from 1 to 0. Therefore, a higher probability value indicates that the vertex pair with an edge is close in the lower-dimensional space. This reflects that vertex pairs connected in the high-dimensional space should be relatively close in the lower-dimensional space. For the vertex pair without an edge $(v_i, v_j) \notin E$, the probability monotonically increases as the distance increases, ranging from 0 to 1. Thus, a higher probability value indicates that the vertex pair without an edge is farther apart in the lower-dimensional space. This result suggests that vertex pairs not connected in the high-dimensional space should be relatively farther away in the low-dimensional space. This function has a gentle slope at the ends and a larger slope in the middle region, thus naturally focusing the optimization on those vertex pairs with intermediate distances that require adjustment.

Graph Structure Representation Measure

Based on VPPF, the Graph Structure Representation Measure (GSRM) is designed as an objective function to evaluate the quality of graph structure representation in low-dimensional space. By transforming local estimations of the vertex pair probabilities into a comprehensive measure, GSRM evaluates how well the embedding maintains the original graph structure. The mathematical definition of GSRM is as follows:

$$\text{GSRM}(Z) = \frac{1}{N_{vp}} \left[\sum_{1 \le i < j \le n} -\log_2 \left(P_{vp}(||\mathbf{z}_i - \mathbf{z}_j||) \right) \right] \tag{5.2}$$

where N_{vp} represents the total number of vertex pairs, serving as a normalization factor for the overall measure. The core of GSRM utilizes the negative logarithm of $P_{vp}(||\mathbf{z}_i -$

$\mathbf{z}_j||$), which further amplifies the effect of the VPPF on this measure. For vertex pairs with edges, we expect them to be as close as possible in the low-dimensional space so that their VPPF is approaching 1. For vertex pairs without edges, we expect them to be as distant as possible so that their VPPF is also approaching 1. In this way, the minimization of GSRM aims to apply an attractive force to vertex pairs with edges and a repulsive force to vertex pairs without edges, thus ensuring that the relative positions of the vertex pairs in the low-dimensional space accurately reflect the connection patterns in the original graph structure.

Weighted Majorization for Embedding

Weighted Majorization (WM) [dL05], derived from MDS [BG05], is optimized for non-metric stress minimization. It handles non-metric distance functions while allowing differential weighting of vertex pair distances. By minimizing the weighted sum of squared differences between actual and target distances in the low-dimensional space, WM effectively preserves the structural properties of the graph. The optimization objective of WM is expressed as follows:

$$\text{WM}(Z) = \sum_{1 \leq i < j \leq n} w_{ij}(||\mathbf{z}_i - \mathbf{z}_j|| - d_{ij})^2 \tag{5.3}$$

where w_{ij} is the weight assigned to each vertex pair and d_{ij} is their target distance in the low-dimensional space. The parabola of the quadratic error function of WM is similar to the curve of GSRM, which supports the use of WM as a suitable approximation of GSRM. It is consistent with the approach taken in [PBB20], where WM is also used to approximate the objective function, which is very similar to the GSRM function. By aligning the first and second partial derivatives of GSRM and WM to the Euclidean distances between vertex pairs, we ensure that the approximation of WM precisely follows the optimization trajectory of GSRM. The derivative equations establishing this connection are:

$$\frac{\partial\left[-\log_2\left(P_{vp}(||\mathbf{z}_i-\mathbf{z}_j||)\right)\right]}{\partial||\mathbf{z}_i-\mathbf{z}_j||}=\frac{\partial\left[w_{ij}(||\mathbf{z}_i-\mathbf{z}_j||-d_{ij})^2\right]}{\partial||\mathbf{z}_i-\mathbf{z}_j||} \tag{5.4}$$

$$\frac{\partial^2\left[-\log_2\left(P_{vp}(||\mathbf{z}_i-\mathbf{z}_j||)\right)\right]}{\partial||\mathbf{z}_i-\mathbf{z}_j||^2}=\frac{\partial^2\left[w_{ij}(||\mathbf{z}_i-\mathbf{z}_j||-d_{ij})^2\right]}{\partial||\mathbf{z}_i-\mathbf{z}_j||^2} \tag{5.5}$$

On the right side of Eq. (5.4), the first-order derivative of WM with respect to the Euclidean distance $||\mathbf{z}_i-\mathbf{z}_j||$ is formulated as follows:

$$\frac{\partial\left[w_{ij}(||\mathbf{z}_i-\mathbf{z}_j||-d_{ij})^2\right]}{\partial||\mathbf{z}_i-\mathbf{z}_j||}=2w_{ij}(||\mathbf{z}_i-\mathbf{z}_j||-d_{ij}) \tag{5.6}$$

The second-order derivative of WM in Eq. (5.5) is formulated as follows:

$$\frac{\partial^2\left[w_{ij}(||\mathbf{z}_i-\mathbf{z}_j||-d_{ij})^2\right]}{\partial||\mathbf{z}_i-\mathbf{z}_j||^2}=2w_{ij} \tag{5.7}$$

For the left side of Eq. (5.4), integrating the VPPF for vertex pairs with edges, the first-order derivative of GSRM with respect to the Euclidean distance $||\mathbf{z}_i-\mathbf{z}_j||$ as follows:

$$\frac{\partial\left[-\log_2\left(P_{vp}(||\mathbf{z}_i-\mathbf{z}_j||)\right)\right]}{\partial||\mathbf{z}_i-\mathbf{z}_j||}=\frac{\sqrt{2}e^{-\frac{||\mathbf{z}_i-\mathbf{z}_j||^2}{2}}}{\sqrt{\pi}\ln 2\left(1-\operatorname{erf}\left(\frac{||\mathbf{z}_i-\mathbf{z}_j||}{\sqrt{2}}\right)\right)} \tag{5.8}$$

For vertex pairs without edges, the first-order derivative of GSRM can be described as follows:

$$\frac{\partial\left[-\log_2\left(P_{vp}(||\mathbf{z}_i-\mathbf{z}_j||)\right)\right]}{\partial||\mathbf{z}_i-\mathbf{z}_j||}=\frac{\sqrt{2}e^{-\frac{||\mathbf{z}_i-\mathbf{z}_j||^2}{2}}}{\sqrt{\pi}\ln 2\left(1+\operatorname{erf}\left(\frac{||\mathbf{z}_i-\mathbf{z}_j||}{\sqrt{2}}\right)\right)} \tag{5.9}$$

For vertex pairs with edges, the second-order derivative of GSRM in Eq. (5.5) is formulated as follows:

$$\frac{\partial^2\left[-\log_2\left(P_{vp}(||\mathbf{z}_i-\mathbf{z}_j||)\right)\right]}{\partial||\mathbf{z}_i-\mathbf{z}_j||^2}=\frac{2e^{-||\mathbf{z}_i-\mathbf{z}_j||^2}}{\pi\ln 2\left(1-\operatorname{erf}\left(\frac{||\mathbf{z}_i-\mathbf{z}_j||}{\sqrt{2}}\right)\right)^2}-\frac{\sqrt{2}||\mathbf{z}_i-\mathbf{z}_j||e^{-\frac{||\mathbf{z}_i-\mathbf{z}_j||^2}{2}}}{\sqrt{\pi}\ln 2\left(1-\operatorname{erf}\left(\frac{||\mathbf{z}_i-\mathbf{z}_j||}{\sqrt{2}}\right)\right)} \tag{5.10}$$

For vertex pairs without edges, the second-order derivative of GSRM is shown below:

$$\frac{\partial^2\left[-\log_2\left(P_{vp}(||\mathbf{z}_i-\mathbf{z}_j||)\right)\right]}{\partial||\mathbf{z}_i-\mathbf{z}_j||^2}=\frac{2e^{-||\mathbf{z}_i-\mathbf{z}_j||^2}}{\pi\ln 2\left(1+\operatorname{erf}\left(\frac{||\mathbf{z}_i-\mathbf{z}_j||}{\sqrt{2}}\right)\right)^2}+\frac{\sqrt{2}||\mathbf{z}_i-\mathbf{z}_j||e^{-\frac{||\mathbf{z}_i-\mathbf{z}_j||^2}{2}}}{\sqrt{\pi}\ln 2\left(1+\operatorname{erf}\left(\frac{||\mathbf{z}_i-\mathbf{z}_j||}{\sqrt{2}}\right)\right)} \tag{5.11}$$

By substituting the second-order derivative formulas of GSRM and WM for vertex pairs with edges (Eqs. (5.10) and (5.7)) and vertex pairs without edges (Eqs. (5.11) and (5.7)), respectively, into Eq. (5.5), we can derive the expression for w_{ij}:

$$w_{ij}=\frac{\frac{2}{\pi\ln 2}e^{-\|\mathbf{z}_i-\mathbf{z}_j\|^2}}{\left(1\mp\operatorname{erf}\left(\frac{\|\mathbf{z}_i-\mathbf{z}_j\|}{\sqrt{2}}\right)\right)^2}\mp\frac{\frac{\|\mathbf{z}_i-\mathbf{z}_j\|}{\sqrt{2\pi}\ln 2}e^{-\frac{(\|\mathbf{z}_i-\mathbf{z}_j\|)^2}{2}}}{\left(1\mp\operatorname{erf}\left(\frac{\|\mathbf{z}_i-\mathbf{z}_j\|}{\sqrt{2}}\right)\right)} \tag{5.12}$$

By integrating the first-order derivative expressions of GSRM and WM for vertex pairs with edges (Eqs. (5.8) and (5.6)) and vertex pairs without edges (Eqs. (5.9) and (5.6)), respectively, we obtain the formula for d_{ij} that incorporates the parameter w_{ij} as follows:

$$d_{ij}=\|\mathbf{z}_i-\mathbf{z}_j\|\mp\frac{\frac{1}{\sqrt{2\pi}\ln 2}e^{-\frac{\|\mathbf{z}_i-\mathbf{z}_j\|^2}{2}}}{w_{ij}\left(1\mp\operatorname{erf}\left(\frac{\|\mathbf{z}_i-\mathbf{z}_j\|}{\sqrt{2}}\right)\right)} \tag{5.13}$$

In the $\mp$ notation used in these equations, the minus sign $(-)$ indicates vertex pairs with edges, and the plus sign $(+)$ indicates vertex pairs without edges.

Vertices in the low-dimensional space are iteratively updated to optimize Eq. (5.3) by the following update rule for WM:

$$\mathbf{z}_i=\frac{\sum_{j\neq i}w_{ij}\left(\mathbf{z}_j+d_{ij}\frac{\mathbf{z}_i-\mathbf{z}_j}{||\mathbf{z}_i-\mathbf{z}_j||}\right)}{\sum_{j\neq i}w_{ij}} \tag{5.14}$$

Algorithm 7: DynoGraph

Input: Dataset $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathbb{R}^{n \times d}$, Target dimension m
Output: Embedding $Z = \{\mathbf{z}_1, \ldots, \mathbf{z}_n\} \in \mathbb{R}^{n \times m}$

Initialize Z with uniform distribution over $[0, 1]$
Construct graph $G(V, E)$ using **Algorithm 5**
Initialize $G_{current} = G$
for $iter = 0$ **to** $iter_{graph}$ **do**
    if $iter > 0$ **then**
        Construct modified graph $G_{mod}(V, E_{mod})$ using **Algorithm 6**
        $G_{current} = G_{mod}$
    repeat
        UpdateEmbedding(Z, $G_{current}$)
    until $iter_{embedding}$
return Z
Procedure *UpdateEmbedding (Z, G)*:
    foreach $(v_i, v_j) \in E$ **do**
        Compute w_{ij} and d_{ij} according to Eqs. (5.12) and (5.13)
        Update $\mathbf{z}_i$ and $\mathbf{z}_j$ according to Eq. (5.14)
        `// Negative sampling for` v_i `&` v_j
        for $v_k \in \{v_i, v_j\}$ **do**
            Select a random vertex v_l s.t. $(v_k, v_l) \notin E$
            Compute w_{kl} and d_{kl} according to Eqs. (5.12) and (5.13)
            Update $\mathbf{z}_k$ and $\mathbf{z}_l$ according to Eq. (5.14)

where $\frac{\mathbf{z}_i - \mathbf{z}_j}{||\mathbf{z}_i - \mathbf{z}_j||}$ is the unit vector pointing from $\mathbf{z}_j$ to $\mathbf{z}_i$. The position of vertex $\mathbf{z}_i$ in the low-dimensional space is refined through a weighted sum designed to adjust the actual distances between vertices closer to their target distances, thus iteratively improving the embedding. The convergence of WM [GKN04] proves its suitability as an approximation for GSRM.

5.3.5 Algorithm Overview

DynoGraph, as described in Algorithm 7, incorporates ANG, graph modification, and embedding updates to obtain a low-dimensional embedding of the high-dimensional data. Given a high-dimensional dataset X and target dimension m, it outputs the embedding

Z in the low-dimensional space. It starts with initializing Z using a uniform random distribution within the range [0,1] (line 1). Subsequently, it constructs the original graph $G(V, E)$ from X using Algorithm 5 (line 2). Next, it refines the embedding by iteratively updating Z based on G (lines 8-10). The procedure of updating the embedding involves calculating the weights w_{ij} and target distances d_{ij} for the vertex pair (v_i, v_j) with an edge according to Eqs. (5.12) and (5.13). The positions of $\mathbf{z}_i$ and $\mathbf{z}_j$ are then updated according to Eq. (5.14) (lines 13-15). Furthermore, negative sampling is implemented by randomly selecting a random vertex v_l, ensuring that for each $v_k \in \{v_i, v_j\}$, the edge $(v_k, v_l) \notin E$, and then applying the same update rule to z_k and z_l (lines 16-19). In the dynamic graph modification phase, Algorithm 6 combines X, Z, and G to construct the modified graph $G_{mod}(V, E_{mod})$, which in turn iteratively update Z based on G_{mod} (line 6). DynoGraph can effectively transform high-dimensional data into the target dimensional space while preserving the spatial proximity of the original data. The elegance of DynoGraph is its adaptive nature, requiring no external parameter tuning and completely relying on the inherent properties of the original data.

5.3.6 Complexity Analysis

The DynoGraph algorithm integrates ANG, graph modification, and embedding updates. Its computational complexity is analyzed as follows: The time complexity of ANG is reduced from $O(n^2)$ to $O(n \log n)$ by using nearest neighbor search; Graph modification computes the distances between all vertex pairs, resulting in a complexity of $O(n^2)$. During the embedding update, N_{edges}, the number of edges, are iteratively processed with a complexity of $O(N_{edges})$, leading to an embedding update complexity of $O(iter_{embedding} \cdot N_{edges})$. The dynamic graph modification phase combines graph modification and embedding updates, with an overall complexity of $O(iter_{graph} \cdot (n^2 + iter_{embedding} \cdot N_{edges}))$. Here, $iter_{graph}$ and $iter_{embedding}$ are set to 2 and 200 by default, respectively. In summary, the overall asymptotic runtime complexity of DynoGraph is $O(n^2)$.

5.4 Experiments

5.4.1 Experimental Setup

Experimental Environment

The proposed DynoGraph algorithm was implemented in Python, utilizing `faiss-gpu`, the GPU implementation of the Faiss library [JDJ19], for nearest neighbor search. All experiments related to this algorithm were conducted on a machine equipped with an Intel Core i9-12900H CPU (14 cores, 2.50 GHz), 64 GB of RAM, and an NVIDIA GeForce RTX 3080 Ti GPU. The code repository is available at `https://github.com/Qian-Lily/DynoGraph`. All experiments aim to learn a two-dimensional embedding representation of the original data for visualization and comprehensive comparison.

Datasets

We selected seven real-world datasets from various domains to demonstrate the broad applicability of DynoGraph. The WarpPIE10P [SBB02] dataset is used for facial recognition. From the UCI Machine Learning Repository [DG17], we introduced Landsat-Satellite and Human Activity Recognition Using Smartphones (HAR) datasets, which represent satellite images and activity recognition using time series data collected from smartphone sensors, respectively. From the COIL project at Columbia University, we used the COIL-20 [NNM96b] and COIL-100 [NNM96a] datasets, which consist of images of objects captured from different angles, commonly used for object recognition tasks ranging from small to large object sizes. Additionally, we used the Fashion-MNIST [XRV17] and MNIST [LCB98] datasets for image classification of clothing and handwritten digits, respectively. These datasets have samples ranging from 210 to 70,000, dimensions ranging from 3 to 49,152, and classes ranging from 1 to 100. The statistics of these datasets are briefly summarized in Table 5.1. For image data, we scaled the pixel values to the range $[0, 1]$ by dividing by the maximum pixel value. For multivariate data, we prepro-

Table 5.1: Statistics of datasets used in DynoGraph.

Dataset	#Samples	#Dimensions	#Classes
3D Scurve-hole	8540	3	1
WarpPIE10P	210	2420	10
COIL-20	1440	16384	20
LandsatSatellite	6435	36	6
COIL-100	7200	49152	100
HAR	10299	561	6
Fashion-MNIST	70000	784	10
MNIST	70000	784	10

cessed each dataset by applying the `StandardScaler`[36] from the `scikit-learn` library, which standardizes each attribute by centering it to a mean of 0 and scaling it to a unit standard deviation of 1 [GLH15].

Comparison Methods

To evaluate the effectiveness of DynoGraph, we conducted a comprehensive comparative analysis against nine representative or state-of-the-art dimensionality reduction techniques. These included PCA[37] [Smi02], MDS[38] [BG05], LLE[39] [RS00], Eigenmaps[40] [BN03], t-SNE[41] [vdMH08], LargeVis[42] [TLZM16], UMAP[43] [MHM18], TriMap[44] [AW19], and SpaceMAP[45] [ZT22]. To ensure consistency and fairness in comparisons, we parameterized all comparison methods as recommended in the corresponding publications. Notably, DynoGraph does not require any input parameters.

[36] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`
[37] `https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html`
[38] `https://scikit-learn.org/stable/modules/generated/sklearn.manifold.MDS.html`
[39] `https://scikit-learn.org/stable/modules/generated/sklearn.manifold.LocallyLinearEmbedding.html`
[40] `https://scikit-learn.org/stable/modules/generated/sklearn.manifold.SpectralEmbedding.html`
[41] `https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html`
[42] `https://github.com/lferry007/LargeVis`
[43] `https://github.com/lmcinnes/umap`
[44] `https://github.com/eamid/trimap`
[45] `https://github.com/zuxinrui/SpaceMAP`

Evaluation Metric

We evaluate the effectiveness of dimensionality reduction algorithms using two global evaluation metrics, including Procrustes analysis [BB22], described in Chapter 2.3.8, and AMI score [VEB10], described in Chapter 2.3.2, along with a local evaluation metric, the k-NN classifier accuracy [CD21], detailed in Chapter 2.3.4.

We generate two-dimensional ground truth coordinates for the synthetic dataset and fold them into three-dimensional space through a nonlinear transformation. Procrustes analysis is used to assess the difference between the ground truth coordinates and the embeddings obtained by the dimensionality reduction algorithms.

For real-world datasets, we perform k-means clustering (initialized with k-means++) and evaluate the clustering performance of the low-dimensional embeddings using AMI score. Additionally, the k-NN classifier accuracy is used to evaluate local structure preservation, where the choice of k depends on the number of data points in the dataset, ranging from 1% to 20%. The k-NN classifier accuracy is evaluated using 10-fold stratified cross-validation. All experiments were repeated ten times, and we report the average results.

5.4.2 Results on Synthetic Dataset

The *3D Scurve-hole* dataset, shown in Fig. 5.4, originates from a two-dimensional uniform distribution rectangle with a hole in the center, which is then folded into an S-curve in three-dimensional space. This synthetic dataset evaluates several key properties of the dimensionality reduction algorithm. First is the unfolding, which examines whether the algorithm can faithfully and fully unfold the three-dimensional manifold back to two-dimensional ground truth. It requires the algorithm to balance global and local preservation: global preservation focuses on maintaining the overall structure and shape, while local preservation focuses on similar relationships between neighboring points. Second, the hole provides a key topological challenge. The algorithm needs to recognize and handle this hole correctly to avoid incorrectly establishing connections or disconnections

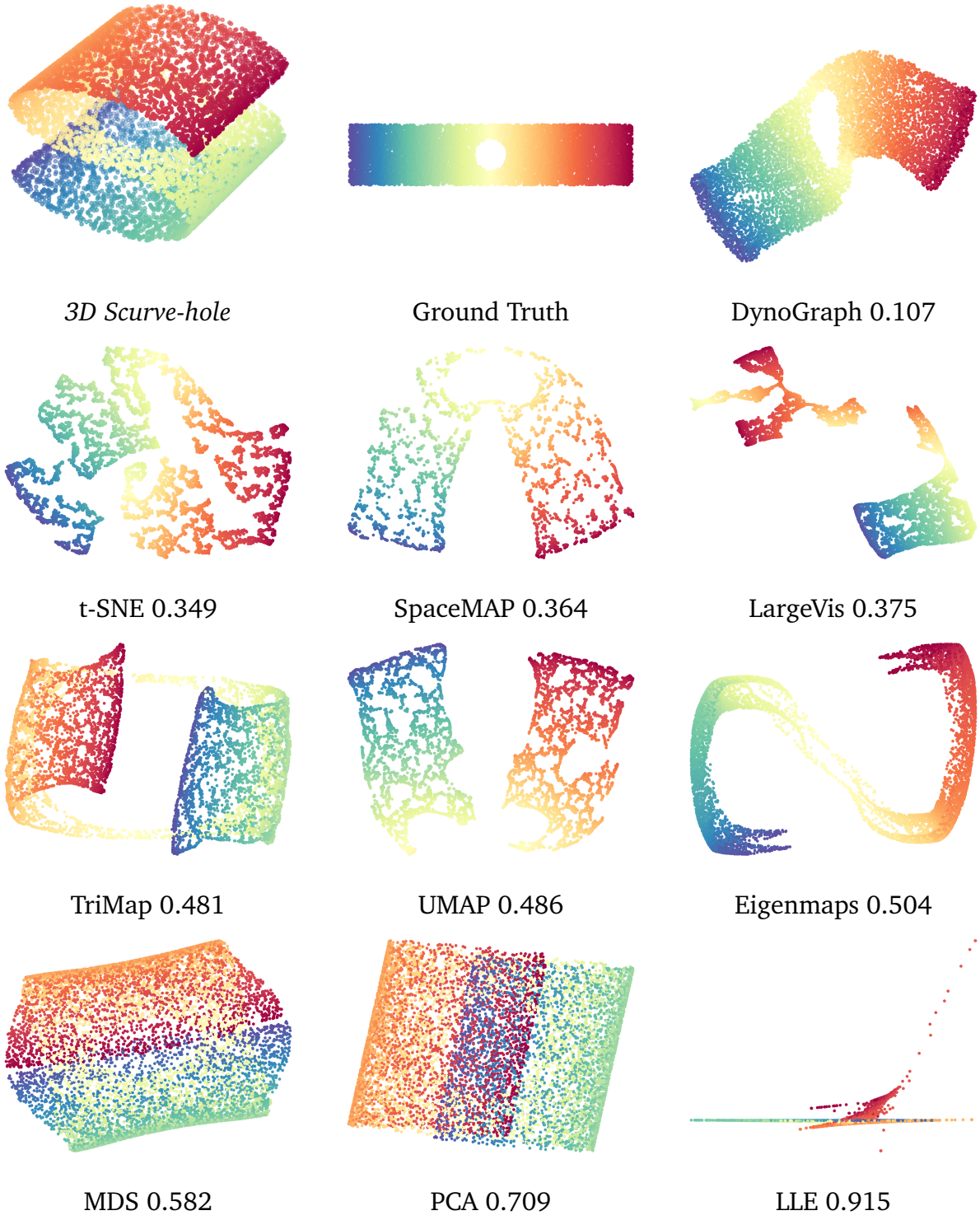


Figure 5.4: Visualization of synthetic *3D Scurve-hole* dataset, sorted by Procrustes error (Lower is better).

in a region with no data, which tests its ability to handle complex data structures. To evaluate these aspects, we use Procrustes analysis to quantitatively compare the similarity between the low-dimensional embedding and ground truth. Meanwhile, visualization is used to qualitatively evaluate whether the data layout, shape, and key topologies in the embedding representation are accurately preserved.

As shown in Fig. 5.4, DynoGraph almost fully unfolds the dataset, maintaining the global rectangular structure indicated by the smooth color gradient transition from blue to red and correctly identifying the hole. Besides, DynoGraph maintains the uniform distribution of the data and obtains the lowest Procrustes error (0.107), demonstrating advantages in global and local preservation. However, some algorithms such as t-SNE, SpaceMAP, and UMAP appear to hallucinate a local cluster structure, exhibit numerous voids that are not perceivable in the ground truth, and fail to preserve uniform distribution. LargeVis and UMAP misinterpret the hole as a signal for data segmentation, splitting the dataset into discrete segments and failing to preserve the overall global structure. Eigenmaps, MDS, and PCA show a 2D projection of the original 3D structure from a specific viewpoint rather than unfolding it, and completely ignore the hole, with the Procrustes error rising from 0.504 to 0.709. TriMap identifies the hole but is significantly expanded and not fully unfolded, presenting a 3D-like ring-wrapped shape with Procrustes errors of 0.481. LLE shows compression or stretching of data points in specific directions as a direct result of its attempt to maintain local linear relationships in the low-dimensional space. However, this approach sacrifices the global structure of the data, resulting in the highest Procrustes error of 0.915.

5.4.3 Results on Real-World Datasets

Clustering Analysis

Table 5.2 presents a comparative analysis of clustering accuracy measured by AMI score [VEB10] on real-world datasets. DynoGraph ranks first on three out of seven datasets, with an overall average score of 0.72, outperforming all baselines. In par-

Table 5.2: AMI score on real-world datasets: mean ± standard deviation; highest scores in bold; ranking in parentheses (lower is better). Notations: N/A (No Results), O/M (Out-of-Memory, >64GB).

Dataset	PCA	LLE	MDS	Eigenmaps	TriMap
WarpPIE10P	0.04±0.00 (10)	0.29±0.00 (4)	0.07±0.01 (9)	0.10±0.00 (8)	0.27±0.02 (5)
COIL-20	0.62±0.00 (8)	0.85±0.04 (4)	0.62±0.01 (8)	0.56±0.00 (10)	0.85±0.01 (4)
LandsatSatellite	0.47±0.00 (10)	0.61±00.0 (8)	0.60±0.01 (9)	0.63±0.00 (3)	0.62±0.00 (5)
COIL-100	0.60±0.00 (9)	N/A	0.64±0.00 (7)	0.61±0.00 (8)	0.88±0.00 (3)
HAR	0.46±0.00 (9)	0.41±0.01 (10)	0.49±0.01 (8)	0.67±0.00 (6)	0.69±0.00 (5)
Fashion-MNIST	0.42±0.00 (7)	0.36±0.00 (8)	O/M	O/M	0.60±0.01 (5)
MNIST	0.36±0.00 (8)	0.67±0.00 (7)	O/M	O/M	0.79±0.00 (5)
Average	0.42±0.00 (10)	0.53±0.01 (7)	0.48±0.01 (9)	0.51±0.00 (8)	0.67±0.01 (5)

Dataset	LargeVis	SpaceMAP	t-SNE	UMAP	DynoGraph
WarpPIE10P	0.24±0.01 (7)	0.26±0.00 (6)	0.30±0.01 (3)	0.31±0.01 (2)	**0.50±0.03 (1)**
COIL-20	0.81±0.02 (7)	0.87±0.00 (2)	0.84±0.01 (6)	0.87±0.01 (2)	**0.90±0.01 (1)**
LandsatSatellite	**0.64±0.01 (1)**	0.62±0.00 (5)	0.62±0.01 (5)	**0.64±0.00 (1)**	0.63±0.02 (3)
COIL-100	0.86±0.00 (5)	**0.89±0.00 (1)**	0.88±0.00 (3)	**0.89±0.00 (1)**	0.85±0.01 (6)
HAR	**0.72±0.04 (1)**	0.67±0.00 (6)	0.70±0.03 (4)	0.71±0.03 (2)	0.71±0.03 (2)
Fashion-MNIST	0.61±0.02 (4)	0.62±0.00 (2)	0.56±0.02 (6)	0.62±0.01 (2)	**0.64±0.01 (1)**
MNIST	0.84±0.02 (3)	**0.91±0.00 (1)**	0.76±0.04 (6)	0.85±0.01 (2)	0.81±0.01 (4)
Average	0.67±0.02 (4)	0.69±0.00 (3)	0.67±0.02 (6)	0.70±0.01 (2)	**0.72±0.02 (1)**

ticular, on WarpPIE10P dataset, the AMI score of DynoGraph improved by approximately 61% compared to the runner-up UMAP. UMAP, SpaceMAP, and LargeVis rank first on two of the seven datasets, with average rankings ranging from 2nd to 4th. TriMap and t-SNE perform stably across datasets, while classical dimensionality reduction algorithms such as LLE, Eigenmaps, MDS, and PCA perform poorly on average. MDS and Eigenmaps suffer from out-of-memory errors when dealing with large datasets (70k), showing the limitations when facing big data. In addition, LLE fails to produce valid results on COIL-100 dataset due to a singular weight matrix, revealing its limitations when handling highly linearly correlated feature data.

Classification Analysis

Fig. 5.5 illustrates the k-NN classifier accuracy of various dimensionality reduction techniques on real-world datasets. On WarpPIE10P and COIL-20 datasets, DynoGraph outperforms all baselines when k is 4%, and the performance gap widens further as the k

increases. DynoGraph exhibits robustness and superior ability to maintain the integrity of data relationships when considering a wider range of neighborhoods. Across most datasets, DynoGraph achieves comparable performance to other state-of-the-art algorithms such as LargeVis, SpaceMAP, t-SNE, UMAP, and TriMap, which are known for their superior preservation of local structures. In contrast, classical algorithms such as PCA, LLE, MDS, and Eigenmaps perform poorly in preserving local structure, resulting in lower accuracy.

5.4.4 Ablation Studies

Graph Construction

DynoGraph employs ANG instead of the standard k-NNG commonly used in many dimensionality reduction algorithms. After replacing ANG with k-NNG (e.g., setting k to 15 as in UMAP), labeled as the variant *DynoGraph_knn*, a notable degradation in performance is observed in Fig. 5.5. This degradation is pronounced in classification tasks on small datasets such as WarpPIE10P, where the fixed k parameter significantly decreases accuracy. In contrast, ANG adapts to the density and size of the dataset, recognizing only up to 8 neighbors, which highlights the limitations of applying a fixed k to various datasets.

Graph Modification

To test the impact of the dynamic graph modification phase on the performance of DynoGraph, we define two variants: *DynoGraph w/o modi*, which removes the modification phase from DynoGraph, and *DynoGraph_knn w/o modi*, which removes the modification phase from *DynoGraph_knn*. Fig. 5.5 shows that the inclusion of a dynamic graph modification phase notably enhances the performance of both DynoGraph and *DynoGraph_knn*, especially on the LandsatSatellite dataset.

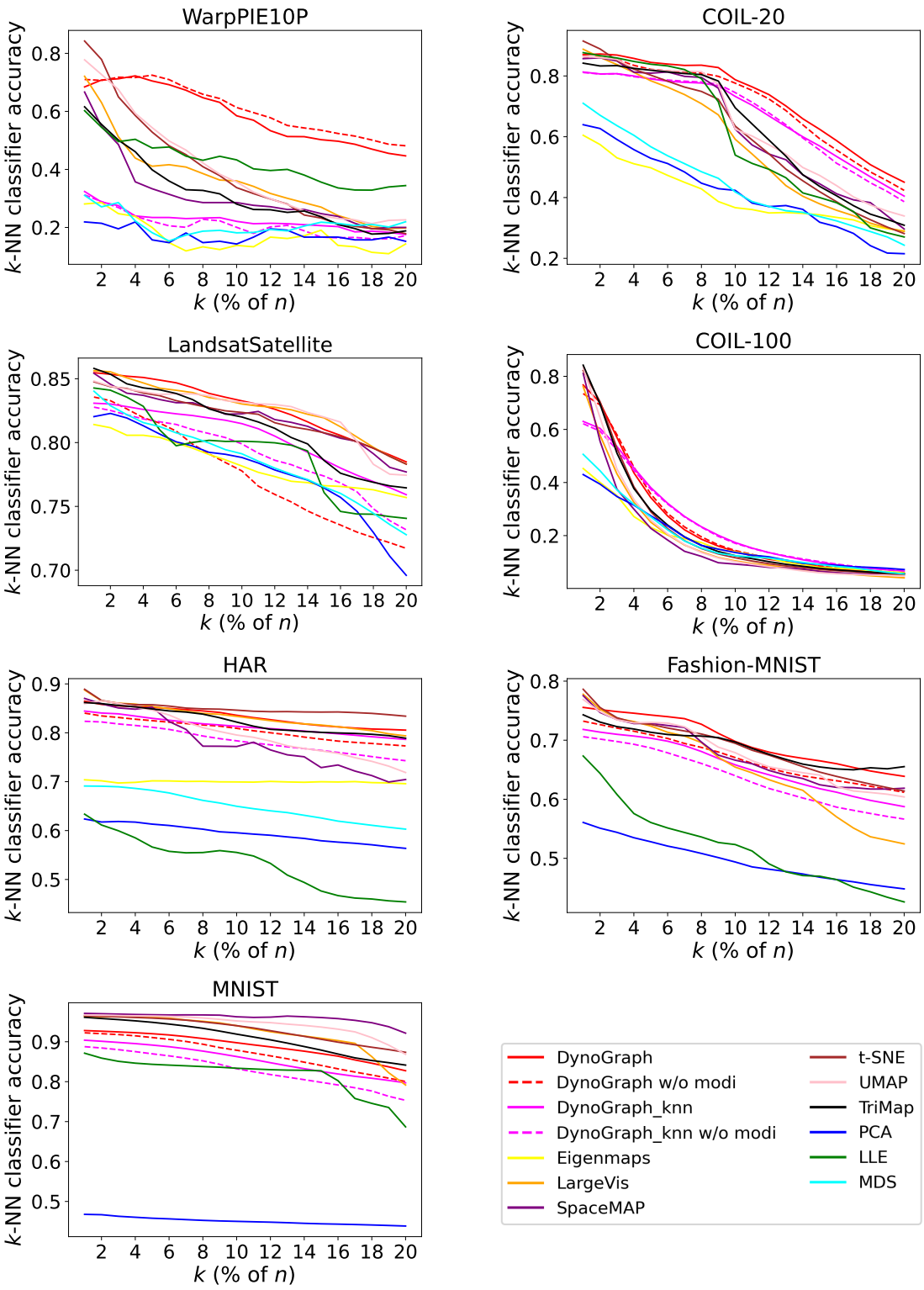


Figure 5.5: k-NN classifier accuracy on real-world datasets.

5.4.5 Visualization on Real-World Datasets

Fig. 5.6 shows the visualization of WarpPIE10P dataset. DynoGraph demonstrates significantly superior performance compared to all baselines. This superiority is reflected in the tighter clustering of points within the same class and the clearer identification of distinct classes, particularly for class 10. Notably, DynoGraph effectively distinguishes class 10, which other algorithms often do not achieve. In many cases, class 10 appears as a linear strip, such as in UMAP, SpaceMAP, LargeVis, TriMap, t-SNE, and LLE, or remains visually indistinguishable, as observed in Eigenmaps, MDS, and PCA.

Fig. 5.7 and Fig. 5.9 show the visualization of COIL-20 and COIL-100 datasets, respectively. DynoGraph exhibits clearer class separation and uniform distribution within classes, with overlap between some classes. In contrast, t-SNE, UMAP, SpaceMAP, TriMap, and LargeVis present a circle-shaped class structure with compact intra-class structures. However, the boundaries between classes are not as distinct as those in DynoGraph and contain more overlap between classes. PCA, MDS, and Eigenmaps lack clear separation between classes and an over-dispersion of intra-class distribution.

Fig. 5.8 shows the visualization of LandsatSatellite dataset. Unlike UMAP and LargeVis, which exhibit compact and well-separated clusters for class 1 and class 2 but overlap in the remaining four classes, DynoGraph distributes the data points of all six classes almost uniformly, maintaining similar shapes across the classes. SpaceMap, TriMap, t-SNE, and MDS face similar challenges in distinguishing these classes. Algorithms like LLE, Eigenmaps, and PCA reveal a triangular linear relationship.

Fig. 5.10 shows the visualization of HAR dataset. Unlike LLE, Eigenmaps, MDS, and PCA, the remaining algorithms can clearly separate class 5. Among these, class 3 and class 4 show overlapping regions in most algorithms, but DynoGraph achieves a relatively distinct boundary between them. For class 0, class 1, and class 2, DynoGraph represents these classes with a more dispersed, circular intra-class structure. In comparison, algorithms like UMAP, SpaceMap, LargeVis, and TriMap produce overly compact, dot-like clusters. In contrast, t-SNE shows smaller inter-class distances with a more scat-

tered overall distribution of data points.

Fig. 5.11 shows the visualization of Fashion-MNIST dataset. UMAP and LargeVis clearly separate class 1 from the other classes. However, LargeVis struggles with class 8, splitting it into three disjoint clusters. DynoGraph performs comparably with other state-of-the-art algorithms, relatively clearly identifying certain classes, such as class 1, 5, 7, 8, and 9. In contrast, others exhibit overlap and are difficult to distinguish, such as class 0, 2, 3, 4, and 6. PCA fails to reveal any clear class structure or separation, while LLE compresses most data points into a narrow region, lacking distinct boundaries between classes.

Fig. 5.12 shows the visualization of MNIST. DynoGraph demonstrates relatively clear classes and inter-class separation, but some scattered points exist between the classes. SpaceMAP and TriMap exhibit a similar problem of scattered points. t-SNE presents non-overlapping classes, with less spacing between classes, and uniform distribution within each class. UMAP and LargeVis produce similar results, showing some overlap between classes, while non-overlapping classes are separated farther apart and have a more compact intra-class structure.

5.5 Conclusion

This chapter introduces DynoGraph, a Dynamic Graph construction algorithm for Nonlinear dimensionality reduction. Our adaptive neighborhood graph captures the intrinsic geometric structure of the original data. For the first time, we propose a dynamic graph modification approach that detects deviations in the proximity of the original data and adjusts the graph structure during dimensionality reduction, ensuring that the low-dimensional embedding aligns with the high-dimensional data. DynoGraph effectively overcomes parameter dependencies and mitigates distortions. Extensive experiments have demonstrated that DynoGraph faithfully reconstructs the original data structure and shows excellent performance in visualization, clustering, and classification tasks.

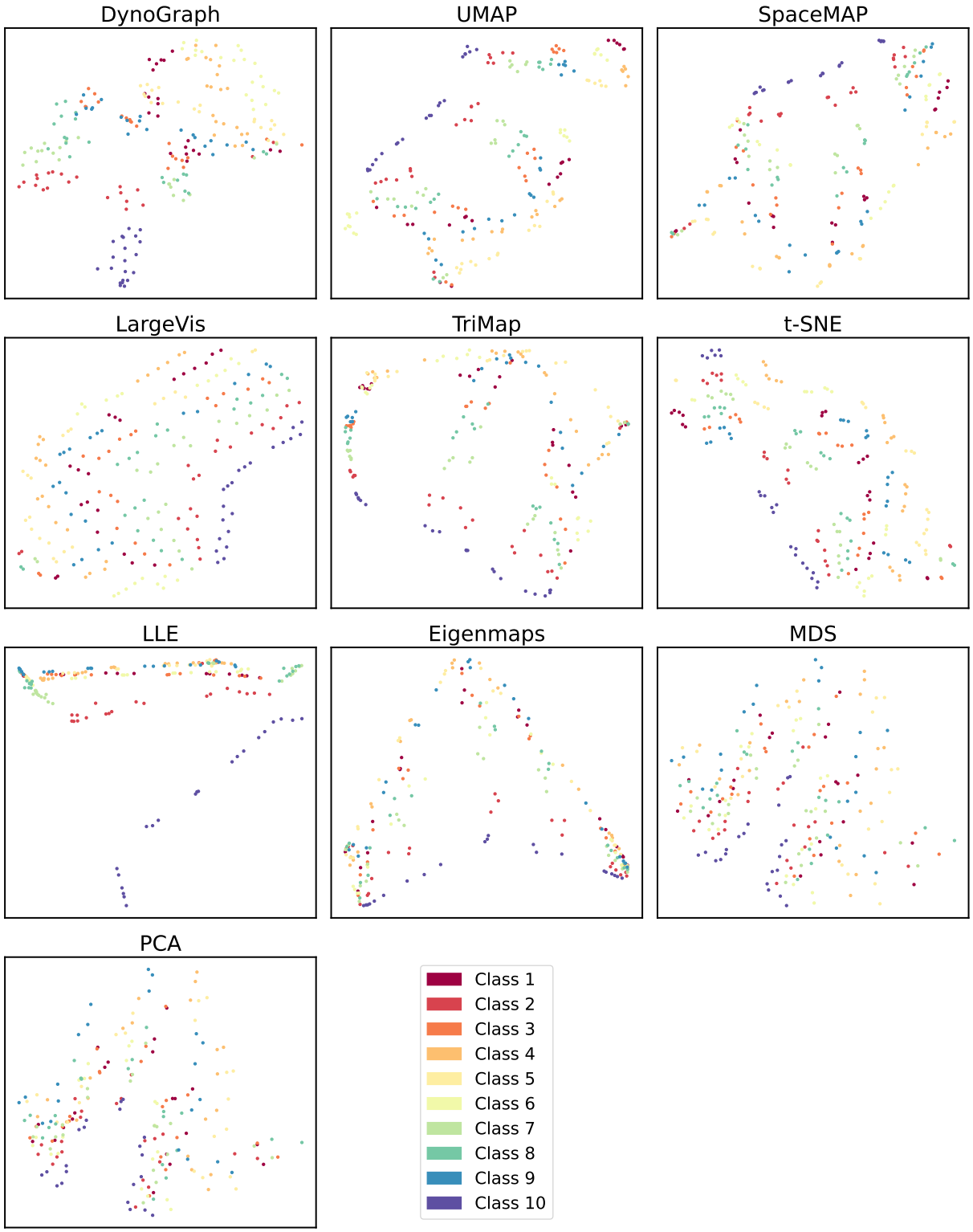


Figure 5.6: Visualization of WarpPIE10P dataset (sorted by average AMI score ranking).

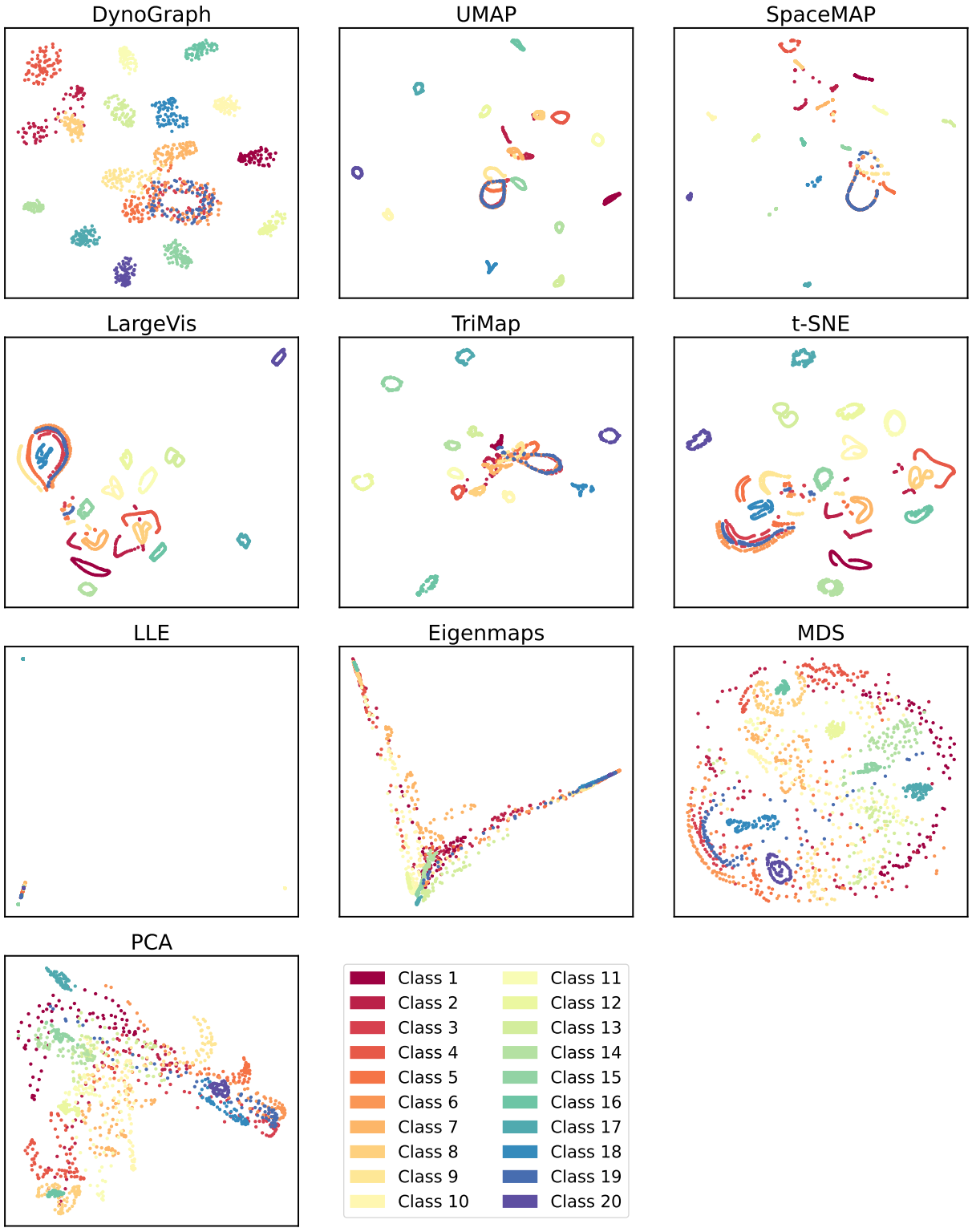


Figure 5.7: Visualization of COIL-20 dataset.

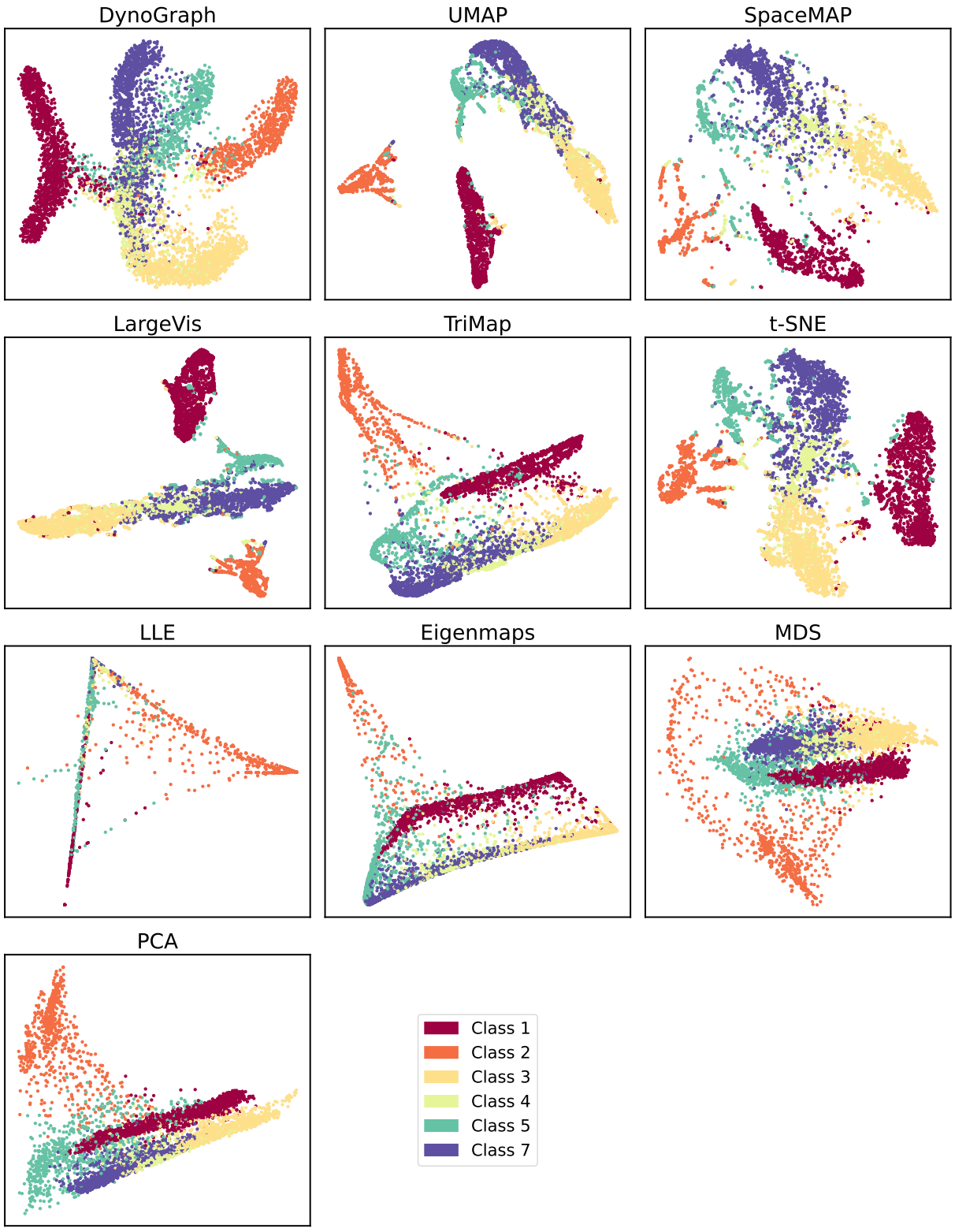


Figure 5.8: Visualization of LandsatSatellite dataset.

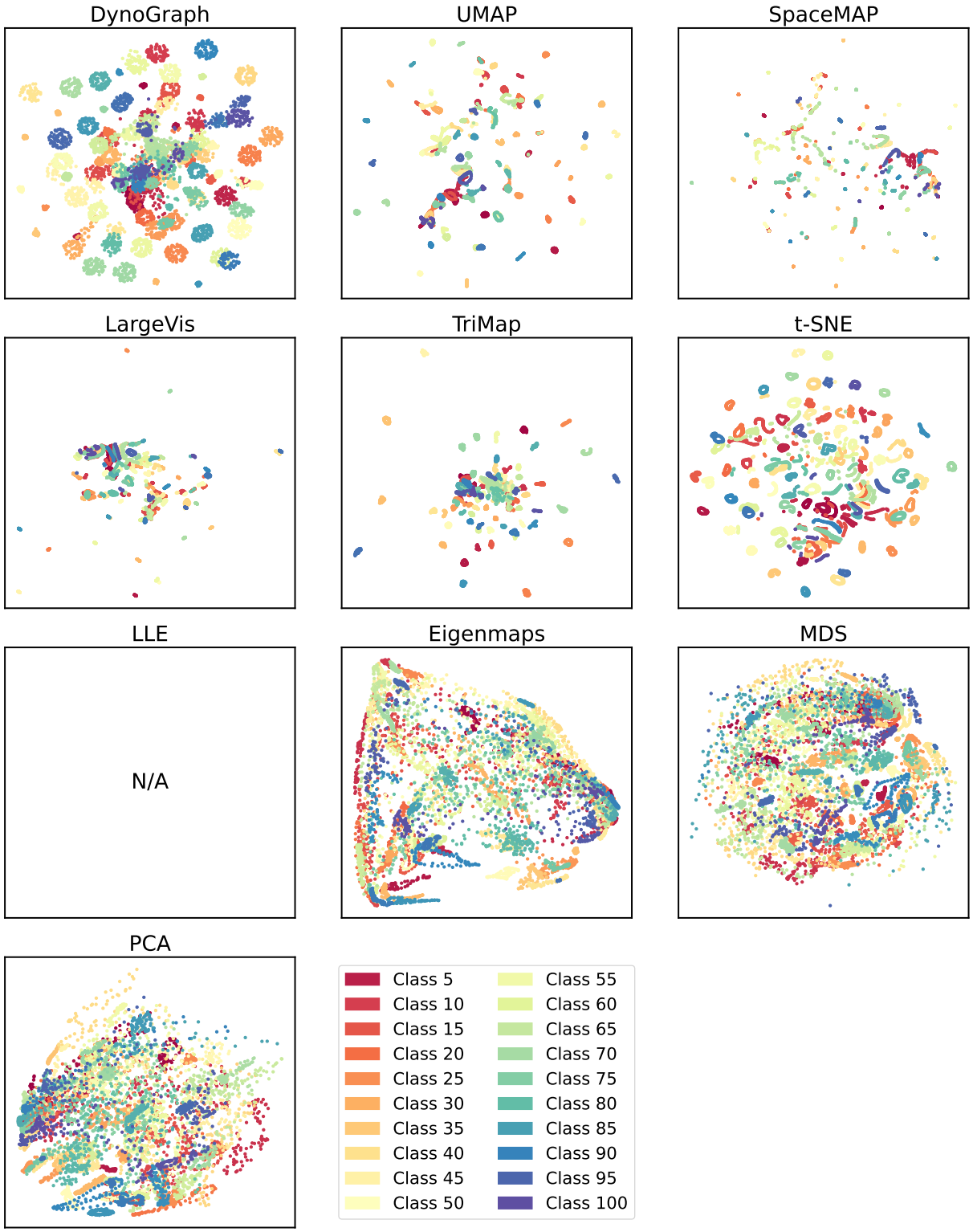


Figure 5.9: Visualization of COIL-100 dataset.

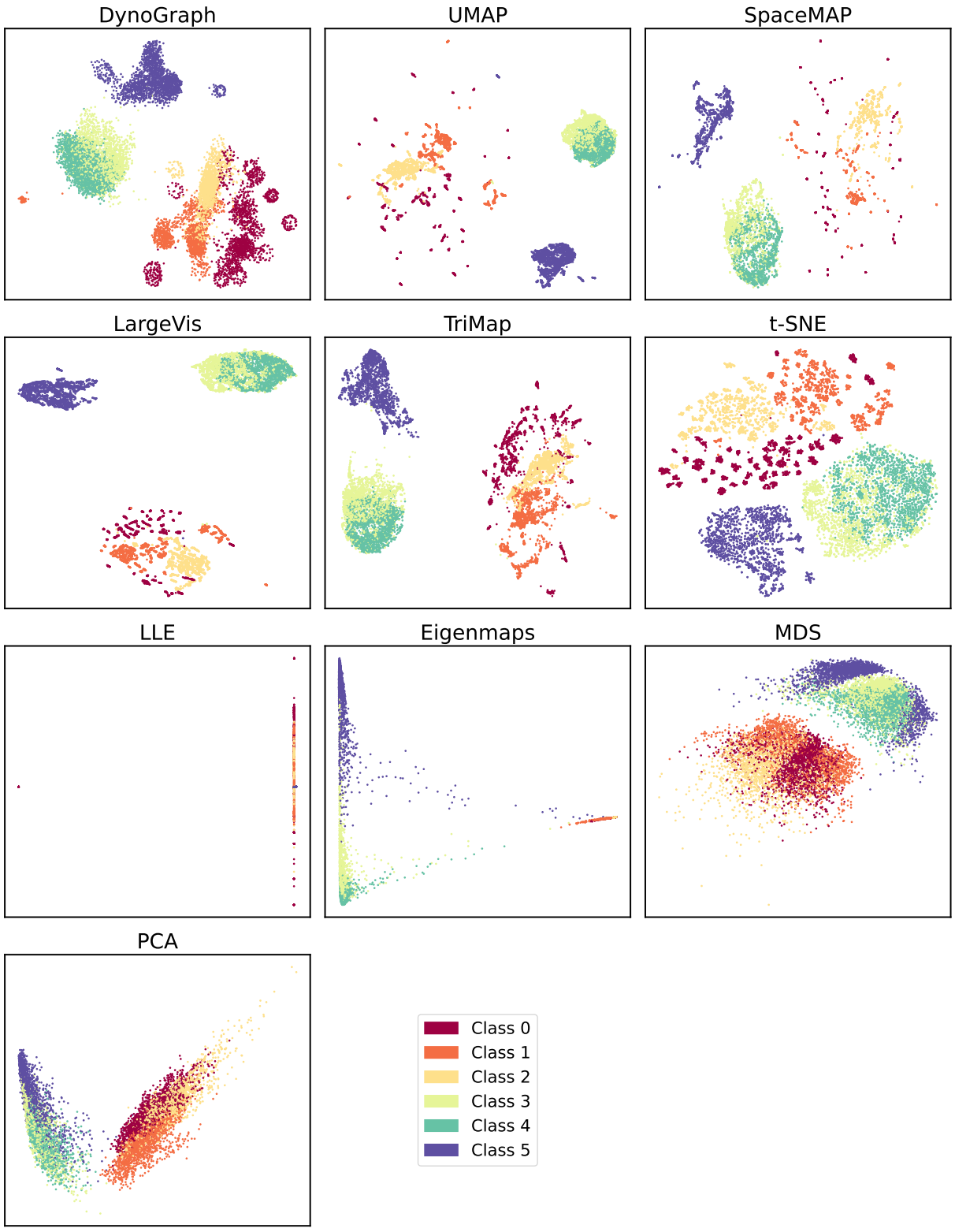


Figure 5.10: Visualization of HAR dataset.

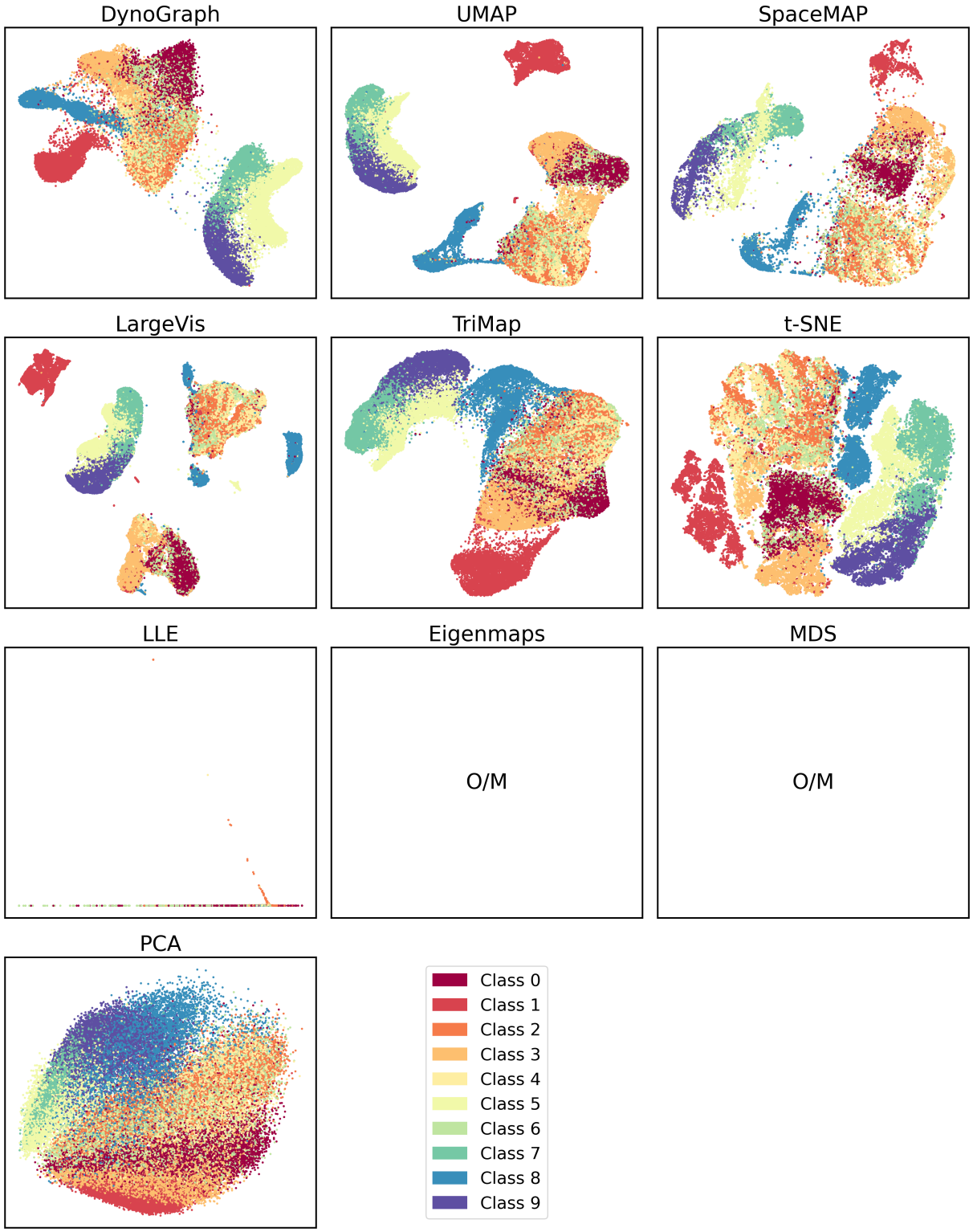


Figure 5.11: Visualization of Fashion-MNIST dataset.

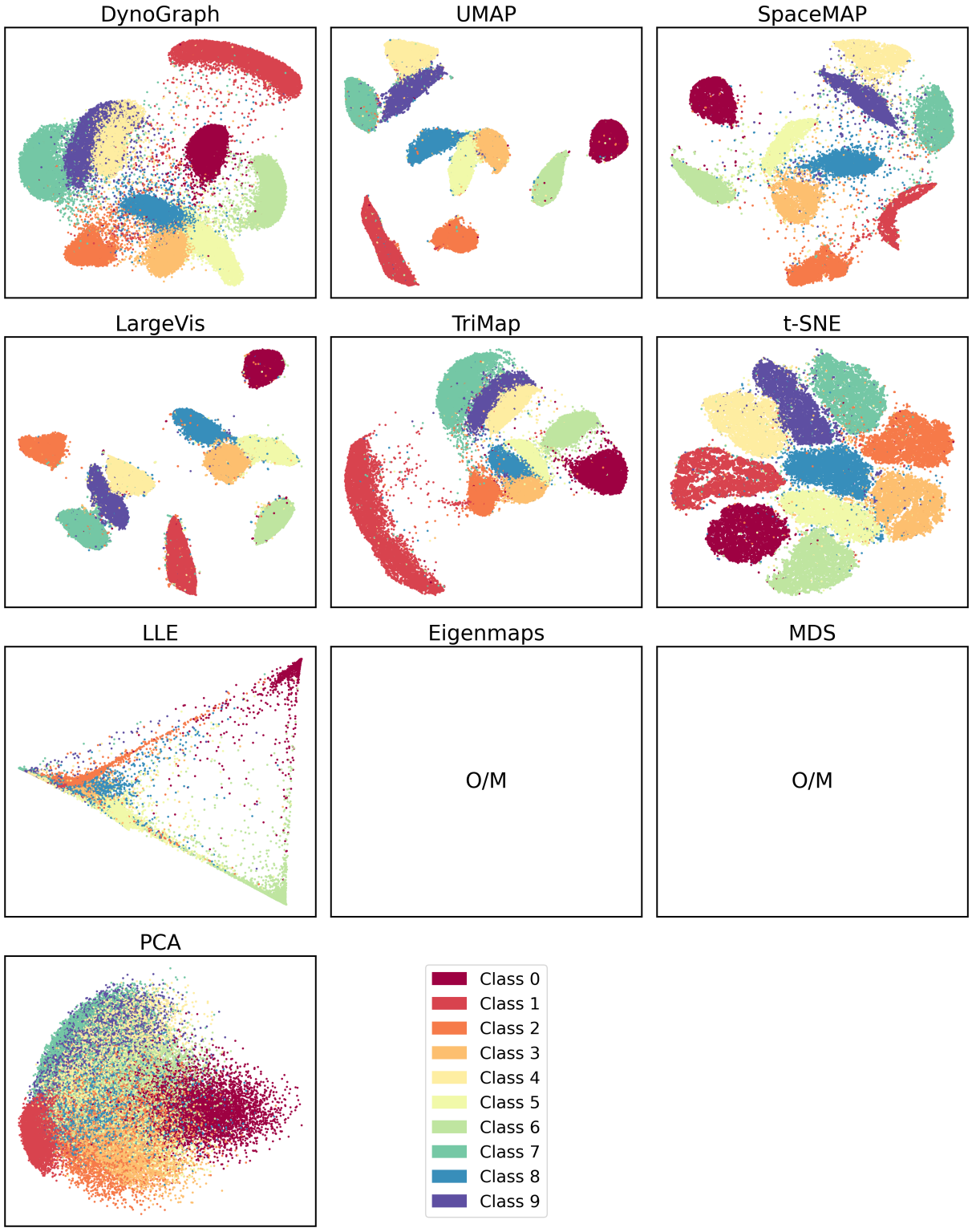


Figure 5.12: Visualization of MNIST dataset.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, our primary goal is to develop an adaptive exploration framework for unsupervised learning by leveraging intrinsic data properties. Specifically, we propose adaptive algorithms for three fundamental unsupervised learning tasks: clustering, outlier detection, and dimensionality reduction. These algorithms focus on adapting to arbitrary data distributions, capturing local density, and designing self-adaptive parameters, thereby addressing the challenges posed by diverse data distributions, density variations, and the dependency on dataset-specific parameters. The main contributions of this thesis are summarized as follows:

Chapter 3 proposes a density-based clustering algorithm, DBADV (Density-Based Adaptive Clustering with Density Variation). First, DBADV is the first clustering method to employ perplexity to derive local density information for each data point, which captures the individual properties of the data points. Second, this information is then used to determine an adaptive search range for each point, which enables the identification of clusters with varying densities. Third, DBADV incorporates a mutual nearest neighbors metric to effectively identify boundary points between clusters and further contribute to distinguishing outliers and noise points from the clusters.

Chapter 4 proposes a proximity-based unsupervised outlier detection algorithm, ADOD (Adaptive Density Outlier Detection). First, ADOD is one of the first outlier detection algorithms to employ perplexity to calculate the local scale of each data point, which is then used to estimate its local density, allowing the algorithm to adapt to density variations. Second, ADOD combines local density with density differences between each point and its mutual neighbors to distinguish outliers that significantly deviate from their surroundings. Third, ADOD uses nearest neighbor search to improve scalability and efficiency for large datasets. Finally, ADOD generalizes to unknown data by comparing it with known data, extending its applicability to real-time scenarios.

Chapter 5 proposes a nonlinear graph-based dimensionality reduction algorithm, DynoGraph (Dynamic Graph Construction for Nonlinear Dimensionality Reduction). First, DynoGraph proposes an adaptive neighborhood graph construction method to capture the intrinsic geometric structure of the data. Second, DynoGraph introduces, for the first time in dimensionality reduction, a dynamic graph modification process to mitigate distortion and maintain the similarity structure of the high-dimensional data. Third, DynoGraph designs an objective function, the Graph Structure Representation Measure, to ensure that the low-dimensional embedding preserves the geometric relationships captured in the graph representation of the high-dimensional data. Specifically, it applies attractive forces to vertex pairs with edges and repulsive forces to vertex pairs without edges. Finally, DynoGraph uses nearest neighbor search to optimize the graph construction process, improving efficiency and scalability for large datasets.

Despite the contributions presented in this thesis, there are still some limitations. For the clustering algorithm DBADV, although the perplexity parameter is less sensitive than the epsilon parameter of DBSCAN, the algorithm still requires two input parameters as in DBSCAN. For the dimensionality reduction algorithm DynoGraph, despite the nearest neighbor search being used in the graph construction phase to improve efficiency, the time complexity of the dynamic graph modification phase is $O(n^2)$, which highlights the need for further optimization to improve the scalability and computational efficiency of the entire algorithm.

6.2 Future Work

There are several promising directions for future research, especially in extending the scope of unsupervised learning and delving deeper into intrinsic data properties.

Extending the scope of unsupervised learning offers several opportunities for future research. One direction is to expand the range of tasks covered by adaptive exploration methods, such as feature selection, feature extraction, and association rule learning. These tasks are often challenged by high-dimensionality, redundancy, sparsity, and complex data distributions, making them ideal candidates for leveraging intrinsic data properties to enhance both the adaptability and performance of methods. Another promising direction is to apply these methods to a wide range of data types, such as graph data, time series data, and high-dimensional sparse data. In addition, there is great potential for optimizing these methods to support real-time processing, which can be applied in dynamic and rapidly evolving environments.

Delving deeper into intrinsic data properties offers several promising directions for future research. An important research focus is to leverage these properties more comprehensively to address additional limitations of existing methods, such as inadequate handling of multimodal data and insufficient adaptability to dynamic or streaming data environments. Strengthening the theoretical foundations of adaptive exploration methods would also be crucial, contributing to their interpretability. In addition, reducing or even completely eliminating the reliance on manual parameter settings is crucial to expand the accessibility and improve the performance of the methods in diverse applications. Finally, optimizing the computational efficiency and scalability of these methods would ensure their applicability to large and high-dimensional datasets.

References

[AA17] Charu C Aggarwal and Charu C Aggarwal. *An introduction to outlier analysis*. Springer, 2017.

[AASM21] Omar Alghushairy, Raed Alsini, Terence Soule, and Xiaogang Ma. A review of local outlier factor algorithms for outlier detection in big data streams. *Big Data and Cognitive Computing*, 5(1), 2021.

[ABC22] Yahya Almardeny, Noureddine Boujnah, and Frances Cleary. A novel outlier detection method for multivariate data. *IEEE Transactions on Knowledge and Data Engineering*, 34(9):4052–4062, 2022.

[ABKS99] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: Ordering points to identify the clustering structure. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 49–60, 1999.

[ABPV+05] Juan A. Acebrón, L. L. Bonilla, Conrad J. Pérez Vicente, Félix Ritort, and Renato Spigler. The kuramoto model: A simple paradigm for synchronization phenomena. *Reviews of Modern Physics*, 77:137–185, 2005.

[AEZS21] Mohamed Abbas, Adel El-Zoghabi, and Amin Shoukry. DenMune: Density peak based clustering using mutual nearest neighbors. *Pattern Recognition*, 109:107589, 2021.

[AHT20] Shaeela Ayesha, Muhammad Kashif Hanif, and Ramzan Talib. Overview and comparative study of dimensionality reduction techniques for high dimensional data. *Information Fusion*, 59:44–58, 2020.

[AP02] Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. In *Principles of Data Mining and Knowledge Discovery*, pages 15–27, 2002.

[ARBM15] Ahmed Shamsul Arefin, Carlos Riveros, Regina Berretta, and Pablo Moscato. The MST-kNN with paracliques. In *Artificial Life and Computational Intelligence*, pages 373–386, 2015.

[AW19] Ehsan Amid and Manfred K. Warmuth. TriMap: Large-scale dimensionality reduction using triplets. *arXiv preprint arXiv:1910.00204*, 2019.

[AWY16] D.A. Adeniyi, Z. Wei, and Y. Yongquan. Automated web usage data mining and recommendation system using k-nearest neighbor (knn) classification method. *Applied Computing and Informatics*, 12(1):90–108, 2016.

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2):281–305, 2012.

[BB22] Fang Bai and Adrien Bartoli. Procrustes analysis with deformations: A closed-form solution by eigenvalue decomposition. *International Journal of Computer Vision*, 130(2):567–593, 2022.

[BCQY97] M.R. Brito, E.L. Chávez, A.J. Quiroz, and J.E. Yukich. Connectivity of the mutual k-nearest-neighbor graph in clustering and outlier detection. *Statistics & Probability Letters*, 35(1):33–42, 1997.

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[BG05] Ingwer Borg and Patrick JF Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.

[BKNS00] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. LOF: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 93–104, 2000.

[BN03] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

[BN06] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Springer, 2006.

[BN13] Leonid Boytsov and Bilegsaikhan Naidan. Engineering efficient and effective non-metric space library. In *Similarity Search and Applications*, pages 280–293, 2013.

[BPSY10] Christian Böhm, Claudia Plant, Junming Shao, and Qinli Yang. Clustering by synchronization. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 583–592, 2010.

[CD21] Pádraig Cunningham and Sarah Jane Delany. k-nearest neighbour classifiers - a tutorial. *ACM Computing Surveys*, 54(6):1–25, 2021.

[CFNTV16] Julien Clément, James Allen Fill, Thu Hien Nguyen Thi, and Brigitte Vallée. Towards a realistic analysis of the quickselect algorithm. *Theory of Computing Systems*, 58(4):528–578, 2016.

[CH67] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[Chi92] Nancy Chinchor. MUC-4 evaluation metrics. In *Proceedings of the 4th Conference on Message Understanding*, pages 22–29, 1992.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[CMZS15] Ricardo J. G. B. Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. Hierarchical density estimates for data clustering, visualization, and outlier detection. *ACM Transactions on Knowledge Discovery from Data*, 10(1):1–51, 2015.

[CPA+20] Benoît Colange, Jaakko Peltonen, Michaël Aupetit, Denys Dutykh, and Sylvain Lespinats. Steering distortions to preserve classes and neighbors in supervised dimensionality reduction. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pages 13214–13225, 2020.

[CPnZ04] Miguel Á. Carreira-Perpiñán and Richard S. Zemel. Proximity graphs for clustering and manifold learning. In *Proceedings of the 17th International Conference on Neural Information Processing Systems*, pages 225–232, 2004.

[CVLD19] Benoît Colange, Laurent Vuillon, Sylvain Lespinats, and Denys Dutykh. Interpreting distortions in dimensionality reduction by superimposing neighbourhood graphs. In *2019 IEEE Visualization Conference (VIS)*, pages 211–215, 2019.

[CY21] Jianguo Chen and Philip S. Yu. A domain adaptive density clustering algorithm for data with varying density distribution. *IEEE Transactions on Knowledge and Data Engineering*, 33(6):2310–2321, 2021.

[DFO20] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning*. Cambridge University Press, 2020.

[DG17] Dheeru Dua and Casey Graff. UCI machine learning repository. `https://archive.ics.uci.edu`, 2017.

[DGD+24] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024.

[DHS12] Richard O Duda, Peter E Hart, and David G Stork. *Pattern Classification*. John Wiley & Sons, 2012.

[dL05] Jan de Leeuw. Applications of convex analysis to multidimensional scaling. *UCLA: Department of Statistics, UCLA*, 2005.

[DML11] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web*, pages 577–586, 2011.

[Dou16] Christopher Dougherty. *Introduction to Econometrics*. Oxford University Press, 2016.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[EMK+21] Mateus Espadoto, Rafael M. Martins, Andreas Kerren, Nina S. T. Hirata, and Alexandru C. Telea. Toward a quantitative survey of dimension reduction techniques. *IEEE Transactions on Visualization and Computer Graphics*, 27(3):2153–2173, 2021.

[ESK03] Levent Ertöz, Michael Steinbach, and Vipin Kumar. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 47–58, 2003.

[Faw06] Tom Fawcett. An introduction to ROC analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[FD07] Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[FPSS96] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37–54, 1996.

[GHNN22] Adam Goodge, Bryan Hooi, See Kiong Ng, and Wee Siong Ng. LUNAR: Unifying local outlier detection methods via graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, number 6, pages 6737–6745, 2022.

[Gil00] Warren Gilchrist. *Statistical modelling with quantile functions*. Chapman and Hall/CRC, 2000.

[GIM99] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529, 1999.

[GKN04] Emden R. Gansner, Yehuda Koren, and Stephen North. Graph drawing by stress majorization. In *Graph Drawing*, pages 239–250, 2004.

[GLH15] Salvador García, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*. Springer, 2015.

[Gow75] John C Gower. Generalized procrustes analysis. *Psychometrika*, 40:33–51, 1975.

[Grü04] Peter Grünwald. A tutorial introduction to the minimum description length principle. *arXiv preprint arXiv:math/0406077*, 2004.

[Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, 1984.

[HAYSZ11] Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, pages 1312–1317, 2011.

[HDHM19] Sibylle Hess, Wouter Duivesteijn, Philipp Honysz, and Katharina Morik. The SpectACl of nonconvex clustering: A spectral approach to density-based clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3788–3795, 2019.

[HG17] Xin Huang and Yulia R. Gel. CRAD: Clustering with robust autocuts and depth. In *2017 IEEE International Conference on Data Mining*, pages 925–930, 2017.

[HHH+22] Songqiao Han, Xiyang Hu, Hailiang Huang, Minqi Jiang, and Yue Zhao. ADBench: Anomaly detection benchmark. In *Advances in Neural Information Processing Systems*, pages 32142–32159, 2022.

[HKP11] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2011.

[HM82] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1):29–36, 1982.

[Hof07] Heiko Hoffmann. Kernel PCA for novelty detection. *Pattern Recognition*, 40(3):863–874, 2007.

[IFFW+19] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[İnk23] Tülin İnkaya. Consensus similarity graph construction for clustering. *Pattern Analysis and Applications*, 26(2):703–733, 2023.

[Jai10] Anil K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[JDJ19] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[JDS11] Herve Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

[JHPvdH12] Jeroen Janssens, Ferenc Huszár, Eric Postma, and HJ van den Herik. Stochastic outlier selection. Technical Report TiCC TR 2012-001, Tilburg University, 2012.

[JK17] Kalervo Järvelin and Jaana Kekäläinen. IR evaluation methods for retrieving highly relevant documents. *SIGIR Forum*, 51(2):243–250, 2017.

[JM15] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[KKSZ11] Hans-Peter Kriegel, Peer Kröger, Jörg Sander, and Arthur Zimek. Density-based clustering. *WIREs Data Mining and Knowledge Discovery*, 1(3):231–240, 2011.

[KR17] K. Mahesh Kumar and A. Rama Mohan Reddy. An efficient k-means clustering filtering algorithm using density based initial cluster centers. *Information Sciences*, 418:286–301, 2017.

[KSZ08] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 444–452, 2008.

[LCB98] Yann LeCun, Corinna Cortes, and Christopher J. C. Burges. The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`, 1998.

[LK05] Aleksandar Lazarevic and Vipin Kumar. Feature bagging for outlier detection. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 157–166, 2005.

[LLX+10] Yanchi Liu, Zhongmou Li, Hui Xiong, Xuedong Gao, and Junjie Wu. Understanding of internal clustering validation measures. In *2010 IEEE International Conference on Data Mining*, pages 911–916, 2010.

[LLZ+20] Yezheng Liu, Zhe Li, Chong Zhou, Yuanchun Jiang, Jianshan Sun, Meng Wang, and Xiangnan He. Generative adversarial active learning for unsupervised outlier detection. *IEEE Transactions on Knowledge and Data Engineering*, 32(8):1517–1528, 2020.

[LTZ08] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008.

[LV07] John A Lee and Michel Verleysen. *Nonlinear dimensionality reduction*. Springer, 2007.

[LWY18] Rui Liu, Hong Wang, and Xiaomei Yu. Shared-nearest-neighbor-based clustering by fast search and find of density peaks. *Information Sciences*, 450:200–226, 2018.

[LZB+20] Zheng Li, Yue Zhao, Nicola Botta, Cezar Ionescu, and Xiyang Hu. COPOD: Copula-based outlier detection. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1118–1123, 2020.

[LZH+23] Zheng Li, Yue Zhao, Xiyang Hu, Nicola Botta, Cezar Ionescu, and George H. Chen. ECOD: Unsupervised outlier detection using empirical cumulative distribution functions. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12181–12193, 2023.

[MHM18] Leland McInnes, John Healy, and James Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[ML14] Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

[MPM07] Antonio Menditto, Marina Patriarca, and Bertil Magnusson. Understanding the meaning of accuracy, trueness and precision. *Accreditation and quality assurance*, 12:45–47, 2007.

[Mur12] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[Mur22] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[MWX+23] Xiaoxiao Ma, Jia Wu, Shan Xue, Jian Yang, Chuan Zhou, Quan Z. Sheng, Hui Xiong, and Leman Akoglu. A comprehensive survey on graph anomaly

detection with deep learning. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12012–12038, 2023.

[MY20] Yu A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020.

[NJW01] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, pages 849–856, 2001.

[NNM96a] Sameer A. Nene, Shree K. Nayar, and Hiroshi Murase. Columbia object image library (COIL-100). Technical Report CUCS-006-96, Columbia University, 1996.

[NNM96b] Sameer A. Nene, Shree K. Nayar, and Hiroshi Murase. Columbia object image library (COIL-20). Technical Report CUCS-005-96, Columbia University, 1996.

[OSKM11] Kohei Ozaki, Masashi Shimbo, Mamoru Komachi, and Yuji Matsumoto. Using the mutual k-nearest neighbor graphs for semi-supervised classification on natural language data. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, pages 154–162, 2011.

[PBB20] Claudia Plant, Sonja Biedermann, and Christian Böhm. Data compression as a comprehensive framework for graph drawing and representation learning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1212–1222, 2020.

[PR02] Seth Pettie and Vijaya Ramachandran. An optimal minimum spanning tree algorithm. *Journal of the ACM (JACM)*, 49(1):16–34, 2002.

[QPB21] Li Qian, Claudia Plant, and Christian Böhm. Density-based clustering for adaptive density variation. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 1282–1287, 2021.

[QPQ+24] Li Qian, Claudia Plant, Yalan Qin, Jing Qian, and Christian Böhm. Dynograph: Dynamic graph construction for nonlinear dimensionality reduction. In *2024 IEEE International Conference on Data Mining (ICDM)*, pages 827–832, 2024.

[QQS+24] Li Qian, Jing Qian, Xin Sun, Wengang Guo, and Christian Böhm. Adod: Adaptive density outlier detection. In *2024 IEEE International Conference on Data Mining (ICDM)*, pages 400–409, 2024.

[QZCS18] Lishan Qiao, Limei Zhang, Songcan Chen, and Dinggang Shen. Data-driven graph construction and graph learning: A review. *Neurocomputing*, 312:336–351, 2018.

[Ray16] Shebuti Rayana. ODDS library, 2016.

[RKV+21] Lukas Ruff, Jacob R. Kauffmann, Robert A. Vandermeulen, Grégoire Montavon, Wojciech Samek, Marius Kloft, Thomas G. Dietterich, and Klaus-Robert Müller. A unifying review of deep and shallow anomaly detection. *Proceedings of the IEEE*, 109(5):756–795, 2021.

[RL14] Alex Rodriguez and Alessandro Laio. Clustering by fast search and find of density peaks. *Science*, 344(6191):1492–1496, 2014.

[RRS00] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 427–438, 2000.

[RS00] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[RY21] R Rawat and R Yadav. Big data: Big data analysis, issues and challenges and technologies. *IOP Conference Series: Materials Science and Engineering*, 1022(1):012014, 2021.

[SB21] Georg Steinbuss and Klemens Böhm. Benchmarking unsupervised outlier detection with realistic synthetic data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(4):1–20, 2021.

[SBB02] Terence Sim, Simon Baker, and Maan Bsat. The CMU pose, illumination, and expression (PIE) database. In *Proceedings of Fifth IEEE International Conference on Automatic Face Gesture Recognition*, pages 53–58, 2002.

[SCSC03] Mei-Ling Shyu, Shu-Ching Chen, Kanoksri Sarinnapakorn, and LiWu Chang. A novel anomaly detection scheme based on principal component classifier. In *Proceedings of the IEEE foundations and new directions of data mining workshop*, pages 172–179, 2003.

[SE14] Alessandro Soranzo and Emanuela Epure. Very simply explicitly invertible approximations of normal cumulative and normal quantile function. *Applied Mathematical Sciences*, 8(87):4323–4341, 2014.

[Sen12] Rico Sennrich. Perplexity minimization for translation model domain adaptation in statistical machine translation. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 539–549, 2012.

[Smi02] Lindsay I Smith. A tutorial on principal components analysis. Technical Report OUCS-2002-12, University of Otago, 2002.

[SPG+17] Amit Saxena, Mukesh Prasad, Akshansh Gupta, Neha Bharill, Om Prakash Patel, Aruna Tiwari, Meng Joo Er, Weiping Ding, and Chin-Teng Lin. A review of clustering techniques and developments. *Neurocomputing*, 267:664–681, 2017.

[SPST+01] Bernhard Schölkopf, John C. Platt, John C. Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

[SS08] György Steinbrecher and William T Shaw. Quantile mechanics. *European Journal of Applied Mathematics*, 19(2):87–112, 2008.

[SSE+17] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. DBSCAN revisited, revisited: why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems (TODS)*, 42(3):1–21, 2017.

[ST23] Durgesh Samariya and Amit Thakkar. A comprehensive survey of anomaly detection algorithms. *Annals of Data Science*, 10(3):829–850, 2023.

[STG+19] Junming Shao, Yue Tan, Lianli Gao, Qinli Yang, Claudia Plant, and Ira Assent. Synchronization-based clustering on evolving data stream. *Information Sciences*, 501:573–587, 2019.

[TCFC02] Jian Tang, Zhixiang Chen, Ada Wai-chee Fu, and David W. Cheung. Enhancing effectiveness of outlier detections for low density patterns. In *Advances in Knowledge Discovery and Data Mining*, pages 535–548, 2002.

[TdSL00] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[TLZM16] Jian Tang, Jingzhou Liu, Ming Zhang, and Qiaozhu Mei. Visualizing large-scale and high-dimensional data. In *Proceedings of the 25th International Conference on World Wide Web*, pages 287–297, 2016.

[VCP16] Patrick Veenstra, Colin Cooper, and Steve Phelps. Spectral clustering using the kNN-MST similarity graph. In *2016 8th Computer Science and Electronic Engineering (CEEC)*, pages 222–227, 2016.

[vdMH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[VEB10] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11:2837–2854, 2010.

[WHRS21] Yingfan Wang, Haiyang Huang, Cynthia Rudin, and Yaron Shaposhnik. Understanding how dimension reduction tools work: An empirical approach to deciphering t-sne, UMAP, TriMap, and PaCMAP for data visualization. *Journal of Machine Learning Research*, 22(201):1–73, 2021.

[XPWW23] Hongzuo Xu, Guansong Pang, Yijie Wang, and Yongjun Wang. Deep isolation forest for anomaly detection. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12591–12604, 2023.

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[YGPB16] Wei Ye, Sebastian Goebl, Claudia Plant, and Christian Böhm. FUSE: Full spectral clustering. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1985–1994, 2016.

[Zhu04] Mu Zhu. Recall, precision and average precision. Working Paper 2004-09, University of Waterloo, 2004.

[ZMP04] Lihi Zelnik-Manor and Pietro Perona. Self-tuning spectral clustering. In *Proceedings of the 17th International Conference on Neural Information Processing Systems*, pages 1601–1608, 2004.

[ZNHL19] Yue Zhao, Zain Nasrullah, Maciej K Hryniewicki, and Zheng Li. LSCP: Locally selective combination in parallel outlier ensembles. In *Proceedings of the 2019 SIAM International Conference on Data Mining (SDM)*, pages 585–593, 2019.

[ZNL19] Yue Zhao, Zain Nasrullah, and Zheng Li. PyOD: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019.

[ZRF+18] Houssam Zenati, Manon Romain, Chuan-Sheng Foo, Bruno Lecouat, and Vijay Chandrasekhar. Adversarially learned anomaly detection. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 727–736, 2018.

[ZT22] Xinrui Zu and Qian Tao. SpaceMAP: Visualizing high-dimensional data by space expansion. In *Proceedings of the 39th International Conference on Machine Learning*, pages 27707–27723, 2022.

[ZTA18] Ye Zhu, Kai Ming Ting, and Maia Angelova. A distance scaling method to improve density-based clustering. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 389–400, 2018.

[ZTC16] Ye Zhu, Kai Ming Ting, and Mark J Carman. Density-ratio based clustering for discovering clusters with varying densities. *Pattern Recognition*, 60:983–997, 2016.

[ZTJA22] Ye Zhu, Kai Ming Ting, Yuan Jin, and Maia Angelova. Hierarchical clustering that takes advantage of both density-peak and density-connectivity. *Information Systems*, 103:101871, 2022.

[ZTY+20] Yaliang Zhao, Samwel K. Tarus, Laurence T. Yang, Jiayu Sun, Yunfei Ge, and Jinke Wang. Privacy-preserving clustering for big data in cyber-

physical-social systems: Survey and perspectives. *Information Sciences*, 515:132–155, 2020.